The effect of time delays on the stability of load balancing algorithms for parallel computations

John Chiasson

Zhong Tang

Jean Ghanem

Chaouki T. Abdallah

J. Douglas Birdwell

See next page for additional authors

Authors
John Chiasson, Zhong Tang, Jean Ghanem, Chaouki T. Abdallah, J. Douglas Birdwell, Majeed M. Hayat, and Henry Jérez

The effect of time delays on the stability of load balancing algorithms for parallel computations

John Chiasson
Electrical and Computer Engineering Department, University of Tennessee, Knoxville, TN
Zhong Tang
Electrical and Computer Engineering Department, University of Tennessee, Knoxville, TN
Jean Ghanem
Electrical and Computer Engineering Department, University of New Mexico, Albuquerque, NM
Chaouki T. Abdallah
Electrical and Computer Engineering Department, University of New Mexico, Albuquerque, NM
J. Douglas Birdwell
Electrical and Computer Engineering Department, University of Tennessee, Knoxville, TN
Majeed M. Hayat

Electrical and Computer Engineering Department, University of New Mexico, Albuquerque, NM

Henry Jérez

Electrical and Computer Engineering Department, University of New Mexico, Albuquerque, NM

Abstract:

A deterministic dynamic nonlinear time-delay system is developed to model load balancing in a cluster of computer nodes used for parallel computations. 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). Further, it is shown that using the proposed load balancing algorithms, the system is stable in the sense of Lyapunov. 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 distributed computing network.

SECTION I. Introduction

Distributed computing architectures utilize a set of computational elements (CEs) to achieve performance that is not attainable on a single CE. 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 a background on mathematical treatments of load balancing, the reader is referred to [1] [2] [3]. 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. [4]–[5][6][7] 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. In this type of application, the search itself, initiated on any particular node, can initiate subsequent new searches which are added to the node's queue. Consequently, it is of great advantage to the overall system to carry out load balancing to make effective use of the overall computational resources. The projected growth of the NDIS database and the demand for searches of the database can be met by migration to a parallel computing platform.

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 therefore 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 [8]. 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 [9] and gradient methods, stochastic, or optimization based. A comparison of several deterministic methods is provided by Willebeek-LeMair and Reeves [10]. Here, a deterministic model is developed.

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. Motivated by the authors' previous work in [11] and [12], a new nonlinear model is developed here (see also [13]). Specifically, a deterministic dynamic nonlinear time-delay system is developed to model load balancing. The model is shown to be self consistent in that the queue lengths cannot go negative and that the total number of tasks in all the queues and the network is conserved (i.e., load balancing can neither create nor lose tasks). Further, it is shown that the controller proposed here is asymptotically stable in the sense of Lyapunov. Simulations of the nonlinear model are compared with an *experimental implementation* of the load balancing algorithm performed on a distributed computing network.

Section II presents our approach to modeling the computer network and load balancing algorithm to incorporate the presence of delay in the communication between nodes and task transfers. In Section III, we show that the model captures the nonnegativity of the queue lengths as well as the fact that the totality of tasks in all the queues and in transit is conserved by the load balancing algorithm. Section IV shows that the system is asymptotically stable in the sense of Lyapunov for any choice of positive gains in the load balancing algorithm (controller). Section V presents simulations of the nonlinear models for comparison with the actual experimental data. Section VI presents experimental results from an implementation of the load balancing controller on a parallel computer consisting of a networked cluster of nodes. Section VII presents experiments conducted over a geographically-dispersed distributed environment (i.e., PlanetLab). Both the effects of the network delays and the variances in the task processing time on the behavior of the system are assessed. Finally, Section VIII is a summary and conclusion of the present work and a discussion of future work.

