1

Closed Loop Control of a Load Balancing Network with Time Delays and Processor Resource Constraints

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

[1] ECE Dept, The University of Tennessee, Knoxville, TN 37996-2100, USA
{ztang,birdwell,chiasson}@utk.edu
[2] ECE Dept, University of New Mexico, Alburquerque, NM 87131-1356, USA
{chaouki,hayat}@ece.unm.edu

Summary. A deterministic dynamic nonlinear time-delay system 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 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. A closed loop controller is proposed to carry out the load balancing dynamically. Experimental results on a parallel computer network are presented to demonstrate the efficacy of the proposed load balancing strategy.

1.1 Introduction

The objective of parallel processing is to reduce wall-clock time and increase the size of solvable problems by dividing the code into multiple fragments that can be executed simultaneously on each of a set of computational elements (CE) interconnected via a high bandwidth network. A common parallel computer 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.

To effectively utilize parallel computer architecture, the computational loads throughout all parallel nodes need 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. To distribute the load evenly is a key activity in producing efficient implementations of applications on parallel architectures.

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 Willebeek-LeMair 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. Previous 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 are always nonnegative 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 system was always asymptotically stable, the new model is now only (Lyapunov) stable, and asymptotic stability must be insured by judicious choice of the feedback.

A controller on any node which uses delayed information from other nodes can cause unnecessary data transfers among the nodes. Here a control law is proposed that uses not only its estimate of the queue size of the other nodes, but also measurements of the number of tasks in transit to it. The communication of the number of tasks being sent to a node is much faster than the actual transfer of the tasks. In each processing loop, each node broadcasts its current queue size as well as the number of tasks it is sending to each of the other nodes (rather than have the other nodes wait for the actual arrival of the tasks to obtain this information). This way each node's estimate of the whole network status will be more up to date when making its control decision. Experimental results on a parallel computer network are presented to demonstrate the efficacy of the load balancing strategy. In particular, load balancing experiments using the controller based on the local queue size only are compared with those using the controller based on the anticipated local queue size, demonstrating a substantial improvement.

Section 1.2 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 1.3 shows that the proposed model correctly predicts that the queue lengths are nonnegative 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 system model. Section 1.4 addresses the feedback control law on a local node and how a node decides to portions out its tasks to the other nodes. Feedback controllers based on the actual queue size and on the *anticipated* queue size are discussed in this section. Section 1.5 presents how the model parameters are obtained and the experimental setup. Section 1.6 presents a comparison of the feedback controller based on the actual queue size with the feedback controller based on the anticipated queue size. This comparison is done with the actual experimental data. Finally, Section 1.7 is a summary and conclusion of the present work.

1.2 Mathematical Model of Load Balancing

In this section, a nonlinear continuous time model is developed to model load balancing among a network of computers. 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[3] of the average number of tasks in the queues of the n computers in the network. 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 or some positive threshold, the node sends some of its tasks to the other nodes. If it is less than zero, no tasks are sent (see Figure 1.1). Further, the tasks sent by node i are received by node j with a delay h_{ij}. The task transfer delay h_{ij} depends on the number of tasks to be transferred and is much greater than the communication delay τ_{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. As just explained, each node controller (load balancing

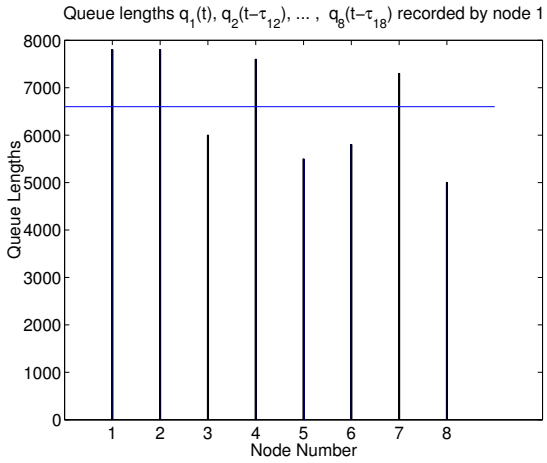


Fig. 1.1. Graphical description of load balancing. This bar graph shows the load for each computer vs. node of the network. The thin horizontal line is the average load as estimated by node 1. Node 1 will transfer (part of) its load only if it is above its estimate of the network average. Also, it will only transfer to nodes that it estimates are below the network average.

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

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.

1.2.1 Basic Model

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

$$\begin{aligned}\frac{dx_i(t)}{dt} &= \lambda_i - \mu_i\left(1-\eta_i(t)\right) - U_m(x_i)\eta_i(t) \\ &\quad + \sum_{j=1}^{n} p_{ij}\frac{t_{p_i}}{t_{p_j}} U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij}) \qquad (1.1)\\ p_{ij} &\geqslant 0, p_{jj} = 0, \sum_{i=1}^{n} p_{ij} = 1\end{aligned}$$

where

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

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).
- $\mu_i \geq 0$ 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 (1.1) 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

$$p_{ji} = 1/(n-1) \text{ for } j \neq i \text{ and } p_{ii} = 0 \tag{1.2}$$

where it is clear that $p_{ji} \geqslant 0, \sum_{j=1}^{n} p_{ji} = 1$. Another approach is to base them on the estimated state of the network; this approach is given in section 1.5.

1.3 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.

