A Physical Particle and Plane Framework for Load Balancing in Multiprocessors

Navid Imani[1], Hamid Sarbazi-Azad[1,2]

[1] IPM School of Computer Science
Tehran, Iran.
navid_imani@softhome.net

[2] Sharif University of Tech.
Dept. of Computer Engineering
Tehran, Iran.
azad@ipm.ir

Abstract

Different models for load balancing have been proposed before, each of which has its own features and advantages when considered for a specific scenario. Yet, nearly all of the existing techniques have assumed an oversimplified model of the system which is often not the case of the real world. In this paper, a new gradient based algorithm for dynamic load balancing on multiprocessors is proposed. This algorithm is an analogy of a classical physical model of a Particle & Plane system which operates based on the classic laws of physics dictated by the nature.

1. Introduction

One of the important issues concerning the multiprocessors is the problem of how to allocate/reallocate particular tasks to processors to achieve low response times. Generally, two classes of approaches are envisioned in the literature answering the solution to this problem: *Static Mapping* and *Dynamic Load Balancing*.

Given a parallel program with m communicating tasks and a multicomputer with $n<m$ processors, the problem of static mapping is to find a mapping of the tasks to the processors such that the program's execution time be minimized. This problem can be reduced to a sophisticated version of the Knapsack problem and hence it lies in the region of NP-hard problems. Heuristic algorithms for this problem try to find a balanced allocation with minimal communication delays. They are often based on graph partitioning techniques [14] and use modern optimization heuristics such as Simulated Annealing or Genetic Algorithms [3,13]. However, they are designed to be executed off-line and consider neither the actual load situation (due to multiprogramming) at run-time nor the data dependent behavior of the program. The major disadvantage of such algorithms is that they are unable to deal with the dynamic changes in the state of the system such as dynamic task creation and deletion during program execution.

The second class of approaches is designed to adapt the distributed systems where new tasks may enter the system at any time and at any node. Dynamic load balancing proposes a solution to the problem of how to distribute the tasks as evenly as possible to avoid idling nodes and to minimize overall response times. Since the allocation is done at run-time, the basic mechanism is the migration of a task from one node to another which usually means the transfer of a considerable amount of data. Most of the proposals use a decentralized algorithm where load balancing agents at the particular nodes negotiate the possible transfer of tasks from overloaded to under-loaded nodes. Most of the dynamic load balancing algorithms reported in the literature have been put forth to operate on local area networks where all of the processors can be considered as adjacent nodes [5,7,17]. Hence, it is not strange to see that in most of the cases the proposed algorithms have ignored some of the issues like inter-process communication, resource dependency and fault-tolerance which are deemed to be critical in certain multiprocessor systems.

Another reason for the lack of in depth studies on these issues is the fact that considering task and resource dependencies and occurrence of fault in the system makes the model too much complicated to be handled.

In this paper, in an attempt to fill the previous gaps in the study of dynamic load balancing in multiprocessors we propose a dynamic load balancing algorithm which takes into account all the important issues of a real system. In order to evade from the intricacies of modeling a real world system, we envision a physical system as an analogy for the load balancing system. Modeling the physical system is much simpler as it only deals with the evidences we practice in our every day life. This paper can be considered as a base towards a design methodology of optimal schemes for modeling dynamic load balancing systems.

Throughout this paper we use the terms task and load interchangeably to refer to the same entity. In particular, when we are interested in the dynamic nature of the loads as the dependency or affinity to a processor, we use the term task. The word load on the other hand is used when the size of a task is considered.

The rest of this paper is organized as follows. In section 2, we will review the related works in the field of dynamic load balancing. Section 3 mainly deals with a classic problem in physics which will later serve as an analogy for our load balancing system while in section 4 we show that how good the physical model can be mapped to the load balancing problem. In section 5, we propose our load balancing algorithm and derive its equations by inspiration from the physical model. Some heuristics and improvements for the algorithm are also put forth in this section. Finally, section 6 concludes the paper.

2. Related Works

The most common approaches to dynamic load balancing are the nearest neighbor based methods. While this family of load balancing algorithms has been studied early in the literature with concentration on the problem of reaching a balanced state as in [4,5,16], recent works are mostly dedicated to the problem of convergence. Some of the most important load balancing algorithms are the *diffusion method*, *dimension exchange method* and *gradient-based method* [6,10,12,15].

In the diffusion method, each processor of the system balances the total quantity of load on itself with the immediate neighboring nodes, i.e. the extra loads of a processor are distributed evenly between the local neighboring processors. Under the synchronous assumption, the diffusion method has been proven to converge in polynomial time for any initial workload distribution given the quiescent assumption that no new workload is generated and no existing workload is completed during execution of the algorithm [1,6]. Also optimal parameters that maximize the convergence rate have been derived on the mesh, torus, and n-D hypercube [19].

In the dimension exchange method each processor balances its loads with its neighbor's one at a time. It has been proven that on a hypercube, the entire system is balanced when every processor has exchanged workload with all its neighbors once [6].

In the gradient based method the workload variations in the system are maintained in gradient maps for each processor in the system. Tasks are moved toward the processors with the steepest gradient. In the gradient model (GM) method, a pressure surface that represents the propagated pressure of the workload is defined [12]. In the contracting within a neighborhood (CWN) method, on the other hand the workload index is used directly and the tasks are sent to the processor with the smallest index [15]. Various stochastic techniques such as randomized methods and simulated annealing and algorithms based on evolutionary algorithms have been researched extensively in the literature [18,19]. Load balancing using the models inspired from physics has also been studied in [8,9,11].

