A Time Delay Model for Load Balancing with Processor Resource Constraints

Zhong Tang[1], J. Douglas Birdwell[1], John Chiasson[1], Chaouki T. Abdallah[2] and Majeed M. Hayat[2]

Abstract— **A deterministic dynamic nonlinear time-delay systems is developed to model load balancing in a cluster of computer nodes used for parallel computations. This model refines a model previously proposed by the authors to account for the fact that the load balancing operation involves processor time which cannot be used to process tasks. Consequently, there is a trade-off between using processor time/network bandwidth and the advantage of distributing the load evenly between the nodes to reduce overall processing time.**

The new model is shown to be self consistent in that the queue lengths cannot go negative and the total number of tasks in all the queues are conserved (i.e., load balancing can neither create nor lose tasks). It is shown that the proposed model is (Lyapunov) stable for any input, but not necessarily asymptotically stable. Experimental results are presented and compared with the predicted results from the analytical model. In particular, simulations of the models are compared with an experimental implementation of the load balancing algorithm on a parallel computer network.

Keywords— **Load balancing, Networks, Time Delays**

I. Introduction

Parallel computer architectures utilize a set of computational elements (CE) to achieve performance that is not attainable on a single processor, or CE, computer. A common architecture is the cluster of otherwise independent computers communicating through a shared network. To make use of parallel computing resources, problems must be broken down into smaller units that can be solved individually by each CE while exchanging information with CEs solving other problems. For example, the Federal Bureau of Investigation (FBI) National DNA Index System (NDIS) and Combined DNA Index System (CODIS) software are candidates for parallelization. New methods developed by Wang et al. [1][2][3][4] lead naturally to a parallel decomposition of the DNA database search problem while providing orders of magnitude improvements in performance over the current release of the CODIS software.

Effective utilization of a parallel computer architecture requires the computational load to be distributed more or less evenly over the available CEs. The qualifier "more or less" is used because the communications required to distribute the load consume both computational resources and network bandwidth. A point of diminishing returns exists.

Distribution of computational load across available resources is referred to as the *load balancing* problem in the literature. Various taxonomies of load balancing algorithms exist. Direct methods examine the global distribution of computational load and assign portions of the workload to resources before processing begins. Iterative methods examine the progress of the computation and the expected utilization of resources, and adjust the workload assignments periodically as computation progresses. Assignment may be either deterministic, as with the dimension exchange/diffusion [5] and gradient methods, stochastic, or optimization based. A comparison of several deterministic methods is provided by Willeback-LeMain and Reeves [6]. Approaches to modeling and static load balancing are given in [7][8][9].

To adequately model load balancing problems, several features of the parallel computation environment should be captured: (1) The workload awaiting processing at each CE; (2) the relative performances of the CEs; (3) the computational requirements of each workload component; (4) the delays and bandwidth constraints of CEs and network components involved in the exchange of workloads and, (5) the delays imposed by CEs and the network on the exchange of measurements. A queuing theory [10] approach is well-suited to the modeling requirements and has been used in the literature by Spies [11] and others. However, whereas Spies assumes a homogeneous network of CEs and models the queues in detail, the present work generalizes queue length to an expected waiting time, normalizing to account for differences among CEs, and aggregates the behavior of each queue using a continuous state model.

The present work focuses upon the effects of delays in the exchange of information among CEs, and the constraints these effects impose on the design of a load balancing strategy. Preliminary results by the authors appear in [12][13][14][15][16][17][18]. An issue that was not considered in this previous work is the fact that the load balancing operation involves processor time which is not being used to process tasks. Consequently, there is a trade-off between using processor time/network bandwidth and the advantage of distributing the load evenly between the nodes to reduce overall processing time. The fact that the simulations of the model in [18] showed the load balancing to be carried out faster than the corresponding experimental results motivated the authors to refine the model to account for processor constraints. A new deterministic dynamic time-delay system model is developed here to capture these constraints. The model is shown to be self consistent in that the queue lengths cannot go negative and the total number of tasks in all the queues and the network are conserved (i.e., load balancing can neither create nor lose tasks). In contrast to the results in [18] where it was analytically shown the systems was always asymptoti-

[1] ECE Dept, The University of Tennessee, Knoxville, TN 37996-2100, ztang@utk.edu, birdwell@utk.edu, chiasson@utk.edu

[2] ECE Dept, University of New Mexico, Alburquerque, NM 87131-1356, chaouki@eece.unm.edu, hayat@eece.unm.edu

cally stable, the new model is only (Lyapunov) stable and asymptotic stability must be insured by judicious choice of the feedback. Simulations of the nonlinear model are compared with data from an *experimental implementation* of the load balancing algorithm on a parallel computer network.

Section II presents our approach to modeling the computer network and load balancing algorithms to incorporate the presence of delay in communicating between nodes and transferring tasks. Section III shows that the proposed model correctly predicts that the queue lengths cannot go negative and that the total number of tasks in all the queues are conserved by the load balancing algorithm. This section ends with a proof of (Lyapunov) stability of the model. Section IV presents how the model parameters were obtained and our experimental setup. Section V presents a comparison of simulations of the nonlinear model with the actual experimental data. Finally, Section VI is a summary and conclusion of the present work and a discussion of future work.