1.3.1 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

$$\begin{aligned}\frac{d}{dt}\left(x_i(t)/t_{p_i}\right) = \frac{\lambda_i - \mu_i\left(1-\eta_i(t)\right)}{t_{p_i}} - \frac{1}{t_{p_i}}U_m(x_i)\eta_i(t) \\ + \sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}U_m(x_j(t-h_{ij}))\eta_j(t-h_{ij}).\end{aligned} \tag{1.3}$$

Given that $x_i(0) > 0$ for all i, it follows from the right-hand side of (1.3) 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, $\lambda_i - \mu_i\left(1 - \eta_i(t)\right) = \lambda_i \geq 0$ 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

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

and thus the queues cannot go negative.

1.3.2 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 (1.3) from $i = 1, ..., n$ to get

$$\begin{aligned}\frac{d}{dt}\left(\sum_{i=1}^{n} q_i(t)\right) &= \sum_{i=1}^{n}\left(\frac{\lambda_i - \mu_i(x_i)\left(1 - \eta_i\right)}{t_{p_i}}\right) - \sum_{i=1}^{n}\frac{U_m(x_i(t))}{t_{p_i}}\eta_i \\ &\quad + \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}} U_m(x_j(t - h_{ij}))\eta_j(t - h_{ij}) \end{aligned} \tag{1.4}$$

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

$$\frac{d}{dt} q_{net_i}(t) = -\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}} U_m(x_j(t - h_{ij}))\eta_j(t - h_{ij}) + \sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}} U_m(x_j(t))\eta_j(t). \tag{1.5}$$

Here $q_{net_i} \triangleq \sum_{j \neq i}^{n} q_{net_{ij}}$ is the number of tasks put on the network that are being sent to node i from the other nodes. This equation simply says that the j^{th} node is putting the number of tasks $q_{net_{ij}}$ on the network to be sent to node i at the rate $dq_{net_{ij}}/dt = \left(p_{ij}/t_{p_j}\right) U_m(x_j(t))\eta_j(t)$ while the i^{th} node is taking these tasks from node j off the network at the rate $-p_{ij}/t_{p_j} U_m(x_j(t - h_{ij}))\eta_j(t - h_{ij})$. Summing (1.5) over all the nodes, one obtains

$$\begin{aligned}&\frac{d}{dt}\left(\sum_{i=1}^{n} q_{net_i}(t)\right) \\ &= -\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}} U_m(x_j(t - h_{ij}))\eta_j(t - h_{ij}) + \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}} U_m(x_j(t))\eta_j(t) \\ &= -\sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}} U_m(x_j(t - h_{ij}))\eta_j(t - h_{ij}) + \sum_{j=1}^{n}\frac{U_m(x_j(t))\eta_j(t)}{t_{p_j}}. \end{aligned} \tag{1.6}$$

Adding (1.4) and (1.6), one obtains

$$\frac{d}{dt}\sum_{i=1}^{n}\left(q_i(t)+q_{net_i}(t)\right)=\sum_{i=1}^{n}\left(\frac{\lambda_i-\mu_i\left(1-\eta_i\right)}{t_{p_i}}\right). \tag{1.7}$$

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 $\sum_{i=1}^{n}\mu_i\left(1-\eta_i\right)/t_{p_i}$ at all the nodes. The load balancing itself cannot increase or decrease the total number of tasks in all the queues.

1.3.3 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 (1.1) and (1.5) 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

$$q_{net_i}(t)=\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}\left(\int_{t-h_{ij}}^{t}U_m(x_j(\tau))\eta_j(\tau)d\tau\right)\geq 0. \tag{1.8}$$

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

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

is a positive definite function. Under the conditions of the theorem, equation (1.7) becomes

$$\frac{d}{dt}\sum_{i=1}^{n}\left(q_i(t)+q_{net_i}(t)\right)=-\sum_{i=1}^{n}\frac{\mu_i(q_i/t_{p_i})}{t_{p_i}}\left(1-\eta_i\right) \tag{1.9}$$

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

1.4 Feedback Control

In the work [18], the feedback law at each node i was based on the value of $x_i(t)$ and the *delayed* values $x_j(t-\tau_{ij})$ $(j \neq i)$ from the other nodes. Here τ_{ij} $(\tau_{ii} = 0)$ denote the time delays for communicating the expected waiting time

x_j from node j to node i. These communication delays τ_{ij} are much smaller than the corresponding data transfer delays h_{ij} of the actual tasks. Define

$$x_{i\_avg} \triangleq \left(\sum_{j=1}^{n} x_j(t - \tau_{ij}) \right) / n \tag{1.10}$$

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

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

to be the expected waiting time relative to the estimate of the network average by the i^{th} node.

The control law considered here is

$$\eta_i(t) = h\left(y_i(t)\right) \tag{1.11}$$

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

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

The control law basically states that if the i^{th} node waiting time $x_i(t)$ is above the estimate of the network 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 its estimate of the network 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}.)