Most of the work mentioned above, have not considered data dependencies between the tasks, resource constraints or the probability of the occurrence of fault in the links or nodes of the network. The algorithm proposed here, however, takes into account all of the mentioned issues by analogy from a classic problem in dynamic physics.

3. Particle and Plane Problem in Physics

In this section, we consider a classic problem in dynamic physics, which we will later show that despite its simplicity it can be well adapted to serve as a good model for load balancing in multiprocessors. Our problem of interest is the motion of an object on a bumpy plane under the physical laws governing the environment.

3.1. System Scenario

Let us assume a box is placed on a bumpy yard which consists of high positions or *hills* and low points or *valleys*. Let us represent each position in the yard with an ordered triple (x,y,z) corresponding to a mapping of that point to a $3d$ Cartesian space. Further, we assume that an object is initially placed on a hill so that it may gently slide down towards a valley. As the object traverses its path down to a proximate valley its velocity increases and hence its kinetic energy grows, while it loses its potential energy. As the object reaches the bottom of the valley it poses kinetic energy and thus its inertia make it climb up the steep towards the peak of the hill on its way. If the hill is not much steep and high and the object has enough velocity (and kinetic energy as a result) then it may reach to the peak of that hill and take its path down to another neighboring valley or otherwise the object bounces back towards the bottom of the first valley and oscillates there until after some time, it stops at the bottom of valley.

The yard's surface and the surface of the object are not smooth; thus there exist some friction between the object and the face of the yard which causes the object in motion to lose some of its velocity. We denote this kind of friction force by f_k which depends on a constant value which itself is representative of the roughness properties of the surfaces (of the object and the yard). Friction also shows itself when the object is stationary (at the start of the game) and resists the gravity forces which are tending to move the object down the slope. Therefore, the object moves only if the slope is steep enough. We call this force the static friction, f_s, which is again a function of a constant μ_s and the Natural force N which ground applies to the object as a result of the third principle of Newton.

The force f_k causes the object to lose some of its kinetic energy while moving. Thus, as the object makes more distance from its origin, its overall energy (kinetic and potential) decreases; this guarantees that the object does not go too far from its initial position. It also lets the object be trapped in a local valley; for once it could not climb up the face of the fronting hill and bounced back to the valley, it might not have enough energy to climb the hill from which it used to come down.

3.2. The Exerted Forces

At the start of the game, the net force exerted to the object initiates its movement down the slope. Let $\overline{f_i}$ be the different forces that act on the object. Then, the question of whether or not the object moves and if so in which direction it is ought to move can be answered after the calculation of the net force as $\overline{F} = \sum \overline{f}_i$. The direction of the force $\overline{F}$ gives the direction along which the object will move. Let an object of mass m be placed on a slope which builds an acute angle of α with the perpendicular line. f_i forces, as fig.1 shows, generally encompass two types of forces: the maximum friction force which for the system can be derived as $f_s = \mu_s mg \sin\alpha$ and the thrust force i.e. the force of gravity along the slope which can be calculated as $f_+ = mg\cos\alpha$. Parameters f_s and f_+ are exerted to the object in opposite directions along the slope and hence the criteria for the movement of the object is $f_s < f_+$. After replacing the previous equations for f_s and f_+ in this inequality we get $mg\cos\alpha > \mu_s mg\sin\alpha$ or

$$\tan\alpha < \frac{1}{\mu_s} \qquad (1)$$

μ_s is solely related to the environment and the object and hence is constant for the whole life time of the system; this fact implies that the important factor which determines the object's motion is the steepness of the slope, $1/\tan\alpha$. More formally, there exists some α_t for the system corresponding to the threshold value for the angle α, $\forall \alpha \geq \alpha_t$, where force $\overline{F}$ would not be enough to initiate the movement of the object.

Suppose that the surface is steep enough to cause the object to move then at any given instant when the object is moving, the net force exerted to the object determines the possible direction of acceleration of the object. The net force vector along the steep again is composed of two forces of kinetic frictional force f_k which can be calculated as $f_k = \mu_k mg\sin\alpha$ and the force of gravity along the steep i.e. $mg\cos\alpha$. The forces which act on the object in motion are depicted in Fig.1.

3.3. The Energy Model

According to the laws of conservation of energy, the overall energy of a system is constant over time while different forms of energy can be converted to each other. Each physical object at any instant has two types of energy, namely Potential Energy and Kinetic Energy. The kinetic energy, E_k, is a direct consequence of the velocity of the object and can be calculated as

$$E_k = \frac{mv^2}{2}$$

where v and m denote the magnitude of velocity and the mass of the object, respectively. This equation implies that a stationary object has no kinetic energy.

The potential energy E_p on the other hand, is the amount of energy stored in the object due to its position related to the base level and equals by the amount of the energy that would be released if the object be moved by the gravity force to the base level. Thus, the potential energy of an object at a height h of the base level is $E_p = mgh$; Hence, the total energy of the object would be

$$E = E_k + E_p = \frac{mv^2}{2} + mgh$$

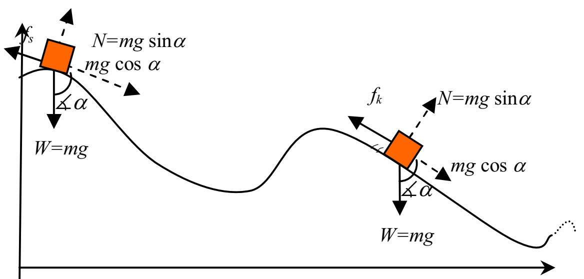

Fig.1. The exerted forces to an object on the hill

For the system under discussion, the total energy of the object E derived in this equation determines whether or not the object can escape a local minimum (valley).