II. Mathematical Model of Load Balancing

In this section, a nonlinear continuous time models is developed to model load balancing among a network of computers. To introduce the basic approach to load balancing, consider a computing network consisting of n computers (nodes) all of which can communicate with each other. At start up, the computers are assigned an equal number of tasks. However, when a node executes a particular task it can in turn generate more tasks so that very quickly the loads on various nodes become unequal. To balance the loads, each computer in the network sends its queue size $q_j(t)$ to all other computers in the network. A node i receives this information from node j *delayed* by a finite amount of time τ_{ij}; that is, it receives $q_j(t-\tau_{ij})$. Each node i then uses this information to compute its local estimate[1] of the average number of tasks in the queues of the n computers in the network. In this work, the simple estimator $\left(\sum_{j=1}^n q_j(t-\tau_{ij})\right)/n$ ($\tau_{ii}=0$) which is based on the most recent observations is used. Node i then compares its queue size $q_i(t)$ with its estimate of the network average as $\left(q_i(t) - \left(\sum_{j=1}^n q_j(t-\tau_{ij})\right)/n\right)$ and, if this is greater than zero, the node sends some of its tasks to the other nodes. If it is less than zero, no tasks are sent (see Figure 1). Further, the tasks sent by node i are received by node j with a delay h_{ij}. The controller (load balancing algorithm) decides how often and fast to do load balancing (transfer tasks among the nodes) and how many tasks are to be sent to each node.

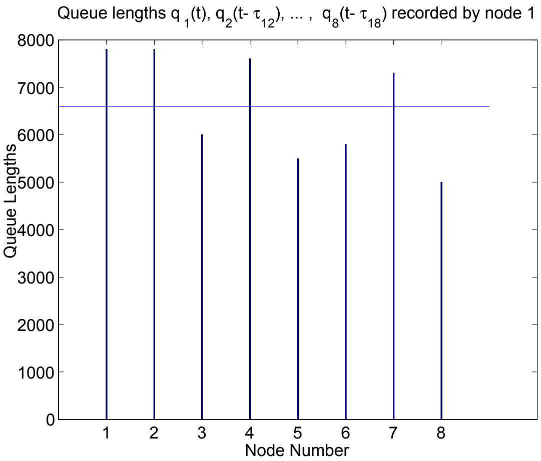

Fig. 1. Graphical description of load balancing. This bar graph shows the load for each computer vs. node of the network. The thin horizontal line is the average load as estimated by node 1. Node 1 will transfer (part of) its load only if it is above its estimate of the average. Also, it will only transfer to nodes that it estimates are below the node average.

As just explained, each node controller (load balancing algorithm) has only *delayed* values of the queue lengths of the other nodes, and each transfer of data from one node to another is received only after a finite time delay. An important issue considered here is the effect of these delays on system performance. Specifically, the model developed here represents our effort to capture the effect of the delays in load balancing techniques as well as the processor constraints so that system theoretic methods could be used to analyze them.

A. Basic Model

The basic mathematical model of a given computing node for load balancing is given by

$$\begin{aligned}\frac{dx_i(t)}{dt} &= \lambda_i - \mu_i\left(1-\eta_i(t)\right) - U_m(x_i)\eta_i(t)\\ &\quad + \sum_{j=1}^n p_{ij}\frac{t_{p_i}}{t_{p_j}} U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij}) \qquad (1)\end{aligned}$$

$$p_{ij} \geqslant 0, p_{jj}=0, \sum_{i=1}^n p_{ij} = 1$$

where

$$\begin{aligned}U_m(x_i) &= U_{m0} > 0 \text{ if } x_i > 0\\ &= 0 \qquad\qquad \text{if } x_i \le 0\end{aligned}$$

In this model we have

- n is the number of nodes.
- $x_i(t)$ is the *expected waiting time* experienced by a task inserted into the queue of the i^{th} node. With $q_i(t)$ the number of *tasks* in the i^{th} node and t_{p_i} the average time needed to process a task on the i^{th} node, the expected (average) waiting time is then given by $x_i(t) = q_i(t)t_{p_i}$. Note that $x_j/t_{p_j} = q_j$ is the number of tasks in the node j queue. If these tasks were transferred to node i, then the waiting time transferred is $q_j t_{p_i} = x_j t_{p_i}/t_{p_j}$, so that the fraction t_{p_i}/t_{p_j} converts waiting time on node j to waiting time on node i.
- $\lambda_i \geq 0$ is the rate of generation of waiting time on the i^{th} node caused by the addition of tasks (rate of increase in x_i)

[1] It is an estimate because at any time, each node only has the delayed value of the number of tasks in the other nodes.

- $\mu_i \geq 0$ is the rate of reduction in waiting time caused by the service of tasks at the i^{th} node and is given by $\mu_i \equiv (1 \times t_{p_i})/t_{p_i} = 1$ for all i if $x_i(t) > 0$, while if $x_i(t) = 0$ then $\mu_i \triangleq 0$, that is, if there are no tasks in the queue, then the queue cannot possibly decrease.
- $\eta_i = 1$ or 0 is the *control input* which specifies whether tasks (waiting time) are processed on a node or tasks (waiting time) are transferred to other nodes.
- U_{m0} is the limit on the rate at which data can be transmitted from one node to another and is basically a bandwidth constraint.
- $p_{ij}U_m(x_j)\eta_j(t)$ is the rate at which node j sends waiting time (tasks) to node i at time t where $p_{ij} \geqslant 0, \sum_{i=1}^{n} p_{ij} = 1$ and $p_{jj} = 0$. That is, the transfer from node j of expected waiting time (tasks) $\int_{t_1}^{t_2} U_m(x_j)\eta_j(t)dt$ in the interval of time $[t_1, t_2]$ to the other nodes is carried out with the i^{th} node being sent the fraction $p_{ij}\int_{t_1}^{t_2} U_m(x_j)\eta_j(t)dt$ of this waiting time. As $\sum_{i=1}^{n}\left(p_{ij}\int_{t_1}^{t_2} U_m(x_j)\eta_j(t)dt\right) = \int_{t_1}^{t_2} U_m(x_j)\eta_j(t)dt$, this results in removing *all* of the waiting time $\int_{t_1}^{t_2} U_m(x_j)\eta_j(t)dt$ from node j.
- The quantity $-p_{ij}U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij})$ is the rate of transfer of the expected waiting time (tasks) at time t from node j by (to) node i where h_{ij} ($h_{ii} = 0$) is the time delay for the task transfer from node j to node i.
- The factor t_{p_i}/t_{p_j} converts the waiting time from node j to waiting time on node i