However, there is additional information that can be made available to the nodes. Specifically, the information of the tasks that are in the network being sent to the i^{th} node q_{net_i} or equivalently, the waiting time $x_{net_i} \triangleq t_{p_i} q_{net_i}$. Here it is proposed to base the controller not only on the local queue size q_i, but also use information about the number of tasks q_{net_i} being sent to node i. The node j sends to each node i in the network information on the number of tasks $q_{net_{ij}}$ it has decided to send to each of the other nodes in the network. This way the other nodes can take into account this information (without having to wait for the actual arrival of the tasks) in making their control decision. The communication of the number of tasks $q_{net_{ij}}$ being sent from node j to node i is much faster than the actual transfer of the tasks. Furthermore, each node i also broadcasts its total (anticipated) amount of tasks, i.e., $q_i + q_{net_i}$ to the other nodes so that they have a more current

estimate of the tasks on each node (rather than have to wait for the actual transfer of the tasks). The information that each node has will be a more up to date estimate of the state of network using this scheme.

Define

$$z_i \triangleq x_i + x_{net_i} = t_{p_i}\left(q_i + q_{net_i}\right) \tag{1.12}$$

which is the *anticipated* waiting time at node i. Further, define

$$z_{i\_avg} \triangleq \left(\sum_{j=1}^{n} z_j(t-\tau_{ij})\right)/n \tag{1.13}$$

to be the i^{th} node's estimate of the average anticipated waiting time of all the nodes in the network. This is still an estimate due to the communication delays. Therefore,

$$w_i(t) \triangleq x_i(t) - z_{i\_avg}(t) = x_i(t) - \frac{\sum_{j=1}^{n} z_j(t-\tau_{ij})}{n}$$

to be the anticipated waiting time relative to the estimate of average (anticipated) waiting time in the network by the i^{th} node. A control law based on the anticipated waiting time is chosen as

$$\eta_i(t) = h\left(w_i(t)\right). \tag{1.14}$$

The difference between (1.14) and (1.11) will be illustrated in Section 1.6.

1.4.1 Nonlinear Model with Non Constant p_{ij}

The model (1.1) did not have the p_{ij} specified explicitly. For example, they can be considered constant as specified by (1.2). However, it is useful to use the local information of the waiting times $x_i(t), i = 1, .., n$ or the anticipated waiting time $z_i(t), i = 1, .., n$ to set their values.

p_{ij} based on the x_i

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 to be relative to node j's estimate of the network average. If the queue of node i is above the network average as estimated by node j, then node j does not send any tasks to it. Define a saturation function by

$$\text{sat}(x) = \begin{cases} x \text{ if } x > 0 \\ 0 \text{ if } x \leq 0. \end{cases}$$

Then $\text{sat}(x_{j\_avg} - x_i(t - \tau_{ji}))$ is node j's estimate of how much node i is *below* the network average. Node j then repeats this computation for all the other nodes and portions out its tasks among the other nodes according to the amounts they are *below* its estimate of the network average, that is,

$$p_{ij} = \frac{\text{sat}\left(x_{j\_avg} - x_i(t - \tau_{ji})\right)}{\sum_{i \ni i \neq j} \text{sat}\left(x_{j\_avg} - x_i(t - \tau_{ji})\right)}. \tag{1.15}$$

This is illustrated in Figure 1.2.

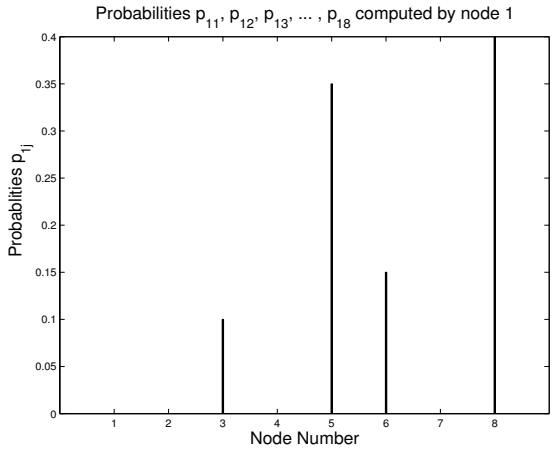


Fig. 1.2. Illustration of a hypothetical distribution p_{i1} of the load at some time t from node 1's point of view. Node 1 will send data out to node i in the proportion p_{i1} that it estimates node i is below the average where $\sum_{i=1}^{n} p_{i1} = 1$ and $p_{11} = 0$

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

$$x_{j\_avg} - x_j(t) = -\sum_{i \ni i \neq j} \left(x_{j\_avg} - x_i(t - \tau_{ji})\right) \geq 0.$$

That is, if the denominator is zero, the expected waiting time on node j is not greater than its estimate of the network average and therefore it is not sending out any tasks.

p_{ij} based on the z_i

Similarly, the p_{ij} can be specified using the anticipated waiting time z_j of the other nodes. The quantity $z_{j\_avg} - z_i(t-\tau_{ji})$ represents what node j estimates the network's average anticipated waiting time is relative to its estimate of the anticipated waiting time in the queue of node i. If the estimate of the queue of node i (i.e., $z_i(t-\tau_{ji})$) is above what node j estimates the network's average (i.e., z_{j_avg}) is, then node j sends tasks to node i. Otherwise, node j sends no tasks to node i. Therefore $\text{sat}(z_{j\_avg} - z_i(t-\tau_{ji}))$ is a 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 its estimate of the network average, that is,

$$p_{ij} = \frac{\text{sat}\left(z_{j\_avg} - z_i(t-\tau_{ji})\right)}{\sum_{i\ni i\neq j} \text{sat}\left(z_{j\_avg} - z_i(t-\tau_{ji})\right)}. \tag{1.16}$$

It is obvious that $p_{ij} \geqslant 0, \sum_{i=1}^{n} p_{ij} = 1$ and $p_{jj} = 0$. All p_{ij} are defined to be zero, and no load is transferred if the denominator is zero, i.e., if

$$\sum_{i\ni i\neq j} sat\left(z_{j\_avg} - z_i(t-\tau_{ji})\right) = 0.$$

As in the above remark, it is easy to see that in this case node j would be below (its estimate of) the network average.

