Dilation Based Bidding Schemes For Dynamic Load Balancing On Distributed Processing Systems††

Taieb F. Znati†, Rami G. Melhem, Kirk R. Pruhs.
Computer Science Department
(† Telecommunication)
University of Pittsburgh
Pittsburgh, PA 15260

Abstract

This paper is concerned with sender-initiated load balancing algorithms for parallel architecture that take into consideration the dilation between the sender and the receiver of the migrated task. The basic scheme proposed in this paper, uses a load contention number that accounts for the dilation among processors. This mechanism is generalized to reflect the specific requirements of different environments. We also describe variations of the basic scheme that aim at reducing the interaction overhead among contending processors.

1 Introduction

The full potential of any distributed processing system can be realized by equally sharing the computational 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 in the literature as *Load Balancing* [1].

Load balancing strategies may be either *static* or *dynamic*. In a static approach, the distributed process is viewed as a collection of tasks each of which can be carried out by any one of the hosts. The tasks and their dependencies are known a priori. Consequently, load balancing can be realized by statically assigning the tasks to different hosts in order to achieve system wide objectives such as minimum average response time, minimum interprocess communication traffic and maximum throughput. Once assigned to a particular process, the task is bound to run on that process until completion [1,2,3]. The problem of finding optimal assignment of tasks to hosts so that some performance criteria is achieved is known to be NP-complete [4]. Several heuristics based on graph theory, queueing theory and network flow theory have been proposed [5]. In general these approaches provide a suboptimal task assignment.

Dynamic load balancing may be modeled by a dynamically created task precedence graph. In this approach, tasks may be spawned dynamically by exploiting concurrent executions embedded in the program. The newly spawned tasks which can be executed on other idle processors may, in turn, spawn more tasks requiring further dispersal of the system load [6,7].

The preformance of the 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 host involved in the process of balancing the load. Different strategies that strike a balance between a high level of performance and low overhead have been suggested in the literature. Most of the existing schemes, however, employ a *centralized model*, a *fully distributed model*, or a *semi distributed model*. This classification is described in Figure 1.

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. Centralized models have the potential of yielding high performance in systems of small to moderate sizes. However, for large systems, the

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

0-8186-2290-3/91/0000/0129/$01.00 © 1991 IEEE

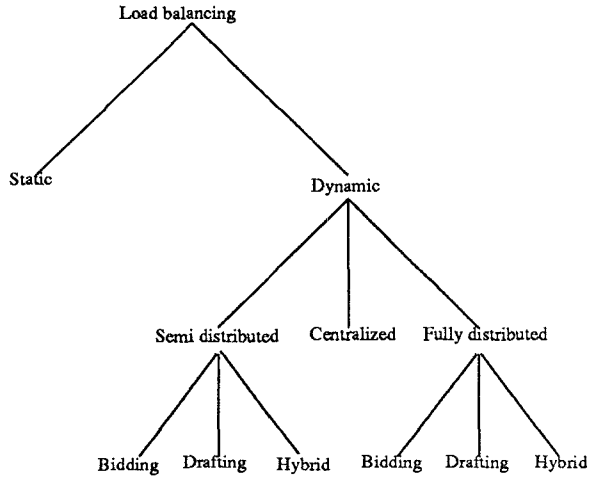

Figure 1. Taxonomy Of Load Balancing Models

computational overhead generated by the need to maintain the global state of the system, and the storage requirements to store the state become prohibitive. The overhead usually causes the controlling node to become the bottleneck of the system. In addition, centralization of the control makes the entire system prone to the central node failure.

In a fully distributed model, nodes of the system build their own views of the system global state, and makes autonomous scheduling decisions based on the information exchanged with other nodes of the system. Several topology dependent and hierarchical schemes have been proposed to improve the performance of the model. In general, however, the communication delays prohibit any node of the system from building an exact view of the global state of the system. Consequently, the decision can only be based on a partial view of the global state, and may lead to suboptimal decisions.

A semidistributed model aims at exploiting the advantages of both centralized and distributed models. This is usually achieved by dividing the 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 performance of this strategy strongly depends on the criteria used to partition the nodes into separate regions.