In this model, all rates are in units of the *rate of change of expected waiting time*, or *time/time* which is dimensionless. As $\eta_i = 1$ or 0, node i can only send tasks to other nodes and cannot initiate transfers from another node to itself. A delay is experienced by transmitted tasks before they are received at the other node. Model (1) is the basic model, but one important detail remains unspecified, namely the exact form p_{ji} for each sending node i. One approach is to choose them as constant and equal

$$p_{ji} = 1/(n-1) \text{ for } j \neq i \text{ and } p_{ii} = 0 \tag{2}$$

where it is clear that $p_{ji} \geqslant 0, \sum_{j=1}^{n} p_{ji} = 1$. Another approach is given in section IV below.

A.1 Control Law

The information available to use in a feedback law at each node i is the value of $x_i(t)$ and the *delayed* values $x_j(t-\tau_{ij})$ ($j \neq i$) from the other nodes. Let τ_{ij} ($\tau_{ii} = 0$) denote the time delay for communicating the expected waiting time x_j from node j to node i. These communication are much smaller than the corresponding data transfer delays h_{ij}. Define

$$x_{i\_avg} \triangleq \left(\sum_{j=1}^{n} x_j(t-\tau_{ij})\right)/n$$

to be the *local average* which is the i^{th} node's estimate of the average of all the nodes. (This is an only an estimate due to the delays). Further, define

$$y_i(t) \triangleq x_i(t) - \frac{\sum_{j=1}^{n} x_j(t-\tau_{ij})}{n}$$

to be the expected waiting time relative to the local average estimate on the i^{th} node.

A simple control law one might consider is

$$\eta_i(t) = h\left(y_i(t)\right) \tag{3}$$

where

$$y_i(t) = x_i(t) - \frac{\sum_{j=1}^{n} x_j(t-\tau_{ij})}{n}$$

and $h(\cdot)$ is a hysteresis function given by

$$\begin{aligned} h(y) &= 1 \text{ if } y \geqslant y_0 \\ &= 1 \text{ if } 0 < y < y_0 \text{ and } \dot{y} > 0 \\ &= 0 \text{ if } 0 < y < y_0 \text{ and } \dot{y} < 0 \\ &= 0 \text{ if } y < 0. \end{aligned}$$

The control law basically states that if the i^{th} node output $x_i(t)$ is above the local average $\left(\sum_{j=1}^{n} x_j(t-\tau_{ij})\right)/n$ by the threshold amount y_0, then it sends data to the other nodes, while if it is less than the local average nothing is sent. The hysteresis loop is put in to prevent chattering. (In the time interval $[t_1, t_2]$, the j^{th} node receives the fraction $\int_{t_1}^{t_2} p_{ji}\left(t_{p_i}/t_{p_j}\right)U_m(x_i)\eta_i(t)dt$ of transferred waiting time $\int_{t_1}^{t_2} U_m(x_i)\eta_i(t)dt$ delayed by the time h_{ij}.)

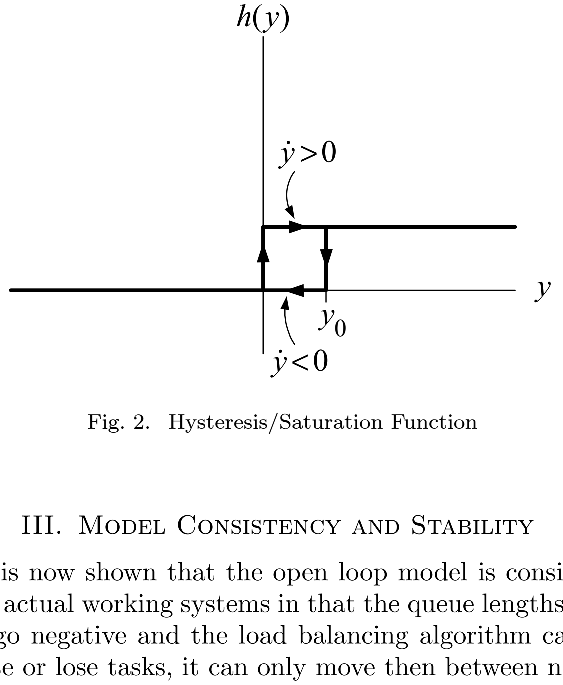

Fig. 2. Hysteresis/Saturation Function

III. Model Consistency and Stability

It is now shown that the open loop model is consistent with actual working systems in that the queue lengths cannot go negative and the load balancing algorithm cannot create or lose tasks, it can only move then between nodes.