1.5 Model Parameters and Experimental Setup

1.5.1 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 1.3 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

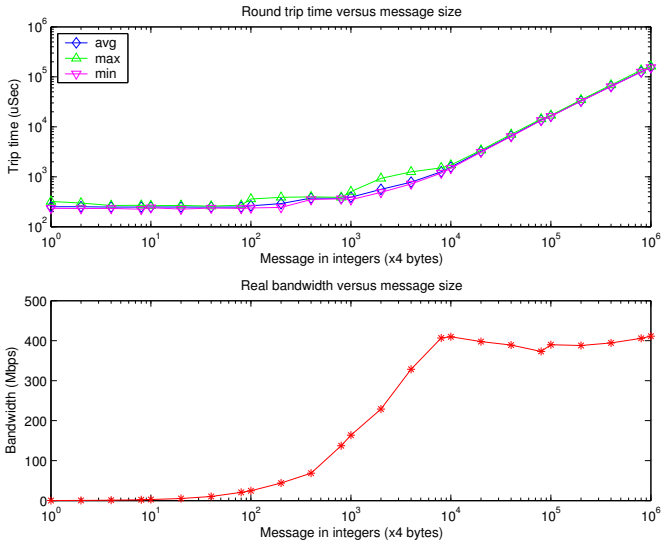


Fig. 1.3. Top: Round trip time vs. amount of data transfered in bytes. Bottom: Measured bandwidth vs. amount of data transfered in bytes.

Mbytes. In order to ensure that anomalies in message timings are minimized the tests are repeated 20 times for each message size. The bottom plot in Figure 1.3 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 these data are transferred at a bandwidth of about 400 Mbps. Messages of larger sizes will not improve the bandwidth and result in higher task transfer delays. That means the system must yield more computing time for communication time. Here the threshold is chosen to be 4×10^3 bytes since, as the top of Figure 1.3 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. The hysteresis threshold y_0 is then

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

while the bandwidth constraint U_{m0} is given by

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

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

where the average processing time $t_{pi} = 400\mu$sec is used.

1.5.2 Experimental Setup of the Parallel Machine

A parallel machine has been built and used as an experimental facility for evaluation of load balancing strategies. A root node communicates with k groups of networked computers. 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. The search engine threads access tree-structured inidices to locate database records that match search or store requests. The structure of the network is shown in Figure 1.4. Due to the structure of the search process, search requests can be

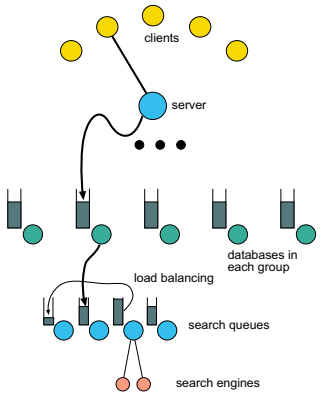


Fig. 1.4. A depiction of multiple search threads in the database index tree. To even out the search queues, load balancing is done between the nodes (hosts) of a group. If a node has a dual processor, then it can be considered to have two search engines for its queue.

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.

1.6 Experimental Results

Experiments are presented to indicate the effects of time delays in load balancing. In the first set of experiments, the load balancing is performed *once at a fixed time* and is referred to as an "open loop" run. This open loop experiment is done in order to facilitate an explanation of the system dynamics with the effects of the delays. The open loop experiments are done for two cases. The first case uses the p_{ij} specified by (1.15) which are based on the x_i and the second case uses the p_{ij} specified by (1.16) which are based on the z_i.

In the second set of experiments, the closed loop controller (1.11) with the p_{ij} specified by (1.15) is compared with the closed loop controller (1.14) with the p_{ij} specified by (1.16).

1.6.1 Open Loop Experiments

Case 1: p_{ij} based on x_i

Figure 1.5 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 $t_{p_i} = 400\ \mu$sec. In this open loop experiment, the software was written to execute the load balancing algorithm at $t_0 = 1$ millisec using the p_{ij} as specified by (1.15). Figure 1.5 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 $h_{31} = 2.8$ millisec. In this experiment the inputs were set as $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 0$. All time symbols for this experimental run are shown in Figure 1.7.

Figure 1.6 is a plot of each nodes' estimate of the network average, i.e.,

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

where $q_{i\_avg}(t) = x_{i\_avg}(t)/t_{p_i}$ (see (1.10)). Figure 1.7 is a plot of the queue size relative to its estimate of the network average, i.e.,

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

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. Figure 1.6 is now discussed in detail as follows.