In an attempt to provide high performance and efficiency, load balancing strategies may adopt a sender initiated approach or receiver initiated approach. In the first strategy, an overloaded sender seeks an underloaded receiver to which some of the excess load may be processed. In the second strategy, underutilized processors search for overloaded processors from which excessive load may be transferred. Analytical and simulated performance studies show that sender initiated policies perform efficiently in an environment with light to moderate load, while receiver initiated policies are preferable in a heavily loaded systems [1,3]. In addition, receiver initiated policies may incur substantial migration cost by requesting the transfer of tasks that only achieved partial execution. Sender initiated policies guarantee that load balancing is only performed when new tasks are spawned, reducing thereby the migration cost to a minimum. This paper is concerned with sender-initiated load balancing algorithms for parallel architecture that take into consideration the dilation between the sender and the receiver of the migrated task. The basic scheme may be adapted to handle a fully distributed or a semidistributed laod balancing model.

In the remainder of this paper, we describe the basic characteristics of the environment and provide a *profitability model* to assess the performance of dynamic load balancing schemes. We then describe a basic scheme to achieve dynamic load balancing for a distributed multiprocessing systems. We finally show how this scheme can be tuned to control the relative emphasis of load and distance, and produce different strategies for load balancing.

2 Environment Characteristics and Assessment Model.

The proposed scheme assumes a loosely coupled large scale multiprocessing system with a number of processing elements interconnected through a broadcast based communication subnet, such as two dimensional spanning bus hypercube, hypertree or cube connected cycles [8]. In addition, we assume that the network communication protocol is completely separated from the inter-task communication policy.

The processing elements are homogeneous, in the sense that a task my be processed at any node of the system. The behavior of the nodes, however, is heterogeneous in that tasks are spawned, terminated or newly generated at arbitrary rates in different nodes. Messages interchanged between different hosts may be of two types, inter-task communication messages, and control messages. We assume that the nodes have the capability of distinguishing between different types of messages when operating in asynchronous mode.

Since messages involve overhead both in time and space, they become the 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 comprised of tasks with high degree of local communication dependencies are allocated to closely connected processors.

Task allocation in dynamic load balancing schemes must include mechanisms to assess the profitability of the decision made with respect to the current system load, determine the migration strategy and select the appropriate task to be migrated [9]. In the next section, we present a formal model of the system and discuss the profitability assessment functions adopted for this study.

2.1 Formal Model

We assume that the topology of the system is represented by an undirected graph, $G=<N,E>$, where N represents the set of nodes, and E the the set of edges. Every node represents a processor, and every edge represents a communication link. We use $\delta(s,d)$ to denote the shortest path between nodes s and d, and Δ to represent the diameter of the graph. Let $U_s(t)$ be the unfinished work at processor s at time t. This represents the computation and communication *load* required by all the tasks assigned to processor s at time t. The current load of a processor dynamically changes as new tasks are forked. For a given task, τ, forked at processor s, $0 \le s < P$, where P is the cardinality of N, we define

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

Two performance criteria may be used to evaluate the performance of the load balancing strategy. The first measures the migration and communication overhead, $C_{s,d}(\tau)$, involved in achieving the migration of a process from a sender node s to a receiver node d. It is clear, that the migration of τ form s to d can only be considered if the cost of processing τ at s exceeds $C_{s,d}(\tau)$. The second evaluates the ability of the strategy to achieve load balancing. In the following sections, we describe the model used to evaluate the migration cost $C_{s,d}(\tau)$ and discuss profitability assessment strategies for load balancing.

2.2 Communication Cost

Based on the specification parameters of the system, we can define the *average cost*, $C_{s,d}(\tau)$, caused by migrating τ from node s to node d as follows:

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

where $M_{s,d}(\tau) = \nu(\tau) \times \mu \times \delta(s,d)$ represents the communication cost, ω represents the overhead required to determine the destination d to which τ is to be migrated, and μ is the cost of sending a message between two adjacent nodes in the network.

2.3 Profitability Assessment Strategies

As stated previously, an important factor which must be taken into consideration in the load balancing strategy assesses the profitability of migrating a given task τ from the source node to another node in the system. This factor can be defined as a *Profitability Assessment Function (PAF)* and relates to the degree of *load imbalance factor (LIB)* at step t, $\phi(t)$. Several metrics have been used in the literature to define $\phi(t)$. Some of the proposed schemes attempt to reduce the difference among unfinished works at different nodes of the system. In this case, the $\phi(t)$ is defined as the root mean square difference in unfinished work average over all hosts [10]. More specifically,

$$\phi(U_1(t),U_2(t),\ldots,U_P(t)) = \sqrt{\frac{\sum_{i \neq j}[U_i(t)-U_j(t)]^2}{P(P-1)/2}}$$

Other metrics aim at reducing the difference between the most loaded processor and the least loaded processor. In this case, $\phi(t)$ may be formulated as

$$\phi(U_1(t),U_2(t),\ldots,U_P(t)) = \{\max_i U_i(t) - \min_j U_j(t)\}$$

In our scheme, we adopt the second metric. Based on the above criteria, the dynamic load balancing algorithm can be conceptually defined as in Figure 2.