SECTION II. Mathematical Model

Continuous time models are developed in this section that model the load balancing dynamics among a network of computers. To introduce the approach, 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 each of which has essentially the same processing time (homogenous tasks). However, in some applications 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 (broadcasts) its queue size *qj*(*t*) to all other computers in the network. A node *i* receives this information from node *j* delayed by a finite amount of time *τij* (with the convention *τii*=0); that is, it receives *qj*(*t*−*τij*). Each node *i* then uses this information to compute its estimate of the network average of the number of tasks in all *n* queues of the network. Based on the most recent observations, the simple (local) estimate of the network average is computed by the *i*th node as

$$q_i^{avg} \triangleq \frac{\sum_{j=1}^{n} q_j(t-\tau_{ij})}{n}.$$

Node *i* then compares its queue size *qi*(*t*) with its estimate of the network average by estimating its excess load, *qi*(*t*)−(∑*nj*=1*qj*(*t*−*τij*))/*n*. If its excess load 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. Further, the tasks sent by node *i* are received by node *j* with a delay *hij*. The controller (load balancing algorithm) decides how often to do load balancing (transfer tasks among the nodes) and how many tasks are to be sent to each node.

The mathematical model of the task load dynamics at a given computing node is given by

$$\begin{aligned}\frac{d}{dt}x_i(t) = &\quad \lambda_i - \mu_i + u_i(t) - \sum_{j=1}^{n} p_{ij}\frac{t_{p_i}}{t_{p_j}}u_j(t-h_{ij}) \\ y_i(t) = &\quad x_i(t) - \frac{\sum_{j=1}^{n} x_j(t-\tau_{ij})}{n} \\ u_i(t) = &\quad -K_i \mathrm{sat}(y_i(t))\end{aligned} \tag{1}$$

where *pij*≥0, *pjj*=0, satisfy $\sum_{i=1}^{n} p_{ij} = 1$, and

$$\mathrm{sat}(y) = \begin{cases} y_{\max} & \text{if } y > y_{\max} \\ y & \text{if } 0 \leqslant y \leqslant y_{\max} \\ 0 & \text{if } y < 0. \end{cases}$$

Further, in this model, we define

1. *xi*(*t*) is the expected waiting time experienced by a task inserted into the queue of the *i*th node. With *tpi* as the average time needed to process a task on the *i*th node, the expected (average) waiting time is given by *xi*(*t*)=*qi*(*t*)*tpi*. Note that *xj*/*tpj*=*qj* is the number of tasks in the queue of node *j*. If these tasks were transferred to node *i*, then the waiting time transferred is *qjtpi*=*xjtpi*/*tpj*, so that the fraction *tpi*/*tpj* converts waiting time on node *j* to waiting time on node *i*.

2. *λi*≥0 is the rate of generation of waiting times on the *i*th node caused by the addition of tasks (rate of increase in *xi*).

3. *μi*≥0 is the rate of reduction in waiting time caused by the service of tasks at the *i*th node and is given by *μi*≡(1×*tpi*)/*tpi*=1 for all *i* if *xi*(*t*)>0, while if *xi*(*t*)=0 then *μi*=Δ0; that is, if there are no tasks in the queue, then the queue cannot possibly decrease.

4. *ui*(*t*) is the rate of removal (transfer) of the tasks from node *i* at time *t* by the load balancing algorithm at node *i*. Note that *ui*(*t*)≤0.

5. *pij* is the fraction of the *j*th node's tasks to be sent out that it sends to the *i*th node. In more detail, *pijuj*(*t*) is the rate at which node *j* sends waiting time (tasks) to node *i* at time *t* where, as all the tasks must go to some node, one requires that *pij*⩾0, ∑*ni*=1*pij*=1 and *pjj*=0. That is, the transfer from node *j* of expected waiting time (tasks) ∫*t*2*t*1*uj*(*t*)*dt* in the interval of time [*t*1,*t*2] to the other nodes is carried out with the *i*th node receiving the fraction *pij*(*tpi*/*tpj*)∫*t*2*t*1*uj*(*t*)*dt*, where the ratio *tpi*/*tpj* converts the task from waiting time on node *j* to waiting time on node *i*. As ∑*ni*=1*pij*∫*t*2*t*1*uj*(*t*)*dt*=∫*t*2*t*1*uj*(*t*)*dt*, this results in removing *all* of the excess waiting time ∫*t*2*t*1*uj*(*t*)*dt* from node *j*.