As there exists friction between the object and the surface, when the object moves, the friction force f_k tries to decrease its velocity, i.e. in the presence of f_k it wastes some of its kinetic energy in the form of heat in the environment (see fig.2). Let us assume that μ_k is constant over time; the total energy that is lost due to friction in a path of distance d on a slope can then be calculated as

$$E_h = f_k d = \mu_k N d = \mu_k mgd \sin\alpha = \mu_k mgd^{\perp}$$

$d^{\perp}$ is the perpendicular distance between the source and the destination. This equation implies that when the object moves from a source to a destination on a steep slope, the energy which is wasted in the form of heat equals the energy which is lost as if the object were to move on a flat surface between the same source and destination.

Let us assume that the object is placed at the top of a hill with a height h_0 at the time 0. Let h_t and v_t be the height of the object's position and the velocity of the object at a given instant t, respectively. Furthermore, Let $E_{h,t}$ and $E_{T,t}$ denote the amount of energy that object has lost due to friction from the time $t-1$ to t for a given instant t and the total energy of the object at t, respectively.

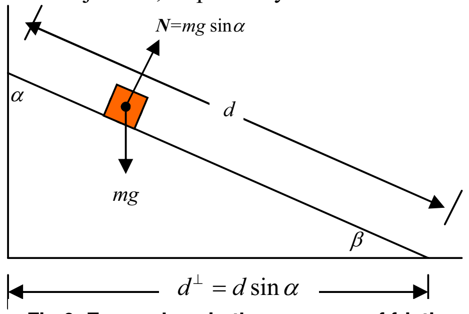

Fig.2. Energy loss in the presence of friction forces.

Then the total energy of the object at the instant t can be calculated as

$$E_{T,t} = E_{T,t-1} - E_{h,t} = mgh_{t-1} + \frac{mv_{t-1}^{\ 2}}{2} - E_{h,t}$$

knowing that

$$E_{h_0,0} = mgh_0$$

The height of the highest point where the object can be at time t, i.e. the *potential height* of the object at time t, is derived as

$$h^*_t = \frac{mgh_0 - \sum_{i=1}^{t} E_{h,i}}{mg} = h_0 - \frac{\sum_{i=1}^{t} E_{h,i}}{mg}$$

Definition 1: We say an object is *trapped* inside a contour c at a given time t if it cannot exit that contour at any given time $t' > t$.

Definition 2: Given a contour c, the *Peak* of c, P_c, is defined as the maximum height of any point within c.

Definition 3: Given a contour c and a position $p(x,y)$, the *Escape Radius* of c at p, $r_{c,p}$, is defined as the minimum distant of p from a point p' outside c on xy plane. The escape radius of p has been depicted in fig.3.

The following two corollaries are the obvious results of the definitions 1-3.

Corollary 1: If $\mu_s = \mu_k = 0$ then for any given contour c satisfying the condition $p_c < h_0$, the object is not trapped inside c at any given time instant.

Corollary 2: If $\mu_k \neq 0$ then there exist some contour c and time instant t such that the object is trapped inside c at the time t.

Theorem 1: Let the object be in a position p at a instant t then the object is not trapped in a contour c at t if $P_c \le h^*_t - \mu_k r_{c,p}$.

Proof. At time t the total energy of the object is $mgh_0 - \sum_{i=1}^{t} E_{h,i}$. The object is not trapped in c if despite the energy it looses due to the friction it can still afford to climb the hills of c on the shortest path. The wasted energy can be calculated as $\mu_k mgr_{c,p}$. Let us assume that the object exits c at $t + t_e$. The total energy of the object at $t + t_e$ would be

$$E_{T,t+t_e} = mgh_0 - \sum_{i=1}^{t} E_{h,i} - \mu_k mgr_{c,p}$$

If we can guarantee that the object with energy $E_{T,t+t_c}$ can climb the highest hill in c, we have shown that the object is not trapped in c. Therefore, we want $P_c \le h^*_{t+t_c}$ while we have

$$h^*_{t+t_e} = h_0 - \frac{\sum_{i=1}^{t} E_{h,i} + \mu_k mgr_{c,p}}{mg} = h_0 - \mu_k r_{c,p} - \frac{\sum_{i=1}^{t} E_{h,i}}{mg} = h^*_t - \mu_k r_{c,p}$$

and hence $P_c \le h^*_t - \mu_k r_{c,p}$.

Theorem 1 implies that the farther the object goes the smaller the height of the hills it can climb; this property prevents the object from getting too far from its initial position.

Corollary 3: Given an object in a position p at a time t, the object is trapped in any contour c whose escape radius satisfies $r_{c,p} > \frac{h^*_t}{\mu_k}$.

Proof. According to the inequality given for the P_c in theorem 1, if we have $r_{c,p} > \frac{h^*_t}{\mu_k}$ then we reach $P_c < 0$, which is impossible. This can be interpreted as the fact that if μ_k and $r_{c,p}$ are big enough, then regardless of the hills that the object may not be able to climb the object looses all its energy as heat due to friction and will stop at the bottom of some valley within *c*.