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

Figure 2. Dynamic Load Balancing Strategy

The Profitability Assessment Function, *PAF*, can be described as in Figure 3.

It is obvious that some of the parameters involved in the computation of of *PAF*, such as $\nu(\tau)$, may not be available during the time of the decision. However, it is clear that a mechanism that keeps highly interacting tasks within a domain of small radius will preserve the local communication dependencies of the application. In this paper, we propose load balancing strategies that attempts to achieve dynamic load balancing taking into consideration the load distribution among processors, and the local communication dependency requirements of the underlying application. In the next section we provide a descrip-

Function PAF(t , τ, s)
Begin
Find d that minimizes

$$\phi(U_1(t+1),U_2(t+1) \ldots,U_P(t+1)),where$$

$$U_i(t+1) = U_i(t),\text{for} 0 \le i < P, i \neq s, i \neq d,$$

$$U_s(t+1) = U_s(t)+M_{s,d}(\tau),(s \neq d)$$

and

$$U_d(t+1) = U_d(t)+M_{s,d}(\tau)+\pi_d(\tau)$$

if $M_{s,d}(\tau) < \pi_s(\tau)$ **then**
return (d)
else
return (s)
end

Figure 3. Profitability Assessment Function

tion of the proposed basic load balancing algorithm, and proceed to describe variations of the basic algorithm.

3. Basic Dynamic Load Balancing Scheme

The algorithm implements a sender initiated load balancing strategy. The main purpose of the algorithm is to migrate tasks from a heavily loaded processors to lightly loaded processors in the system. The basic scheme insures that the communication overhead, $M_{s,d}(\tau)$, is reduced to a *minimum*, taking into consideration the importance of the communication architecture with respect to the processing capabilities of the processors. This is reflected by the migration condition in the *PAF* procedure. The satisfaction of the above condition ensures that it is more profitable to migrate the task then to process it locally. In addition, since migration will only take place if the recipient of the task has a lighter load, $\phi(U_1(t),U_2(t), \ldots,U_P(t))$ is reduced.

The above objective is achieved using a *Load Contention Number* (*LCN*). The *LCN* is a factor of the current load of the processor and the distance between the sender of the task and the receiver of the task. Based on the *LCN*, the load balancing strategy is reduced to determining the processor currently holding the minimum LCN.

Let s be the processor attempting to migrate a load. At the occurrence of the new task, the originating node, s, computes its *LCN* and advertises it in an attempt to determine the processor where the spawned task may be executed. The search for the recipient of the newly spawned task 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 host the newly generated task by advertizing their own *LCN*. The search continues until the closest least loaded processor is identified. Notice that the termination detection of the bidding process depends on the type of environment supporting the computation. In a broadcast based communication 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 over the communication subnet. 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 contention phase consists in determining the minimum of a set of numbers, and can be achieved in $\log_2(P)$ steps.

3.1. Load Contention Number Generation

In this section , we describe the mechanism used by all processors to compute their contention numbers. We first derive the contention number for the general case. We then proceed to generalize the mechanism to handle different strategies of load balancing.

Let s, $0 \le s < P$, be the identity of the processor where the newly created task originated, and d the identity of any processor contending for the migration of the new task. If the newly generated task was to be migrated to d, the unfinished work of node s, and d, $U_s(t+1)$ and $U_d(t+1)$, respectively, become:

$$U_s(t+1) = U_s(t) + M_{s,d}(\tau)$$

$$U_d(t+1) = U_d(t) + M_{d,s}(\tau) + \pi_d(\tau)$$

where $M_{i,j}(\tau) = (\nu(\tau) \times \mu \times \delta(i,j) + \omega)$ represents the communication cost between node i and node j. It is clear that the migration of the task τ is profitable, if :

$$U_d(t) + M_{d,s}(t) + \pi_d(\tau) < U_s(t) + M_{s,d}(t) \quad \text{(i)}$$

$$\pi_s(\tau) > M_{s,d}(\tau) \quad \text{(ii)}$$

However, for a homogeneous systems, $\pi_s(\tau) = \pi_d(\tau)$. Consequently, the migration of task τ to d is profitable if:

$$U_d(t) + \delta(s,d) \times \nu(\tau) \times \mu < U_s(t)$$

Based on the above observation, for a given process q, currently holding a load $U_q(t)$ and contending to migrate a task from processor s, the contention number $LCN_q^*(s)$ is given by the following formula:

$$LCN_q^*(s) = \frac{1}{\mu} U_q(t) + \delta(s,q) \times \nu(\tau)$$

The load balancing strategy can then be formulated as described in Figure 4.