A. Non Negativity of the Queue Lengths

To show the non negativity of the queue lengths, recall that the queue length of each node is given by $q_i(t) =$

$x_i(t)/t_{p_i}$. The model is rewritten in terms of these quantities as

$$\frac{d}{dt}\Big(x_i(t)/t_{p_i}\Big) = \frac{\lambda_i - \mu_i\,(1-\eta_i(t))}{t_{p_i}} - \frac{1}{t_{p_i}}U_m(x_i)\eta_i(t) + \sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij}). \quad (4)$$

Given that $x_i(0) > 0$ for all i, it follows from the right-hand side of (4) that $q_i(t) = x_i(t)/t_{p_i} \geq 0$ for all $t \geq 0$ and all i. To see this, suppose without loss of generality that $q_i(t) = x_i(t)/t_{p_i}$ is the first queue to go to zero, and let t_1 be the time when $x_i(t_1) = 0$. At the time t_1, $\lambda_i - \mu_i\,(1-\eta_i(t)) = \lambda_i \geq 0$ as $\mu_i(x_i) = 0$ if $x_i = 0$. Also, $\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij}) \geq 0$ as $\eta_j \geq 0$. Further, the term $U_m(x_i) = 0$ for $x_i \leq 0$. Consequently

$$\frac{d}{dt}\Big(x_i(t)/t_{p_i}\Big) \geq 0 \text{ for } x_i = 0$$

and thus the queues cannot go negative.

B. Conservation of Queue Lengths

It is now shown that the total number of tasks in all the queues and the network are conserved. To do so, sum up equations (4) from $i = 1, ..., n$ to get

$$\frac{d}{dt}\left(\sum_{i=1}^{n} q_i(t)\right) = \sum_{i=1}^{n}\left(\frac{\lambda_i - \mu_i(x_i)\,(1-\eta_i)}{t_{p_i}}\right) \quad (5)$$
$$-\sum_{i=1}^{n}\frac{U_m(x_i(t))}{t_{p_i}}\eta_i + \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij})$$

which is the rate of change of the total queue lengths on all the nodes. However, the network itself also contains tasks. The dynamic model of the queue lengths in the network is given by

$$\frac{d}{dt}q_{net_i}(t) = -\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij}) + \sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}U_m(x_j(t))\eta_j(t). \quad (6)$$

Here q_{net_i} is the number of tasks put on the network that are being sent to node i. This equation simply says that the j^{th} node is putting tasks on the network to be sent to node i at the rate $\frac{p_{ij}}{t_{p_j}}U_m(x_j(t))\eta_j(t)$ while the i^{th} node is taking these tasks from node j off the network at the rate $-\frac{p_{ij}}{t_{p_j}}U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij})$. Summing (6) over all the nodes, one obtains

$$\frac{d}{dt}\left(\sum_{i=1}^{n} q_{net_i}(t)\right) = -\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij})$$
$$+\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}U_m(x_j(t))\eta_j(t)$$
$$= -\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij})$$
$$+\sum_{j=1}^{n}\frac{U_m(x_j(t))\eta_j(t)}{t_{p_j}}. \quad (7)$$

Adding (5) and (7), one obtains the conservation of queue lengths given by

$$\frac{d}{dt}\sum_{i=1}^{n}\Big(q_i(t) + q_{net_i}(t)\Big) = \sum_{i=1}^{n}\left(\frac{\lambda_i - \mu_i\,(1-\eta_i)}{t_{p_i}}\right). \quad (8)$$

In words, the total number of tasks which are in the system (i.e., in the nodes and/or in the network) can increase only by the rate of arrival of tasks $\sum_{i=1}^{n}\lambda_i/t_{p_i}$ at all the nodes, or similarly, decrease by the rate of processing of tasks $\sum_{i=1}^{n}\mu_i\,(1-\eta_i)\,/t_{p_i}$ at all the nodes. The load balancing itself cannot increase or decrease the total number of tasks in all the queues.

C. Stability of the Model

Combining the results of the previous two subsections, one can show Lyapunov stability of the model. Specifically, we have

Theorem: Given the system described by (1) and (6) with $\lambda_i = 0$ for $i = 1, ..., n$ and initial conditions $x_i(0) \geq 0$, then the system is Lyapunov stable for any choice of the switching times of the control input functions $\eta_i(t)$.

Proof: First note that the q_{net_i} are non negative as

$$q_{net_i}(t) = \sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}\left(\int_{t-h_{ij}}^{t}U_m(x_j(\tau))\eta_j(\tau)d\tau\right) \geq 0. \quad (9)$$

By the non-negativity property of the q_i, the linear function

$$V\Big(q_i(t), q_{net_i}(t)\Big) \triangleq \sum_{i=1}^{n}\Big(q_i(t) + q_{net_i}(t)\Big)$$

is a positive definite function. Under the conditions of the theorem, equation (8) becomes

$$\frac{d}{dt}\sum_{i=1}^{n}\Big(q_i(t) + q_{net_i}(t)\Big) = -\sum_{i=1}^{n}\frac{\mu_i(q_i/t_{p_i})}{t_{p_i}}\,(1-\eta_i) \quad (10)$$

which is negative semi-definite. By standard Lyapunov theory (e.g., see [19]), the system is stable.