4. Analogy of the Physical Problem in Load Balancing

In this section, we draw an analogy of the physical scenario discussed in section 3 to model the problem of dynamic load balancing. This study is interesting in the sense that many of the rather virtual concepts in load balancing can be well represented with tangible physical evidences drawn from the real would. As we may show later this analogy may help us to put forth new solutions to some of the problems in load balancing. While the proposed solutions in the physical model may seem trivial, the corresponding solutions in load balancing could have been considered out of reach without considering this analogy.

4.1. From Physical Parameters to Load Balancing Concepts

Given a set of processors connected via an interconnection network and a set of tasks of a program assigned to each processor, the problem of load balancing is how to redistribute the tasks of the processors over the network to other processors with the minimum communication cost so that the execution time of the program minimizes.

Let us assign to each node *v* in the network a number l_v indicating the quantity of load assigned to that processor. Furthermore, let us map each node of the network graph to a point in 2D space via an arbitrary mapping function $M_2 : V(G) \to \mathbb{R}^2$. Such a mapping function always exists for any G ; more specifically, if G is planer there exists a mapping function which let us draw the edges between the nodes as non-crossing lines between the corresponding points in the 2D plane.

Taking into account the load quantity of the nodes, the network can be mapped via $M_3 : V(G) \times L \to \mathbb{R}^3$ to a 3D space, such that each given node *v* can be identified via an ordered triple (*x*,*y*,*z*) where the first two elements identify the node in the network and the *z* element determines the quantity of load on *v*, i.e. $M_3(v,l_v) = (M_2(v),l_v)$.

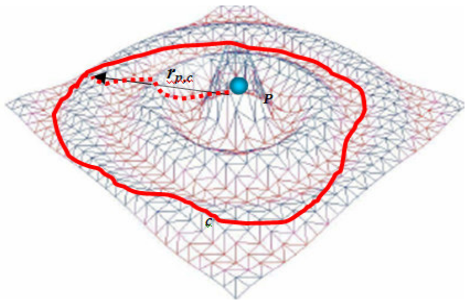

Fig.3. The escape radius of a point *P* from a contour *c*.

Hence, the network can be modeled as a discrete 3D manifold (surface), where the slopes of the manifold contribute to the differences in the load quantities of two neighboring nodes.

We can further consider a load l_v assigned to a processor in a node *v* as an object placed in the point $M_3(v,l_v)$ of the manifold. Each object has a mass *m* which is representative of the quantity of the corresponding load (either in terms of computational complexity or memory size of the task), i.e. a more complex task is presented as a heavier object.

Considering the previous assumptions, now we refine our previous physical scenario so that it adapts to our dynamic load balancing problem. We assume the surface (yard) to be a dynamic surface meaning that the hills and valleys of the surface may change their height over the time as the loads are transferred between processors (this matches well the exchange of loads between the nodes). Initially, some objects are placed on some positions of the surface which are not necessarily flat. We also presume that the objects can only move in the direction leading to a neighboring node, i.e. all the positions in proximity in directions other than the neighboring nodes have infinite height. As the clock ticks the start of the game, the objects that were previously held in order to avoid their movement are released. Each object in the system operates based on the laws of physics as in the simple scenario we envisioned before. The dynamic property of the surface comes into play when an object moves from a hill down to a valley. In such a situation, the hill shrinks its height while the depth of valley shrinks as well. The magnitude of this shrinkage depends on the mass of the transferred object.

Theorem 2: The load balancing scheme presented in this section converges to a nearly perfect load balance.

Proof. To prove this theorem we show that 1) Each load transfer is completed by a known time bound t_{max} and 2) Each load transfer takes the system to a more balanced state. To justify our first argument we consider the physical model. As in the physical model of our load balancing system parameter μ_k is not zero, we can conclude from corollary 2 that there exist some contour *c* and time instant t_{max} such that the object is trapped in *c* at the time t_{max}. To show that after each load transfer the system moves into a more balanced state. We should consider that under no circumstances a load from a more overloaded node would send and reside in a less overloaded node. As the object can never climb a hill higher than its last position, if the difference of the load quantities between two nodes is large enough to compensate the communication delay, congestion and other factors which are modeled as static friction, the load transfer will be initiated, and as was shown in the last two arguments, this transfer after a limited time takes the system to a more balanced state. Thus, after some iterative load balances the system should reach equilibrium which corresponds to a near optimal load balancing solution. The fact that the system reaches the equilibrium is a result directly inferred from the laws of physics.

In load balancing while we are always interested in a perfect distribution of loads, this ideal goal may cost us too much due to the communication delay, in a way that sometimes we rather prefer to ignore the load balancing completely. This can be modeled physically as the presence of static friction force. Static friction force hinders the object from movement if the slope is not steep enough. That is if the tangent of the angle which the slope builds with horizon as tan β= cot α, in fig.2 is less than a specific value of μ_s the object won't move. This is the result derived in equation 1.

The constant μ_s, then can be interpreted as the dependency of a task to a node which can be due to the dependency of the task to other tasks or resources in that node or in the nodes in its proximity.

Although a solution for distributing the loads over a global scope of the network may result in a better load distribution, it demands the task to pass multiple hops over the network to reach its final destination which not only imposes a high communication delay to the system but also increases the congestion in the network. It is why most of the successful dynamic loads balancing algorithms act locally, i.e. each given node exchanges its additional loads with immediate neighbors or with the nodes k hops away. This limitation for transferring loads to far away nodes is modeled in our physical scenario with the kinetic frictional force, f_k.