---

determine processor q that **minimizes**

$$LCN_q^*(s) = \frac{1}{\mu}U_q(t) + \delta(s,q) \times \nu(\tau)$$

If $q \neq s$ **migrate** τ to processor q

Figure 4. LCN Based Load Balancing Strategy

Notice that in the above formula, μ reflects the impact of the underlying communication architecture. In other words, if μ is negligible, the cost is dominated by the load. However, if μ is set to the *Maximum Load* of a processor, the migration cost is dominated by the communication. The parameter $\nu(\tau)$, on the other hand, reflects the degree of task interaction. For a structured computation, $\nu(\tau)$ is uniform among all tasks. In this case, the load balancing strategy will consist in minimizing $\frac{1}{\mu}U_q(t) + \delta(s,q)$, which is equivalent to minimizing $U_q(t) + \mu\delta(s,q)$. In either case, the quantity to be minimized will be refered to as $LCN_q(s)$.

4 Strategies For Load Balancing

In the following sections, we show that the *Load Contention Number* can be tuned to control the relative emphasis of load and distance, resulting thereby in a spectrum of different strategies for load balancing.

It is clear that if $\mu = 0$, the $LCN_d(s)$ generated by a processor d contending for the migration of a task forked at node s is reduced to $U_d(t)$. The strategy resulting from the above contention number, which is frequently encountered in the literature, uses the load as the only criteria to achieve load balancing [3]. 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. In the latter case, $LCN_d(s)$ is reduced to $U_d(t) + \rho \times \delta(s,d)$, where ρ represents the maximum load in the system. An example of the possible contention numbers generated by different processors, $0 \leq d < P$, is described in Table 1. The rows in the table represent all possible processor loads and the columns represent the distance between any contending processor d and processor s, where the newly created task originated. A contending processor, d uses its current load, $U_d(t)$, and its distance $\delta(s,d)$ with respect to processor s, to generate its contention number $LCN_d(s)$. In this example, $\Delta = 4$, and ρ is 10. It is clear from the entries of the table, that irrespective of the load, the processor where the newly created task originated will always produce a smaller contention number.

Load, $U_d(t)$	$\delta(s,d)$				
	0	1	2	3	4
0	0	10	20	30	40
1	1	11	21	31	41
2	2	12	22	32	42
.	.	.	.	.	.
10	10	20	30	40	50

Table 1. LCN: No Migration Strategy

In the remainder of this section, we present other load balancing strategies that aim at striking a balance between the load and the distance in order to satisfy different performance criteria as 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 the tie in favor of the closest least loaded processor in the system. This strategy can be achieved by setting μ to $\frac{1}{\Delta}$. Consequently, the contention number reduces to

$$LCN_d(s) = \Delta \times U_d(t) + \delta(s,d)$$

The resulting contention numbers for the case where $\Delta = 4$ is reported in Table 2.

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

Table 2. LCN: Load Predominant Strategy

The resulting table shows that the load contention number increases as the load of the processors increases. Consequently, the least loaded processors generate the smallest contention numbers. Furthermore, the table shows that if more than one processor holds the smallest load in the system, the closest to the processor where the newly

created task originated will produce the smallest contention number.

4.2 Distance and Load Based Strategy

A closer look at the above scheme reveals that the effect of the distance in the computation of the *LCN* is reduced to breaking the ties among two contending processors currently holding the same load. In some instances, however, it is desirable to add more weight to the factor distance in the migration decision. More specifically, a mechanism whereby a unit of distance equates k units of load is required. The resulting load balancing strategy may obtained by setting μ to k. In this case, the value of the *LCN* becomes $U_d(t) + k \times \delta(s,d)$. An example describing the resulting contention numbers when k=2 is illustrated in Table 3.

Load, $U_d(t)$	$\delta(s,d)$				
	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*
.	.	.	.	.	.