D. Nonlinear Model with Non Constant p_{ij}

The model (1) did not have the p_{ij} specified explicitly. For example, they can be considered constant as specified by (2). However, it could be useful to use the local information of the waiting times $x_i(t), i = 1, .., n$ to set their values. Recall that p_{ij} is the fraction of $\int_{t_1}^{t_2} U_m(x_j)\eta_j(t)dt$ in the interval of time $[t_1, t_2]$ that node j allocates (transfers) to node i and conservation of the tasks requires $p_{ij} \geqslant 0, \sum_{i=1}^{n} p_{ij} = 1$ and $p_{jj} = 0$. The quantity $x_i(t-\tau_{ji}) - x_j^{avg}$ represents what node j estimates the waiting time in the queue of node i is with respect to the local average of node j. If queue of node i is above the local average, then node j does not send tasks to it. Therefore $\text{sat}\big(x_j^{avg} - x_i(t-\tau_{ji})\big)$ is an appropriate measure by node j as to how much node i is *below* the local average. Node j then repeats this computation for all the other nodes and then portions out its tasks among the other nodes according to the amounts they are below the local average, that is,

$$p_{ij} = \frac{\text{sat}\left(x_j^{avg} - x_i(t-\tau_{ji})\right)}{\displaystyle\sum_{i\ \ni\ i\neq j} \text{sat}\left(x_j^{avg} - x_i(t-\tau_{ji})\right)}. \qquad (11)$$

All p_{ij} are defined to be zero, and no load is transferred, if the denominator $\displaystyle\sum_{i\ \ni\ i\neq j} \text{sat}\big(x_j^{avg} - x_i(t-\tau_{ji})\big) = 0$. This is illustrated in Figure 3.

Remark If the denominator

$$\sum_{i\ \ni\ i\neq j} sat\left(x_j^{avg} - x_i(t-\tau_{ji})\right)$$

is zero, then $x_j^{avg} - x_i(t-\tau_{ji}) \leq 0$ for all $i \neq j$. However, by definition of the average,

$$\sum_{i\ \ni\ i\neq j} \left(x_j^{avg} - x_i(t-\tau_{ji})\right) + x_j^{avg} - x_j(t)$$
$$= \sum_{i} \left(x_j^{avg} - x_i(t-\tau_{ji})\right) = 0$$

which implies

$$x_j^{avg} - x_j(t) = -\sum_{i\ \ni\ i\neq j} \left(x_j^{avg} - x_i(t-\tau_{ji})\right) > 0.$$

That is, if the denominator is zero, the node j is not greater than the local average and so it is therefore not sending out any tasks.

IV. Model Parameters and Experimental Setup

A. Model Parameters

In this section, the determination of the model parameters is discussed. Experiments were performed to determine the value of U_{m0} and the threshold y_0. The top plot in Figure 4 is the experimentally determined time to transfer data from one node to another in microseconds as function of message size in bytes. Each host measures the average, minimum and maximum times required to exchange data between itself and every other node in the parallel virtual machine (PVM) environment. The node starts the timer when initiating a transfer and stops the timer when it receives the data back, so the round-trip transfer time is measured. This also avoids the problem of synchronizing clocks on two different machines. The data sizes vary from 4 bytes to 4 Mbytes. In order to ensure that anomalies in message timings are minimized the tests are repeated 20 times for each message size.

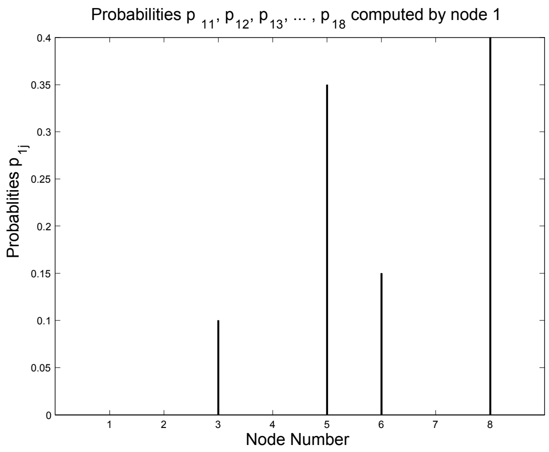

Fig. 3. Illustration of a hypothetical distribution p_{i1} of the load at some time t from node 1's point of view. Node 1 will send data out to node i in proportion p_{i1} it estimates node i is below the average where $\sum_{i=1}^{n} p_{i1} = 1$ and $p_{11} = 0$

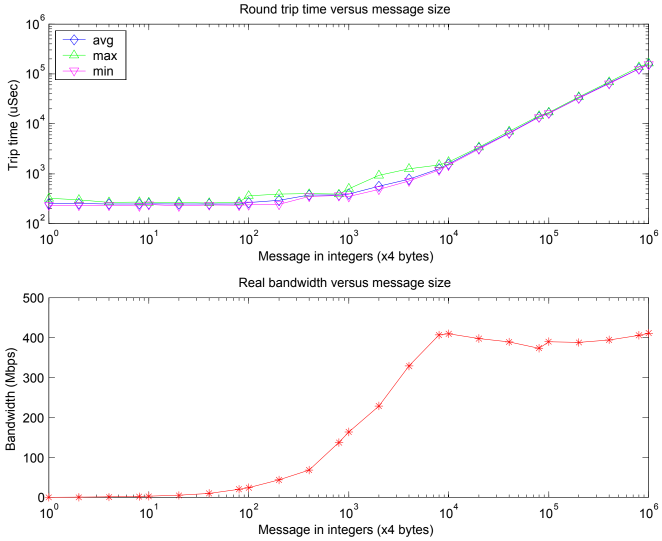

Fig. 4. Top: Round trip time vs. amount of data transfered in bytes. Bottom: Measured bandwidth vs. amount of data transfered in bytes.