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

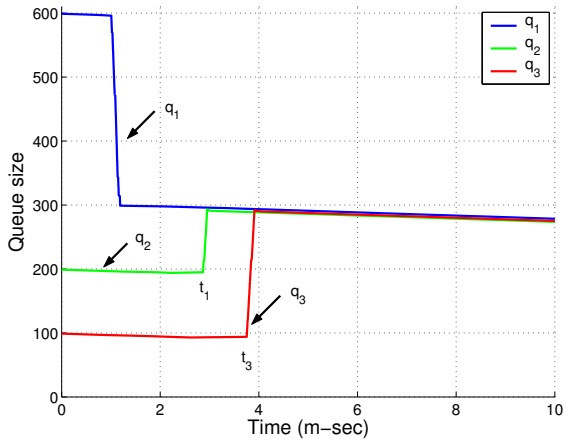


Fig. 1.5. Results of the load balancing algorithm executed at $t_0 = 1$ msec.

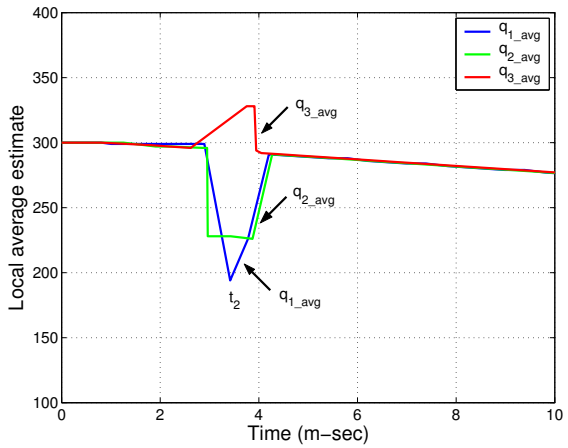


Fig. 1.6. Plot of $q_{i\_avg}(t)$ for $i = 1, 2, 3$.

the network average q_{3_diff} to be -200. Node 1 sends tasks out according to (1.15), which transfers about 100 and 200 tasks to node 2 and node 3 respectively.

At time t_1 node 2 receives 100 tasks from node 1 and updates its local queue size to be 300 as well as its estimate of node 1's queue size to be (about) 300, so that node 2's estimate of the network average is now $(300+300+100)/3 \approx 233$ so that its estimate of its queue size relative to the estimate of the network

average is now $q_{2\_diff} \approx 300 - 233 = 67$. That is, node 2 is not aware of the 200 tasks being sent from node 1 to node 3.

At time t_2 node 1 has already sent out the tasks to node 2 and 3 and updated to its local queue size to 300. Node 1 receives the broadcast of two queue sizes of node 2 and 3 which are still 200 and 100 respectively, as the task transfers from node 1 are still on the way to node 2 and 3. Thus, node 1's estimate of the network average is about $(300+200+100)/3 = 200$ making the computation of its queue size relative to the estimate of the network average now $q_{1\_diff} \approx 300 - 200 = 100$.

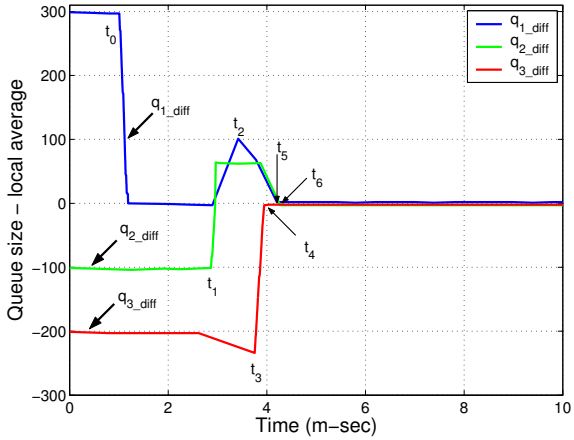


Fig. 1.7. Plot of $q_{i\_diff}(t) = q_i(t) - \left(\sum_{j=1}^{n} q_j(t-\tau_{ij})\right)/n$ for $i = 1, 2, 3$ using p_{ij} specified in (1.15).

At time t_3, node 3 receives the queue size of node 2 (which has already increased to about 300 as shown in Figure 1.5). Node 3 now computes q_{3_diff} to be about $(100 - (600 + 300 + 100)/3) \approx -233$.

At time t_4, node 3 finally receives the 200 tasks from node 1 and updates its queue size and its estimate of node 1's queue size to be about 300. The network average computed by node 3 is now about $(300 + 300 + 300)/3 = 300$ so that $q_{3\_diff} \approx (300 - 300) = 0$.

At time t_5, node 1 receives the communications for the queue sizes of node 2 and node 3 (which is now about 300 - see Figure 1.5). Then node 1 computes $q_{1\_diff} \approx (300 - (300 + 300 + 300)/3) = 0$.

Finally, at time t_6, node 2 receives the queue size of node 3 (which is now about 300 - see Figure 1.5). Node 2 now computes its queue size relative to local average $q_{1\_diff} \approx (300 - (300 + 300 + 300)/3) = 0$.

The transfer of 200 tasks to node 3 takes more time than that of 100 tasks to node 2. It is the task transfer delay that delays node 2's receipt of the new queue size of node 3 at time t_1, and that delays node 3's receipt of the tasks until $t_4 > t_3$. It is the communication delay that delays node 1's receipt of the new queue size of node 2 at time t_2. The effect of both delays could cause the unnecessary transfers for node 2 at time t_1 and node 1 at time t_2 if the load balancing is done in closed loop.