Table 3. LCN : Unit Distance is ≈ to k Units Load

It is clear that as k increases, the weight of the distance in the migration decision increases, and becomes totally predominant when k=*Maximum Load*. In this case, the closest processor will always prevail regardless of the value of the current load at the contending processors. Consequently, no load is migrated between processors resulting into a degenerated load balancing algorithm. In essence, the factor k defines a *load band*, $L, L+k$, such that no gain can be realized by migrating a task from one processor to another, if the two processors are currently holding a load within the band. However, moving from one load band to the upper adjacent one achieves a gain of a unit dilation. A similar observation can be made for the lower range of the parameter k. As k decreases, the emphasis of the load increases, and becomes predominant when $k=\frac{1}{\Delta}$. In this case, the load always prevail over distance, and the latter is only used to break ties between equally loaded contending processors.

4.3 Region and Load Band Based Strategies

As stated earlier, the state based load balancing algorithm tends to improve the system throughput by avoiding idle or lightly loaded processors. However, the interaction cost may still be excessive since all processors are involved in the bidding algorithm. The purpose of the following strategy is to define the notion of *Balancing Region*. For a given processor, a balancing region R, may defined as the set of all neighboring prospective candidates for receiving tasks. Consequently, the bidding process is restricted to the balancing region of every process. This restriction can be easily translated into the load contention number by setting $\mu=\rho R$, resulting in

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

where ρ = *Maximum Load*. Notice 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. An example showing the resulting contention numbers based on

$$LCN_d = U_d(t) + \lfloor \delta(s,d)/R \rfloor$$

for R=2 is illustrated in Table 4.

Load, $U_d(t)$	$\delta(s,d)$			
	0	1	2	3
0	0	0	ρ	ρ
1	1	1	ρ+1	ρ+1
2	2	2	ρ+2	ρ+2
.	.	.	.	.
ρ	ρ	.	.	.

Table 4. LCN: Regions of Influence Based Strategy

For some applications, the profitability of migrating a task from one processor to another may be limited if the difference in the the processors load is small. In this case, it 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

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

Furthermore, using $\left\lfloor U_d(t)/B \right\rfloor$ in $LCN_d(s)$ eliminates the effect of the distance within the load band, resulting in a *threshold based strategy*. An example describing the contention numbers obtained for B=2, and Δ=3 is described in Table 5. The result shows that the loads are grouped in bands of two. Within the same band, only the distance has an effect upon the contention number.

Load, $U_d(t)$	$\delta(s,d)$			
	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 5. LCN: Load Band Based Strategy

4.4 State Based and Node Specified Strategies

As stated previously, dynamic load balancing calls for an optimal task distribution in a configuration space with conflicting demands. The algorithm described above attempts to avoid processor *thrashing* by spreading out tasks evenly over all nodes. In addition, the algorithm takes into consideration the distance separating the sender and the receiver of the task. However, the above algorithm may cause excessive interactions among processors since the load balancing procedure is invoked every time a new task is created. In order to minimize the interaction among processor, state based load balancing algorithms have been proposed [6]. In this case, the load of each processor is determined by the number of tasks currently being served, blocked at a synchronization point, or queued at the processor site. For each processor, we define a *threshold load* to represent the loading condition for a processor such that further addition of tasks to its current load does not lead to any further improvement of its utilization. Based on the instantaneous load, we define the loading states of a processor as *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 advertizes 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 LCP, the heavily loaded processor gradually disperses its load among the set of eligible processors, until the optimal state is reached.

Let L_1-1 be the maximum amount of load a lightly loaded processor may have, and L_2-1 the maximum load optimal processor may accumulate. We also assume that we use the *Load Predominant Strategy* ($\mu=1/\Delta$), as described previously. Consequently, the objective of the strategy would be to select the closest lightly loaded processor for the execution of the newly generated task. This can be accomplished using using

$$LCN_d(s) = \varepsilon_{1i} \times \Delta \times U_d + \alpha$$

where $\alpha=\delta(s,d)\varepsilon_{1i}+((\Delta+1)\times\rho)(\varepsilon_{2i}+\varepsilon_{3i})$, ε_{ij} represents the Kronecker function, and i is determined such that $L_{i-1} \le U_d < L_i, (L_0=0)$.

The resulting contention numbers, when L_1 is 10, and L_2 is 20, assuming $\rho=30$ and $\Delta=3$, are shown in Table 6.

Load, $U_d(t)$	$\delta(s,d)$			
	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
.	.	.	.	.

Table 6. LCN: State Based Strategy

Other variations of the above basic scheme may be obtained based on the the mechanisms described to define the region and to specify the load bands. It is clear that these parameters can be either *Global*, where R and B are fixed for the entire systems, or *Local*, where each node i specifies its own region of influence and its load band, (R_i, B_i). This flexibility may be used to reflect the current status of the processors.