As discussed in the physical scenario, when the object moves along the surface, as a result of the friction it loses some of its kinetic energy mostly in the form of heat. The object is then more tended to be trapped in a local minimum in proximity, unless there is a deep valley around. The analogy of a system in the presence of kinetic friction in load balancing is that a node's additional loads are more tended to be assigned to the local neighbors, and only if no local nodes are under-loaded it will be sent to farther nodes.

Finally, the heat produced in the environment due to the friction between the objects and the surface can be interpreted as the traffic generated as a result of the transport of loads in the network. There are multiple pieces of evidence that affirm this choice. The produced heat is a function of the mass of the object, a constant μ_k and the lengths of the path that the object takes. On the other hand, in a network, the traffic stems from the balk of the loads being sent over the network (mass's analogy), the probability of staying in a node (this probability increases with the increase of μ_k, as a high value of μ_k may exhaust the object so that it can not escape the local minimums) and the time it takes for a task to be received at the destination which is again proportional to the length of the path. The mapping of the physical parameters as defined in the object model to load balancing concepts is shown in table 1.

4.2. Modeling the System Parameters

In order to model the system more accurately, we define each of the discussed physical parameters as a function of some primary load balancing parameters.

Let G (V,E) be the graph corresponding to the interconnection network which is connecting the processing nodes together in a multiprocessor. We denote each given node of the network with v_i and each link between two given nodes of v_i and v_j with $e_{i,j}$. The quantity of the load assigned to a node v_i of the network is identified by $l_{i,k}$. To model the system more accurately, we further assume that each task may be dependant to some other tasks in the system. This dependency can be due to the fact that the former task may need the results of the other task and hence some communication may be necessary. We model these dependencies between the tasks with a task graph T whose vertices are the tasks labeled by their load quantity and the edges represent the dependency relations between the tasks. The edges have different weights which model the amount of communication between two tasks. Hence, $T_{i,j}$ denotes the dependency of a task i to another task j. The tasks can also be dependent to a node due to the need for the resources which are present in that node. We show these dependencies with another matrix $R_{|L|\times|V|}$, where $|L|$ is the number of tasks in the system.

We assume each link in the network has a known bandwidth, length and fault probability, i.e. the probability of occurrence of a fault in a time unit. All of these parameters are constant over the life time of the system. They are considered as the configuration parameters of the system which describe the overall system's attributes. We model these link parameters with BW, D, F matrices each of dimensions $|V|\times|V|$. Hereafter, we use $X_{i,j}$ (or equivalently $x_{i,j}$) to refer to an element (i,j) of a matrix X.
Now that we defined the primary load balancing parameters we can describe each physical parameter as a function of the load balancing primary parameters. We start with the static frictional constant. As previously stated, $\mu_s(l_{j,i},v_j)$ is proportional to the dependency of the task to other tasks or resources in a node. Hence, we have

$$\mu_s(l_{j,i},v_j) \propto \sum_{k}^{l_{j,k}\neq 0} T_{i,k}$$

$$\mu_s(l_{j,i},v_j) \propto R_{i,j}$$

The tan β parameter, on the other hand, determines whether or not a load should be send to a specific neighboring node; hence it is related to the total quantity of the load of both processors of source and destination. tan β is also inversely proportional to the cost of the link which is represented by $e_{i,j}$ in our model. While considering tan β, the algorithm should take into account the changes in total load quantities of the source and destination nodes after moving the current load and should make decisions accordingly.

Table 1. Physical parameters and load balancing concepts.

Parameter	Equivalent in Load balancing model
μ_s	The degree of participation of a node in the load balancing and the dependency of the task to other tasks or resources in the node.
μ_k	The communication cost of sending a task over a network link due to the bandwidth as well as the dependency of the task to other tasks or resources in its source node or in its neighboring node.
m	The load quantity which is a representation of the computational complexity or the mnemonic size of the load.
tanβ	The difference between the number of loads of two neighboring nodes i and j with respect to $e_{i,j}$, i.e. the gradient.
h	The total load quantity of a node.
E_h	The traffic caused by the transfer of loads on a link.
$e_{i,j}$	The distance between two links, the communication delay and/or the probability of occurrence of fault in a time unit.

But as this depends on the size of the load to be transferred, it cannot be considered here. It would rather be taken into account as a safety bound for μ_s while dealing with a specific load. Hence, tan $\beta(v_i,v_j,e_{i,j})$ can be derived as

$$\tan\beta(v_i,v_j,e_{i,j}) \propto \frac{h(v_i)-h(v_j)}{e_{i,j}}$$

It is while in reality after considering the effect of load transfer on *h,* the appropriate relation for $\tan\beta(v_i,v_j,e_{i,j})$ would be

$$\tan\beta(v_i,v_j,e_{i,j},l_{i,k}) \propto \frac{h(v_i)-h(v_j)-2l_{i,k}}{e_{i,j}}$$

where $h(v_i)$ can be calculated with a summation over the quantity of all the loads of v_i as

$$h(v_i)=\sum_{all\ k} l_{i,k}$$

and is a measure of the total amount of load on a node *i*.