The bottom plot in Figure 4 is the experimentally determined bandwidth in Mbps versus the message size in bytes. Based on this data, the threshold for the size of the data transfer could be chosen to be less than 4×10^4 bytes so that this data is transferred at a bandwidth of about 400 Mbps. Messages of larger sizes won't improve the bandwidth. Meanwhile, very high bandwidth means the system must yield computing time for communication time. Here the threshold is chosen to be 4×10^3 bytes since, as the top of Figure 4 shows, this is a trade-off between bandwidth and transfer time. With a typical task to be transferred of size 400 bytes/task (3200 bits/task), this means that the threshold is 10 tasks. Further, the hysteresis threshold y_0 is given by

$$y_0 = 10 \times t_{pi},$$

while the bandwidth constraint U_{m0} is given by

$$\frac{U_{m0}}{t_{pi}} = \frac{400 \times 10^6 \text{ bps}}{3200 \text{ bits/task}} = 12.5 \times 10^4 \text{ tasks/second}$$

$$U_{m0} = 12.5 \times 10^4 \times t_{pi} \, \frac{\text{(waiting-time) seconds}}{\text{second}}$$

B. Experimental Setup of the Parallel Machine

A parallel machine has been built as an experimental facility for evaluation of load balancing strategies. A root node communicates with k groups of computer networks. Each of these groups is composed of n nodes (hosts) holding identical copies of a portion of the database. (Any pair of groups correspond to different databases, which are not necessarily disjoint. A specific record is in general stored in two groups for redundancy to protect against failure of a node.) Within each node, there are either one or two processors. In the experimental facility, the dual processor machines use 1.6 GHz Athlon MP processors, and the single processor machines use 1.33 GHz Athlon processors. All run the Linux operating system. Our interest here is in the load balancing in any one group of n nodes/hosts.

The database is implemented as a set of queues with associated search engine threads, typically assigned one per node of the parallel machine. Due to the structure of the search process, search requests can be formulated for any target profile and associated with any node of the index tree. These search requests are created not only by the database clients; the search process itself also creates search requests as the index tree is descended by any search thread. This creates the opportunity for parallelism; search requests that await processing may be placed in any queue associated with a search engine, and the contents of these queues may be moved arbitrarily among the processing nodes of a group to achieve a balance of the load. This structure is shown in Figure 5. An important point is that the actual delays experienced by the network traffic in the parallel machine are *random*. Experiments were preformed, the resulting time delays measured and these values were used in the simulation for comparison with the experiments.

V. Simulations and Experiments

Here a representative experiment is performed to indicate the effects of time delays in load balancing. The experiment consists of carrying out the load balancing once at a fixed time (open loop) in order to facilitate a comparison with the responses obtained from a simulation based on the model (1)(11). Note that by doing the experiment open loop with a single load balance time, the (random) delays can then be measured after the data is collected and then used in the simulations.

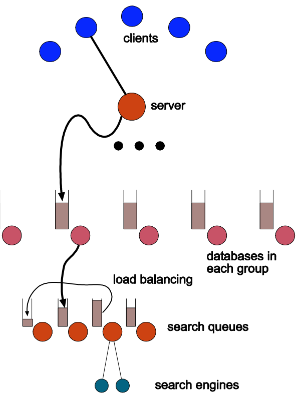

Fig. 5. A depiction of multiple search threads in the database index tree. To even out the search queues, load balancing is done between the nodes (hosts) of a group. If a node has a dual processor, then it can be considered to have two search engines for its queue.

A. Experiment 1

Figure 6 is the experimental response of the queues versus time with an initial queue distribution of $q_1(0) = 600$ tasks, $q_2(0) = 200$ tasks and $q_3(0) = 100$ tasks. The average time to do a search task is 400 μsec. In this experiment, the software was written to execute the load balancing algorithm $t_0 = 1$ millisec using the p_{ij} as specified by (11). The plot shows that the data transfer delay from node 1 to node 2 is $h_{21} = 1.8$ millisec while the data transfer delay from node 1 to node 3 is $h_{31} = 4.0$ millisec. In this experiment the inputs were set as $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 0$.

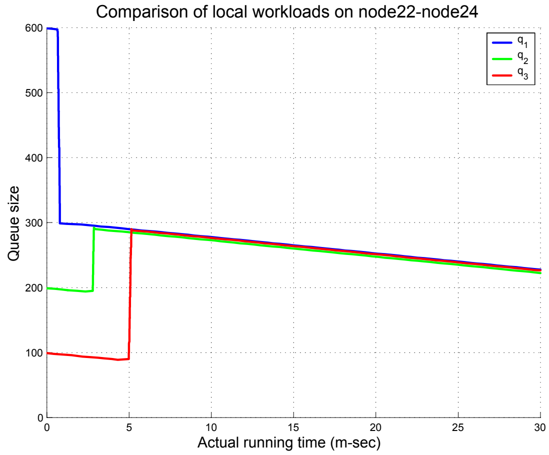

Fig. 6. Experimental results of the load balancing algorithm executed at $t_0 = 1$ millisecond.