Case 2: p_{ij} based on z_i

The node that transfers tasks computes the amounts to be sent to other nodes accoding to how far below the network average its estimate of each recipient node's queue is. Therefore, it is feasible to send the amounts of its next task transfers to each of other nodes before actually transferring tasks. Such communications are efficient; the communication delay of each transferred measurement is much smaller than the actual task transfer delays. A key finding of this research is that knowledge of the anticipated queue sizes can be used to compensate the effect of delays. Figure 1.8 is a plot of the local average of queue sizes on all nodes using measurements of anticipated transfers

$$q^{est}_{i\_avg}(t) \triangleq \left(\sum_{j=1}^{n} q^{est}_{j}(t - \tau_{ij}) \right) /n = \left(\sum_{j=1}^{n} q_j(t - \tau_{ij}) + q_{net_j}(t) \right) /n$$

where $q_{net_j}(t) = \sum_{i=1}^{n} q_{net_{ji}}(t-\tau_{ij})$. The nodes have 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 $t_{p_i} = 400\ \mu$sec. Note that $q^{est}_{i\_avg}(t) = z_{i\_avg}(t)/t_{p_i}$ (see (1.13)). In this experiment, the load balancing algorithm was executed at $t_0 = 1$ millisec using the p_{ij} as specified by (1.16) with $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 0$. Compared with Figure 1.6, the estimates by the nodes of the network average are substantially improved. Figure 1.9 is a plot of the queue size relative to the local average, i.e.,

$$q^{est}_{i\_diff}(t) \triangleq q_i(t) - q^{est}_{i\_avg}(t) = q_i(t) - \left(\sum_{j=1}^{n} q^{est}_{j}(t - \tau_{ij}) \right) /n$$

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 it determines that whether it is above or below the network average is greatly diminished. Compared with Figure 1.7, the tracking differences are near or below zero and no further unnecessary transfers are initiated. Figures 1.10(a) and 1.10(b) show the estimates of the queue sizes by node 2 when using x_i and when using z_i, respectively. In Figure 1.10(a), after node 2 received the tasks from node 1, node 2 computed its queue size relative to its local average to be about 67.

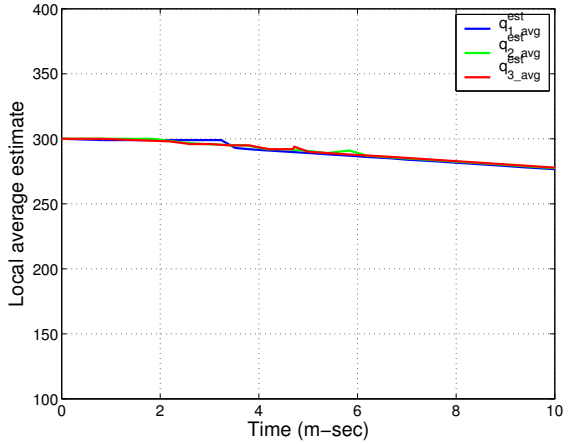


Fig. 1.8. Plot of $q^{est}_{i\_avg}(t) = \left(\sum_{j=1}^{n} q^{est}_{j}(t-\tau_{ij})\right)/n$ for $i = 1, 2, 3$ using p_{ij} by (1.16).

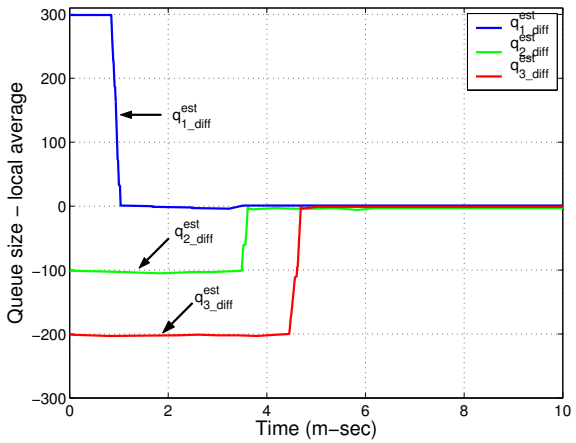


Fig. 1.9. Plot of $q^{est}_{i\_diff}(t) = q_i(t) - \left(\sum_{j=1}^{n} q^{est}_{j}(t-\tau_{ij})\right)/n$ for $i = 1, 2, 3$ using p_{ij} by (1.16).

In Figure 1.10(b), node 1 sends the numbers of tasks to be transferred before actually transferring tasks to the other nodes; node 2 receives the estimate of node 3's queue size and updates its local queue size relative to its estimate of the network average to be zero.