The weight of a link $e_{i,j}$ contributes to the cost of sending a task from a node *i* to a neighboring destination node *j*. Hence, $e_{i,j}$ depends on the length of the link between *i* and *j* while it is inversely proportional to the bandwidth of the link. The factors indicating a degree of safety of the link can also enter this function, as a higher value of $e_{i,j}$ resulting in a less steep slope. Thus, $e_{i,j}$ can be derived as

$$e_{i,j} \propto d_{i,j}$$

$$e_{i,j} \propto \frac{1}{bw_{i,j}}$$

$$e_{i,j} \propto \frac{1}{(1-f_{i,j})^{c\frac{d_{i,j}}{bw_{i,j}}}}$$

where $f_{i,j}$ in the third statement is the probability of occurrence of fault in a time unit. Hence, $(1-f_{i,j})^{c\frac{d_{i,j}}{bw_{i,j}}}$ is a measure of the probability that the load does not encounter any faults during its transmission. The constant μ_k is a representative of the dependency of the task to the resources in a node or to the other tasks on the same processor; hence, it hinders the load from being transferred to the far away nodes. Hence, μ_k and μ_s generally describe the same concept. While the former tries to resist the transport of a dependant task over the network, the latter demands that if the dependant task is being transferred it should stay within some proximity of the source node. Therefore, we can conclude $\mu_k \propto \mu_s$ which is interestingly also true in the physical world.

5. The Load Balancing Algorithm

A set of different load balancing algorithms have been proposed in the literature. Although the proposed algorithms are sometimes very efficient, usually what they model is an oversimplified model of the systems we are dealing with in the real world. Hence, most of them either completely neglected some of the critical and yet complicated issues of the problem over-optimistically, or their algorithms lacks the necessary details and justifications for describing a more generic system in a way that it seems necessary to redesign the algorithm from the scratch for each and any new system.

In order to address the problems of the past and also to bypass the intricacies of modeling a real world system, we try to model the load balancing problem with an inspiration from the real world of physics where its facts are considered as the tangible evidences as we do practice it in our everyday life.

The algorithm we propose here, in the context of dynamic load balancing, can be considered as a variant of gradient model algorithm which is modeled in a new way with the physical parameters. The presence of the underlying physical parameters also gives the algorithm an evolutionary nature where the overall system tries to converge to a minimal energy and yet a more stable state. Some other examples of such algorithms are the evolutionary algorithms, such as simulated annealing, which also have been studied extensively in the past.

5.1. The Main Algorithm

In the previous section, most of the physical parameters in our model have been explained mathematically in terms of load balancing parameters. Once we derived all of the physical parameters, the load balancing algorithm and related equations can be immediately drawn by solving the equilibrium equations in the physical scenario.

Yet, the point that is missed is how to balance the different physical forces applied to the object as a result of the existence of multiple slopes in its neighborhood. The direction of the forces applied to the object varies depending on the fact that how the edges of the graph are mapped to *xy* plane. To solve this problem we presume that any time an object wants to start its movement, we do not apply the net force of all the different force vectors to it but after calculating the parameters (angle) of each slope independently, the object chooses the choicest slope stochastically using an arbiter function. This stochastic nature can also be considered for some other parameters which are not too much rigid like μ_s and μ_k.

As we want the system to converge to a nearly optimal state, as the time passes, the value of system parameters will get more rigid. Hence, it seems quite logical to decrease the stochastic nature of the parameters when time passes.

The condition for initiating the motion of an object in a direction is given as $\tan\beta > \mu_s$ from section 2. By replacing the tan β and μ_s parameters, as derived in the previous section, the equivalent condition for transferring the load $l_{i,k}$ from node *i* to node *j* is obtained as

$$\frac{h(v_i)-h(v_j)}{e_{i,j}} > R_{k,i}\sum_{x}^{l_{i,x}\neq 0} T_{k,x}$$

To consider the size of the task in the feasibility decision, we should decrease the value of tan β by $2l_{i,j}/e_{i,j}$ which is the difference of the load quantities of the source and destination nodes before and after transferring $l_{i,j}$. Hence, the final relation can be expressed as

$$\frac{h(v_i)-h(v_j)-2l_{i,k}}{e_{i,j}} > R_{k,i}\sum_{x}^{l_{i,x}\neq 0} T_{k,x}$$

After replacing the values of $e_{i,j}$ and h we have

$$\sum_{all\ x} l_{i,x} - \sum_{all\ x} l_{j,x} - 2l_{i,k} > \frac{d_{i,j}R_{k,i}\sum_{x}^{l_{i,x}\neq 0} T_{k,x}}{bw_{i,j}(1-f_{i,j})^{c\frac{d_{i,j}}{bw_{i,j}}}}$$

As *i* might have more than one neighbors and we have to decide the direction of load, we may have many of these equations for $l_{i,k}$. We will use tan β parameter as an input to the stochastic function which chooses the proper link.

In order to guarantee the validity of our load balancing model, we should ensure that nothing is missed while transferring from the physical model to our load balancing model. The exertion of forces to the object has been considered in the last paragraph by checking the necessary conditions that will cause the object to move. But we have not yet modeled the wasting of energy which is a result of the application of the dynamic friction to the object.