Figure 7 is a simulation performed using the model (1) with the p_{ij} as specified by (11). In the model (1) the waiting time was converted to tasks by $q_i(t) = x_i(t)t_{p_i}$ where the t_{p_i}'s were taken to be the average processing time for a search task which is 400 μsec. In Figure 7, $q_1(0) = 600$ tasks $(x_1(0) = 600 \times 400 \times 10^{-6} = 0.24\,\text{sec})$, $q_2(0) = 200$ tasks $(x_2(0) = 200 \times 400 \times 10^{-6} = 0.08\,\text{sec})$ and $q_3(0) = 100$ tasks $(x_3(0) = 100 \times 400 \times 10^{-6} = 0.04\,\text{sec})$

with the delay values set at $h_{21} = 1.8$ millisec, $h_{31} = 4.0$ millisec and the load balancing algorithm was started (open loop) at $t_0 = 1$ millisec. In this simulation, the inputs were set as $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 0, \mu_1 = \mu_2 = \mu_3 = 1$. Note the close similarity of the two figures indicating the model proposed in (1) with the p_{ij} given by (11) is capturing the dynamics of the load balancing algorithm. Figure 8 is a plot of the queue size relative to the local average, i.e.,

$$q_{i\_diff}(t) \triangleq q_i(t) - \left(\sum_{j=1}^{n} q_j(t - \tau_{ij}) \right) /n$$

for each of the nodes. Note the effect of the delay in terms of what each local node *estimates* as the queue average and therefore whether it computes itself to be above or below it. This is now discussed in detail as follows:

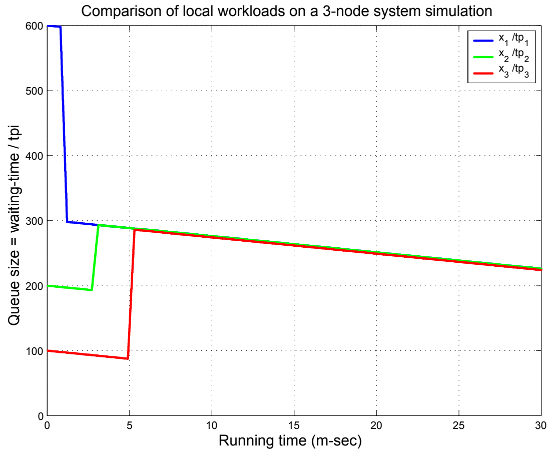

Fig. 7. Simulation of the load balancing algorithm executed at $t_0 = 1$ millisecond.

At the time of load balancing $t_0 = 1$ millisec, node 1 computes its queue size *relative* to its local average (q_{1_diff}) to be 300, node 2 computes its queue size *relative* to its local average (q_{2_diff}) to be -100 and node 3 computes its queue size *relative* to its local average (q_{3_diff}) to be -200.

At time t_1 node 2 receives 100 tasks from node 1 so that node 2's computation of its queue size relative to the local average is now $q_{2\_diff} = 0$. Right after this data transfer, node 2 updates its own queue size to be 300 so its local average is now $(600 + 300 + 100)/3 \approx 333$ making $q_{2\_diff} \approx 300 - 333 = -33$.

At time t_2, node 3 receives the queue size of node 2 (which just increased to about 300 as shown in Figure 6). Node 3 now computes q_{3_diff} to be about $(100 - (600 + 300 + 100)/3) = -233$.

At time t_3, node 1 receives the queue size of node 2 (which is now about 300 - see Figure 6). As node 3 is still broadcasting its node size to be 100, node 1 now computes $q_{1\_diff} \approx (300 - (300 + 300 + 100)/3) = 67$.

At time t_4, node 2 receives the queue size of node 1 ($= 300$) and updates its local average to be $(300 + 300 + 100)/3 = 233$ so that $q_{2\_diff} \approx 300 - 233 = 67$.

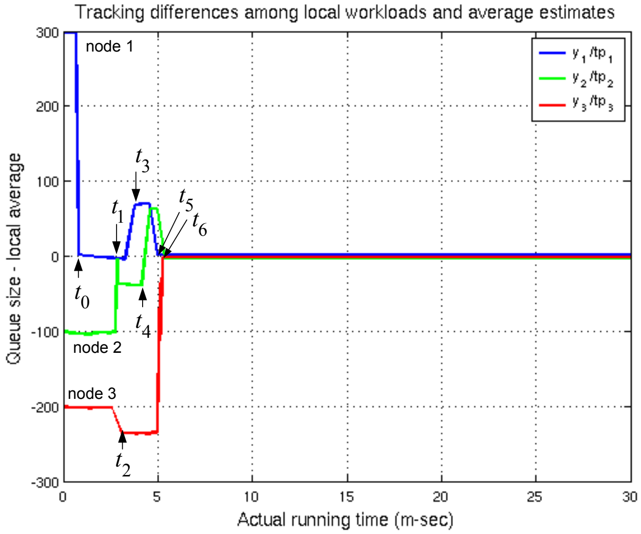

Fig. 8. Experimental plot of $q_{i\_diff}(t) = q_i(t) - \left(\sum_{j=1}^{n} q_j(t - \tau_{ij})\right)/n$ for $i = 1, 2, 3$.

At time t_5, node 3 receives the 200 tasks from node 1 and updates its queue size which is now about 300. The local average computed by node 3 is then $(600 + 200 + 300)/3 = 367$ so that $q_{3\_diff} \approx (300 - 367) = -67$.

Finally, just after t_5 at time t_6, node 1 receives the queue size of node 3 (which is now about 300 - see Figure 6). Node 1 now computes its $q_{1\_diff} \approx (300 - (300 + 300 + 300)/3) = 0$.

Node 2 receives the queue size of node 3 (which is now about 300 - see Figure 6). Node 2 now computes its $q_{2\_diff} \approx (300 - (300 + 300 + 300)/3) = 0$.

Node 3 is now updated with the queue size of node 2 (which is now about 300 - see Figure 6) and now computes $q_{3\_diff} \approx (300 - (300 + 300 + 300)/3) = 0$.