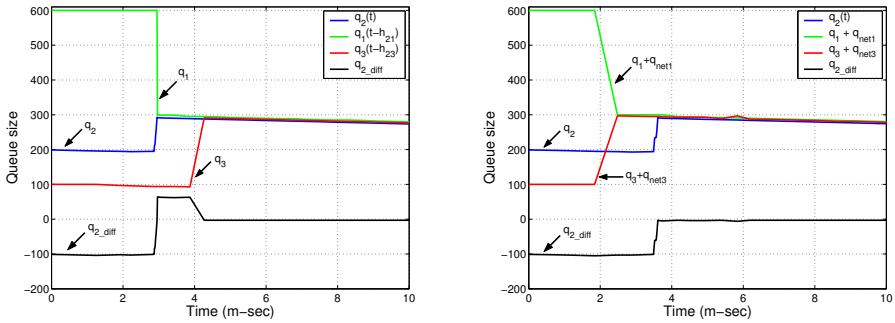
Figures 1.11(a) and 1.11(b) show the estimates of the queue sizes by node 3 when using x_i and when using z_i, respectively. In Figure 1.11(a), when node 2 broadcasts its updated queue size to node 3, node 3 has not received the tasks sent from node 1 due to transfer delays. Node 3 updates its queue size relative to its estimate of the network average to be -233. In Figure 1.11(b), node 1 sends the numbers of tasks to be transferred before transferring actual tasks to the other nodes; node 3 receives the anticipated estimate of the queue sizes on other nodes, and computes its queue size relative to its estimate of the network average to -200. When node 3 receives the tasks transferred from node 1, it updates its local queue and computes the amount relative to the network average to be zero. Compared with Figures 1.10 and 1.11 shown above, these anticipated estimates significantly compensate the effect of delays of task transfers. The comparison of local queue sizes and anticipated estimates on the two recipient nodes 2 and 3 are shown in Figures 1.12 (a) and (b) respectively.

1.6.2 Closed Loop Experiments

The following experiments show the responses using closed loop load balancing 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. After initial communications, the closed loop load balancing algorithm (see Section 1.4) was initiated using the p_{ij} as specified by (1.15) and by (1.16), respectively. In these experiments the inputs were set as $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 0$. Figures 1.13(a) and (b) show the responses of the queues versus time using p_{ij} specified in (1.15) and p_{ij} specified in (1.16) respectively. Note the substantial difference in the load balancing performance between these two schemes.

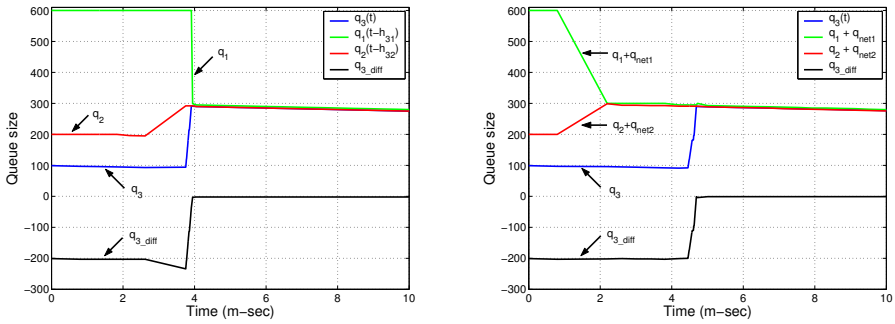
The tracking differences between local queue sizes and estimated network averages (from delayed values of other nodes) using p_{ij} by (1.15) are shown in Figure 1.14(a). There are unnecessary exchanges of tasks back and forth among nodes. Although the system reaches a balanced condition at around $t = 14$ millisec, those additional transfers cost processing time and prolong the completion time. Figure 1.14(b) shows the tracking differences between local queue sizes and anticipated estimates of the network averages using p_{ij} by (1.16) based on z_i.

Figure 1.15(a) shows node 2's estimates of the queue sizes in the network of closed loop load balancing using p_{ij} based on the x_i. Node 2 estimates the network average using the delayed information from other nodes only, and its controller based on its queue size relative to the estimated average causes those unnecessary exchanges of tasks back and forth, as shown in Figure 1.15(a). Figure 1.15(b) shows node 2's estimates of the anticipated queue sizes in the network of closed loop load balancing using p_{ij} based on the z_i. Node 1 sends the numbers of tasks before actually transferring tasks to the other nodes. Node 2 receives the anticipated estimate of node 3's queue size and updates its queue size relative to its estimate of the network average as zero. From the



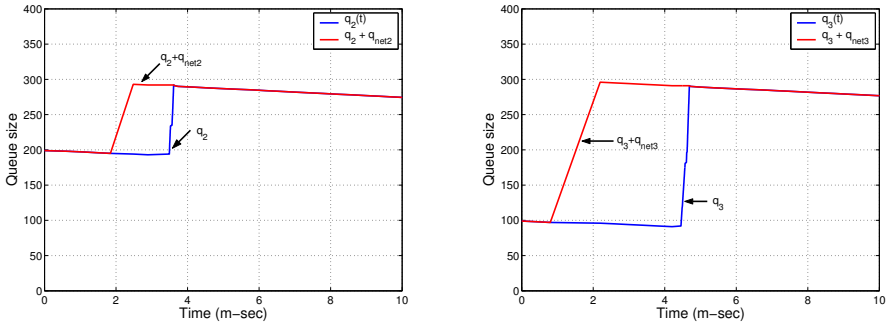
(a) node 2: using x_i (b) node 2: using z_i

Fig. 1.10. Plot of the estimates by node 2 of the queue sizes (a) using x_i and (b) using z_i.