After the proper link (if any) for sending the load has been chosen, the object will start its movement down the chosen slope. While this movement is taking place, the object loses some of its total energy in the form of heat in the environment. By the time the load is received at the destination node, several properties of the physical model such as the energy of object, the total quantity of load of the nodes, etc. have been changed. Hence, we need to manually apply these changes to the parameters in our load balancing model accordingly. In order to monitor the changes in the energy state of an object, we store the potential height which is a measure of the total energy of the object in a flag in the load; this flag is initialized at the start of the game with the height of the initial position of the object, h_0. Let us consider that a load $l_{i,k}$ on the node v_i is being transferred to a node v_j. The initial energy of the load can be calculated as

$$mgh_0 = g.l_{i,k} \sum_{all\ x} l_{i,x}$$

Before the load is transferred to another node and received on the other side of the link, the algorithm updates the flag to reflect the energy which will be wasted due to the kinetic frictional force. This energy can then be calculated as

$$E_h = g\mu_k e_{i,j} l_{i,k}$$

And after replacing the physical parameters we get

$$E_h = c_0 \frac{d_{i,j} l_{i,k} R_{k,i} \sum_{x}^{l_{i,x}\neq 0} T_{k,x}}{bw_{i,j}(1-f_{i,j})^{c_1 \frac{d_{i,j}}{bw_{i,j}}}}$$

where c_0 and c_1 in the last equation are some constants which should be configured according to the properties of the system being modeled. These properties may encompass the system criticality and softness, the desired QOS, etc.

The other parameters that should be updated immediately, after the destination of the load is determined, are the quantity of the loads of the source and the destination nodes, i.e. nodes i and j.

As the load reaches node j, it has lost some of its energy as traffic in the link between i and j. This time the object has a velocity which let it climb a hill unless the hill is too much high. Again as for the case of the stationary load, we have to decide to which neighboring node to send the load. This time as a result of the inertia of the object, the quantity of the loads on the source and destination nodes and the link's weight are not the only effective factors. In order to assess the feasibility of taking a path to a neighboring node, we use the energy model. Based on the energy model, as discussed in section 2, we can determine the height of threshold for a neighboring node as the potential height; the object will not be able to climb any hill with a bigger height than this threshold. The equation for calculating the potential height at a time instant t was derived in section 2 as

$$h^*_t = \frac{mgh_0 - \sum_{i=1}^{t} E_{h,i}}{mg} = h_0 - \frac{\sum_{i=1}^{t} E_{h,i}}{mg}$$

From this equation the following equation is easily drawn

$$h^*_t = h^*_{t-1} - \frac{E_{h,t}}{mg}$$

where $E_{h,t}$ is the energy wasted from time $t-1$ to t. Assuming that at each time unit only a single load is transferred over a link, we can conclude the sufficient condition for the feasibility of using a link as

$$h^*_{t-1} - \frac{E_{h,t}}{l_{i,k}\,g} > h(v_j)$$

and after some replacements as

$$h^*_{t-1} - c_0 \frac{d_{i,j} R_{k,i} \sum_{x}^{l_{i,x}\neq 0} T_{k,x}}{g\,bw_{i,j}(1-f_{i,j})^{c_1 \frac{d_{i,j}}{bw_{i,j}}}} > h(v_j)$$

Assuming that the load $l_{i,k}$ on a processor i is being sent to another processor j. Here, h^*_{t-1} denotes the value which is previously stored in a flag of the load.

One interesting thing to consider is that after some simple replacements the last inequality could be written as

$$h^*_{t-1} - \mu_s e_{i,j} > h(v_j)$$

which is another representation of the equation derived in theorem 1, only if the contour is selected to include the nodes located $e_{i,j}$ links away, i.e. $r_{c,p} = e_{i,j}$.

5.2. Improvements and Heuristics

In the previous section, we probed into the general structures and equations under which the load balancing algorithm is performed. In this section, we present some improvements on the core of the algorithm in order to make it more robust. In particular, we will propose an approach for obtaining stochastic arbitrator function of a parameter from a set of available deterministic results of that parameter. This stochastic function, rather than constricting itself to a rigid value, in a fuzzy manner define correctness function for each parameter and assign probabilities to each possible value according to the defined measure of correctness. The rigidity of the correct values increases over time in an attempt to make the system converge to an optimal solution. Hence, the algorithm follows an evolutionary approach for dynamically configuring the values of parameters.

The first case of such a stochastic function is used for choosing the best neighbor while the load is stationary and it is to be transferred from the node to which it was assigned (generated). The deterministic value of the angle's tangent for each link is obtained from

$$a_{i,j} = \frac{h(v_i) - h(v_j)}{e_{i,j}}$$

where $0 < j \leq m$, and m is the number of neighboring nodes of i satisfying the feasibility criteria.

Having calculated this value for each link of the node i, these values are fed to an arbitrator function which stochastically chooses the proper link. Without loss of generality, we assume that $a_{i,j}$ values for all j's are already sorted in the descending order of magnitude, i.e. $a_{i,1}$ has the biggest value. The stochastic arbiter function gives the most of the chance to the links which are the steepest considers some rare probabilities for choosing the less steep slopes. This function calculates the probabilities based on a probabilistic model of free trials. Note that the probability of success for each trial is not fixed. Let $p_{i,j}(t)$ be the probability of choosing the node j as the destination for the load $l_{i,k}$ at a time t from the beginning of trial. This function can be derived as

$$\beta_{i,k}(t) = \left(1 - \sum_{1}^{k-1} \beta_{i,x}(t)\right)\left(1 - \frac{|a_{i,0} - a_{i,k}|}{|a_{i,0} - a_{i,m}|}\right), \quad 1 < k \le m$$

with the initial value of

$$\beta_{i,1}(t) = 1 - \beta_0 e^{-\frac{c|a_{i,1} - a_{i,m}|}{t_{\max} - t}}$$

where $0 < \beta_0 < 1$ is the initial probability of choosing a link other than the steepest one, and t_{max} and c are the constants which control the convergence of the stochastic function to the rigid maximum value, i.e. $a_{i,1}$ as the time passes.