VI. Summary and Conclusions

In this work, a load balancing algorithm was modeled as a nonlinear time-delay system. It was shown that the model was consistent in that the total number of tasks was conserved and the queues were always non negative. It was also shown the system was always stable, but not necessarily asymptotically stable. Experiments were preformed that indicate a correlation of the continuous time model with the actual implementation. Future work will entail considering feedback controllers to speed up the response and it is expected that by the time of the conference, experimental results using them will be presented.

VII. Acknowledgements

The work of J. D. Birdwell, J. Chiasson and Z. Tang was supported by U.S. Department of Justice, Federal Bureau of Investigation under contract J-FBI-98-083 and by the National Science Foundation grant number ANI-0312182. Drs. C. T. Abdallah and M. M. Hayat were supported by the National Science Foundation under grant number ANI-

0312611. Drs. Birdwell and Chiasson were also partially supported by a Challenge Grant Award from the Center for Information Technology Research at the University of Tennessee. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Government.

References

[1] J. D. Birdwell, R. D. Horn, D. J. Icove, T. W. Wang, P. Yadav, and S. Niezgoda, "A hierarchical database design and search method for codis," in *Tenth International Symposium on Human Identification*, September 1999. Orlando, FL.

[2] J. D. Birdwell, T. W. Wang, R. D. Horn, P. Yadav, and D. J. Icove, "Method of indexed storage and retrieval of multidimensional information," in *Tenth SIAM Conference on Parallel Processing for Scientific Computation*, September 2000. U. S. Patent Application 09/671, 304.

[3] J. D. Birdwell, T.-W. Wang, and M. Rader, "The university of Tennessee's new search engine for codis," in *6th CODIS Users Conference*, February 2001. Arlington, VA.

[4] T. W. Wang, J. D. Birdwell, P. Yadav, D. J. Icove, S. Niezgoda, and S. Jones, "Natural clustering of DNA/STR profiles," in *Tenth International Symposium on Human Identification*, September 1999. Orlando, FL.

[5] A. Corradi, L. Leonardi, and F. Zambonelli, "Diffusive load-balancing polices for dynamic applications," *IEEE Concurrency*, vol. 22, pp. 979–993, Jan-Feb 1999.

[6] M. H. Willebeek-LeMair and A. P. Reeves, "Strategies for dynamic load balancing on highly parallel computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 9, pp. 979–993, 1993.

[7] C. K. Hisao Kameda, Jie Li and Y. Zhang, *Optimal Load Balancing in Distributed Computer Systems*. Springer, 1997. Great Britain.

[8] H. Kameda, I. R. El-Zoghdy Said Fathy, and J. Li, "A performance comparison of dynanmic versus static load balancing policies in a mainframe," in *Proceedings of the 2000 IEEE Conference on Decision and Control*, pp. 1415–1420, December 2000. Sydney, Australia.

[9] E. Altman and H. Kameda, "Equilibria for multiclass routing in mult-agent networks," in *Proceedings of the 2001 IEEE Conference on Decision and Control*, December 2001. Orlando, FL USA.

[10] L. Kleinrock, *Queuing Systems Vol I : Theory*. John Wiley & Sons, 1975. New York.

[11] F. Spies, "Modeling of optimal load balancing strategy using queuing theory," *Microprocessors and Microprogramming*, vol. 41, pp. 555–570, 1996.

[12] C. T. Abdallah, N. Alluri, J. D. Birdwell, J. Chiasson, V. Chupryna, Z. Tang, and T. Wang, "A linear time delay model for studying load balancing instabilities in parallel computations," *The International Journal of System Science*, vol. 34, no. 10-11, pp. 563–573, 2003.

[13] M. Hayat, C. T. Abdallah, J. D. Birdwell, and J. Chiasson, "Dynamic time delay models for load balancing, Part II: A stochastic analysis of the effect of delay uncertainty," in *CNRS-NSF Workshop: Advances in Control of Time-Delay Systems, Paris France*, January 2003.

[14] C. T. Abdallah, J. D. Birdwell, J. Chiasson, V. Churpryna, Z. Tang, and T. W. Wang, "Load balancing instabilities due to time delays in parallel computation," in *Proceedings of the 3rd IFAC Conference on Time Delay Systems*, December 2001. Sante Fe NM.

[15] J. D. Birdwell, J. Chiasson, Z. Tang, C. T. Abdallah, M. Hayat, and T. Wang, "Dynamic time delay models for load balancing Part I: Deterministic models," in *CNRS-NSF Workshop: Advances in Control of Time-Delay Systems, Paris France*, January 2003.

[16] J. D. Birdwell, J. Chiasson, Z. Tang, C. T. Abdallah, M. Hayat, and T. Wang, *Dynamic Time Delay Models for Load Balancing Part I: Deterministic Models*, pp. 355–368. CNRS-NSF Workshop: Advances in Control of Time-Delay Systems, Keqin Gu and Silviu-Iulian Niculescu, Editors, Springer-Verlag, 2003.

[17] J. D. Birdwell, J. Chiasson, Z. Tang, C. T. Abdallah, and M. M. Hayat, "The effect of feedback gains on the perfomance of a load balancing network with time delays," in *IFAC Workshop on Time-Delay Systems TDS03*, September 2003. Rocquencourt, France.

[18] J. D. Birdwell, J. Chiasson, C. T. Abdallah, Z. Tang, N. Alluri, and T. Wang, "The effect of time delays in the stability of load balancing algorithms for parallel computations," in *Proceedings of the 42nd IEEE CDC*, December 2003. Maui, Hi.

[19] H. K. Khalil, *Nonlinear Systems Third Edition*. Prentice-Hall, 2002.