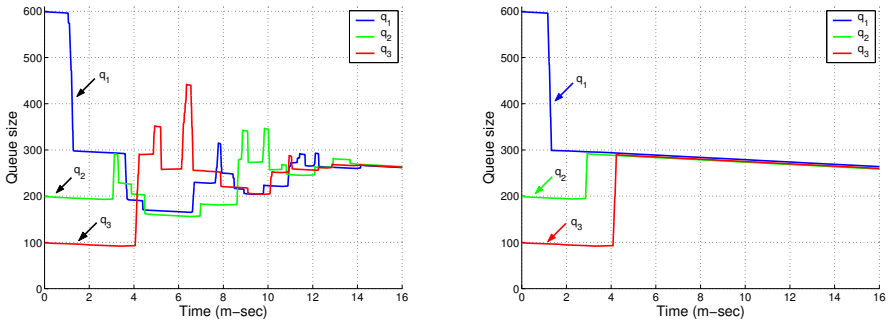
(a) node 3: using x_i (b) node 3: using z_i

Fig. 1.11. Plot of the estimates by node 3 of the queue sizes (a) using x_i and (b) using z_i.



(a) node 2 (b) node 3

Fig. 1.12. Comparison of the actual queue size and the anticipated estimate of queue size on (a) node 2 and (b) node 3.



(a) queue sizes: using x_i (b) queue sizes: using z_i

Fig. 1.13. Results of closed loop load balancing. (a) Plot of queue size using x_i and (1.15) and (b) queue sizes using z_i and (1.16).

Figure 1.15, we can see that the anticipated estimates are used to compensate the effect of delays of task transfers so that there are no unnecessary task transfers initiated.

1.7 Summary and Conclusions

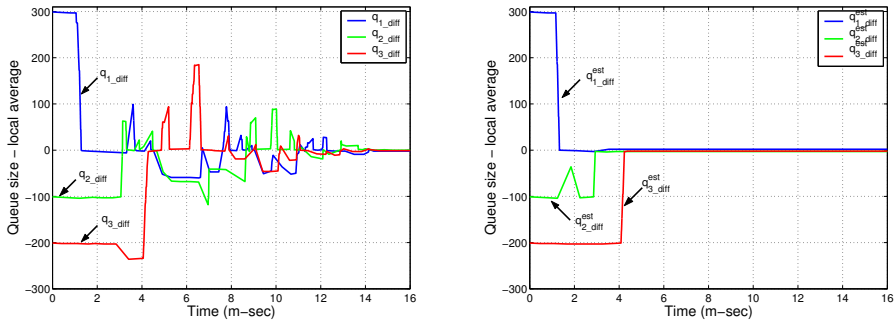
In this work, a load balancing algorithm was modeled as a nonlinear time-delay system. It was shown that the model was consistent in that the total number of tasks was conserved and the queues were always non negative. It was also shown the system was always stable, but not necessarily asymptotically stable. Furthermore, a closed loop controller was proposed based on the local queue size and estimate of the tasks being sent to the queue from other nodes. Experimental results showed a dramatic increase in performance obtained in using this information in the control law.

1.8 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. Birdwell and Chiasson were also partially supported by a Challenge Grant Award from the Center for Information Technology Research at the University of Tennessee. Drs. C. T. Abdallah and M. M. Hayat were supported by the National Science Foundation under grant number ANI-0312611. 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.

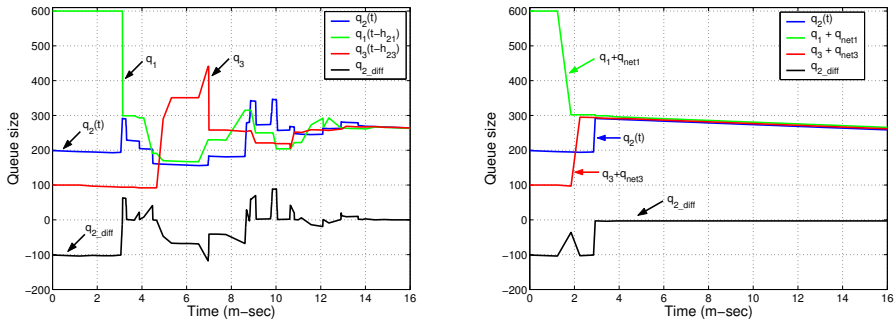
References

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.



(a) $q_{i\_diff}(t)$ by (1.15) based on x_i (b) $q^{est}_{i\_diff}(t)$ by (1.16) based on z_i

Fig. 1.14. Plot of tracking difference between the queue size and the estimated network average on each of the nodes in closed loop load balancing. (a) $q_{i\_diff}(t) = q_i(t) - \left(\sum_{j=1}^{n} q_j(t-\tau_{ij})\right)/n$ and (b) $q^{est}_{i\_diff}(t) = q_i(t) - \left(\sum_{j=1}^{n} q^{est}_j(t-\tau_{ij})\right)/n$ for $i = 1, 2, 3$.



(a) node 2: using x_i (b) node 2: using z_i

Fig. 1.15. Plot of estimated queue sizes by node 2 in closed loop load balancing (a) using x_i and (1.15), and (b) using z_i and (1.16).

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 dynamic versus static load balancing policies in a mainframe," in *Proceedings of the 39th IEEE Conference on Decision and Control*, pp. 1415–1420, December 2000. Sydney, Australia.
9. E. Altman and H. Kameda, "Equilibria for multiclass routing in multi-agent networks," in *Proceedings of the 40th IEEE Conference on Decision and Control*, pp. 604–609, 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, pp. 563–573, August-September 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. Chupryna, 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 performance 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 Conference on Decision and Control*, December 2003. Maui, Hi.
19. H. K. Khalil, *Nonlinear Systems Third Edition*. Prentice-Hall, 2002.