4.5 Efficiency Improvement

It is clear that the cost of the 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 node for the migration of newly generated tasks. During an on-going bidding phase, every node *memorizes* an eventual recipient, *recpt_id*, of "its" future load. In order to determine its future load recipient, node c needs to determine the current load of the bidding processor. This informa-

tion may be easily obtained based on the advertized number. Let s be the processor that initiated the current LCP, and q the current contender. It is clear that $LCN_q(c)$ can be written as

$$LCN_q(c) = \frac{1}{\mu} \times U_q + \delta(s,q) - (\delta(c,q) - \delta(s,q))$$

or

$$LCN_q(c) = Tr(LCN_q(s)) = LCN_q(s) - (\delta(c,q) - \delta(s,q))$$

where $Tr()$ represents an operator that transforms the advertized contention number relative to s into a contention number relative to c.

Based on the above, the procedure node c executes to determine the potential node recipient is described as in Figure 5.

```
Function Future_Recipient()
Begin
  recpt_id = c
  recpt_lcn = Tr(LCN_recpt_id(s))
  while (bidding continues)
    if (Tr(LCN_crnt_bdr(s)) < recpt_lcn) than
      recpt_id = crnt_bdr
      recpt_lcn = Tr(LCN_crnt_bdr(s))
End

when ( New fork )
 start bidding with LCN_recpt_id(my_id)
```

Figure 5. Improved Load Balancing Strategy

Notice that If the load of *recpt_id* does not increase, *recpt_id* remains eligible. If the load of *recpt_id* increases, *recpt_id* forces a bidding phase, to determine the recipient of the load.

5 Conclusion

In this paper, we have developed a general framework to achieve dynamic load balancing. We introduced a basic scheme mechanism for dynamic load balancing strategies. The basic scheme uses a load contention number that accounts for the dilation among processors. We have shown that the basic scheme is flexible and can be adapted to reflect the metrics of basic importance in the system. More specifically, the scheme uses μ, $U_d(t)$, and $\delta_s(d)$, to reflect the influence of the the underlying architecture and the degree of task interaction, taking into consideration the effect of processors load. The different strategies discussed are summarized below in Table 7.

μ				
0	$\frac{1}{\Delta}$	$\frac{B}{\Delta}$	$\frac{\rho}{R}$	ρ
Load Based Schemes	Load over Distance	Band Based Schemes	Region Based Schemes	No Load Balancing Scheme

Table 7. Strategies for Load Balancing

7 References

[1] Livny, M., "The Study of Load Balancing Algorithms For Decentralized Distributed Processing Systems", Ph.D. Dissertation, Wiezmann Institute of Science, Aug. 1983.

[2] Barhen, J., "Combinatorial Optimization of the Computational Load Balance For a Hypercube Supercomputer", pp. 71-80.

[3] Tantawi, A.N., and Towsley, D., "Optimal Load Balancing in Distributed Computer Systems", Journal of the ACM, Vol. 32, No. 2, 1985, pp. 445.

[4] Garey, M.R., and Johnson, D.S., "Computers and Intractability: a Guide to the Theory of NP-Completeness", Freeman, W.H., 1979.

[5] Lo, V., "Task Assignment in Distributed Systems", Technical Report, UIUCDCS-R-83-1144, Department of Computer Science, University of Illinois, Urbana-Champaign, 1984.

[6] Ni, L.M., Xu, C.W., and Gendreau T.B., "Distributed Drafting Algorithm for Load Balancing", IEEE Trans. on Software Engineering, Vol. SE-11, No. 10, Oct. 1985, pp. 1153-1161.

[7] Stankovic, J.A., and Sidhu, I.S., "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.

[8] Gulati, S., Barhen, J., and Iyengar, S.S., "The Pebble Crunching Model for Load Balancing In Concurrent Hypercube Ensembles", Journal of the ACM 1988, pp. 189-199.

[9] Willebeek-LeMair, M., and Reeves, A.P., "Dynamic Load Balancing Strategies for Highly Parallel Multicomputer Systems", Technical Report EE-CEG-89-14, Computer Engineering Group, Cornell University, School of Electrical Engineering, December 1989.

[10] Ksu, C.H., and Liu, J.W.S., "Dynamic Load Balancing Algorithms In Homogeneous Distributed Systems", CH2293-9/86, IEEE 1986.