In the same way, once the load has been received in the destination, the algorithm should decide to which neighboring node to transfer the load if any such a node that satisfies the feasibility criteria ever exists.

The $a_{i,j}$ values are calculated this time from the equation

$$a_{i,j}(t) = h^{*}_{t-1} - \frac{E_{h,t}}{l_{i,k}\,g} - h(v_j)$$

where $a_{i,j}$'s are fed to the same arbiter function as the one proposed in the previous section to produce the stochastic results. The output of stochastic arbiter function then specifies the link which should be chosen as the next destination of the load $l_{i,k}$.

6. Conclusions

In this paper, by envisioning the particle and place scenario in dynamic physics we proposed a framework for modeling load balancing in multiprocessors. We showed that this model converges to the nearly optimal solution though the rate of this convergence depends on the choice of the parameters.

Most of the previously proposed load balancing algorithms have ignored some important issues like task dependencies, resource constraints, and fault tolerance, and hence they fail to adapt to the intricate systems in real world. On the other hand, the complexities present in the real systems make the problem of presenting a general load balancing model for such systems very difficult if not impossible. The goal of this work is to propose a scheme for modeling dynamic load balancing in multiprocessors using a tangible physical model in a way that each new system can be easily modeled by identifying the effect and strictness of each of the considered factors in the system understudy and fine-tuning the configuration parameters which describe systems characteristics.

This study is interesting in the sense that while the proposed model considers most of the important factors of load balancing, it avoids the intricacies of modeling the real-world systems by taking inspiration from the world of physics which is much easier to study and sense.

References

[1] J. E. Boillat, "Load Balancing and Poisson Equation in a Graph", *Concurrency: Practice and Experience*, vol. 2, no. 4, pp. 289-313, Dec. 1990.

[2] R.M. Bryant and R.A. Finkel, "A Stable Distributed Scheduling Algorithm", *Proc. Second Int'l Conf. Distributed Computing Systems*, pp. 314-323, 1981.

[3] T. Bultan and C. Aykanat, "A New Mapping Heuristic Based on Mean Field Annealing", *Journal of Parallel and Distributed Computing,* vol. 16, pp. 292-305, 1992.

[4] T.L. Casavant and J.G. Kuhl, "Effects of Response and Stability on Scheduling in Distributed Computing Systems", *IEEE Trans. Software Eng.*, vol. 14, no. 11, pp. 1,578-1,588, Nov. 1988.

[5] T.L. Casavant and J.G. Kuhl, "A Taxonomy of Scheduling in General-Purpose Distributed computing Systems", *IEEE TOSE,* vol. 14, no. 2 pp. 141-154, Feb. 1988.

[6] G. Cybenko, "Dynamic Load Balancing for Distributed Memory Multiprocessors", *Journal of Parallel and Distributed Computing*, vol. 7, no. 2, pp. 279-301, Oct. 1989.

[7] D.L. Eager, E.D. Lazowska and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing", *Performance Evaluation*, vol. 6, pp. 53-68, 1986.

[8] H. Heiss and M. Schmitz, "Decentralized Dynamic Load Balancing: The Particles Approach", *Information Sciences*, vol. 84, no. 1-2, pages 115--128, May 1995.

[9] D. Henrich, "The Liquid Model Load Balancing Method", *Journal of Parallel Algorithms and Applications* (*Special Issue on Algorithms for Enhanced Mesh*), vol. 8, 285-307, 1996.

[10] S.H. Hosseini, B. Litow, M. Malkawa, J. McPherson, and K.Vairavan, "Dynamic Load Balancing for Distributed Memory Multiprocessors", *Journal of Parallel and Distributed Computing*, vol. 7, no. 2, pp. 279-301, Oct. 1989.

[11] C. Hui and S. T. Chanson, "Hydrodynamic Load Balancing", *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No. 11, November 1999.

[12] C.H. Lin and R.M. Keller, "The Gradient Model Load Balancing Method", *IEEE Trans. Software Eng.,* vol. 13, no.1, pp. 32-38, Jan. 1987.

[13] H. Mühlenbein, M. Gorges-Schleuter and O. Krämer, "New solutions to the mapping problem of parallel systems: The evolution approach", *Parallel Computing*, vol. 4, pp. 269-279, 1987.

[14] P. Sadayappan, F. Ercal and J. Ramanujam, "Cluster partitioning approaches to mapping parallel programs onto a hypercube", *Parallel Computing*, Vol. 13, pp.1-16, 1990.

[15] W. Shu and L.V. Kale, "A Dynamic Scheduling Strategy for the Chare-Kernel System", *Proc. Supercomputing '89*, pp. 389-398, Nov. 1989.

[16] J.A. Stankovic, "Stability and Distributed Scheduling Algorithms", *IEEE Trans. Software Eng.*, vol. 11, no. 10, pp.1141-1152, Oct. 1985.

[17] M.M. Theimer and K.A. Lantz, "Finding Idle Machines in a Workstation-Based Distributed Systems", *IEEE TOSE,* vol. 15, no.11, pp. 1444-1457, Nov. 1989.

[18] R. Williams, "Performance of dynamic load balancing algorithms for unstructured mesh calculations", *Concurrency: Practice and Experience*, vol. 3, 457-481, 1991.

[19] C.Z. Xu and F.C.M. Lau, "Optimal Parameters for Load Balancing with the Diffusion Method in Mesh Networks", *Parallel Processing Letters*, vol. 4, no. 1-2, pp. 139-147, June 1994.