6. The quantity −*pijuj*(*t*−*hij*) is the rate of increase (rate of transfer) of the expected waiting time (tasks) at time *t* from node *j* by (to) node *i* where *hij* (*hii*=0) is the time delay for the task transfer from node *j* to 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. This normalization of the queue length (i.e., *xi*=*qi*/*tpi*) at each node by the local average processing time (*tpi*) is simply a way to account for unequal task processing rates by each node. As *ui*(*t*)≤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. The control law *ui*(*t*)=−*Ki*sat(*yi*(*t*)) states that if the *i*th node output *xi*(*t*) is above its estimate of the network average *xavgi*=(∑*nj*=1*xj*(*t*−*τij*))/*n*, then it sends data to the other nodes, while if it is less than this average nothing is sent. The *j*th node receives the fraction ∫*t*2*t*1*pji*(*tpi*/*tpj*)*ui*(*t*)*dt* of transferred waiting time ∫*t*2*t*1*ui*(*t*)*dt* delayed by the time *hij*.

A. Specification of the Factors *pij*

The model described in (1) is the basic model, but an important detail remains unspecified, namely the exact form of the *pij* for each sending node *i*. One approach is to choose them as constant and equal, that is, *pij*=1/(*n*−1) for *j*≠*i*. Another approach is to use the local information of the waiting times *xi*(*t*), *i*=1,…,*n* to set their values. The quantity *xi*(*t*−*τji*)−*xavgj* is node *j*'s estimate of the excess (or deficit) waiting time in the queue of node *i* with respect to the local average of node *j*. If node *i*'s queue is above the local average, then node *j* does not send tasks to it. Therefore, sat(*xavgj*−*xi*(*t*−*τji*)) is a measure by node *j* as to how much node *i* is *below* the local average. Node *j* performs 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

$$p_{ij} \overset{\Delta}{=} \frac{\text{sat}(x_j^{avg} - x_i(t-\tau_{ji}))}{\sum_{i \ni i \neq j} \text{sat}\,(x_j^{avg} - x_i(t-\tau_{ji}))}. \tag{2}$$

View Source If the denominator $\sum_{i \ni i \neq j} \text{sat}\,(x_j^{avg} - x_i(t-\tau_{ji})) = 0$, then the *pij* are defined to be zero and no load is transferred. This is illustrated in Fig. 1 for node 1.

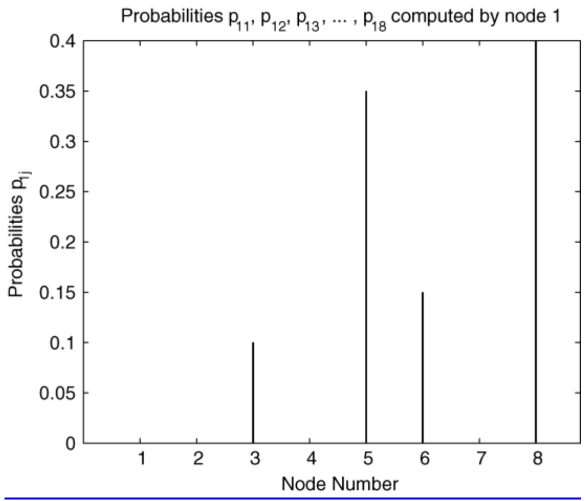


Fig. 1. Illustration of a hypothetical distribution *pi*1 of the load at some time *t* from node 1's point of view. Node 1 will send data out to node *i* in proportion *pi*1 it estimates node *i* is below the average where ∑*ni*=1*pi*1=1 and *p*11=0.

Remark

If the denominator

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

is zero, then *xavgj*−*xi*(*t*−*τji*)≤0 for all *i*≠*j*. However, by definition of the average

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

which implies

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

That is, if the denominator is zero, the node *j* is not greater than the local average, so $u_j(t)=-K_j\text{sat}(y_j(t))=$ 0 and is therefore not sending out any tasks.

SECTION III. Model Consistency

It is now shown that the 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 tasks between nodes [13], [14].

A. Nonnegativity of the Queue Lengths

To show the nonnegativity of the queue lengths, recall that the queue length of each node is given by *qi*(*t*)=*xi*(*t*)/*tpi*. The model is rewritten in terms of these quantities as

$$\frac{d}{dt}\left(\frac{x_i(t)}{t_{p_i}}\right)=\frac{\lambda_i-\mu_i}{t_{p_i}}+\frac{1}{t_{p_i}}u_i(t)-\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}u_j(t-h_{ij}).\tag{3}$$

Given that *xi*(0)>0 for all *i*, it follows from the right-hand side of (3) that *qi*(*t*)=*xi*(*t*)/*tpi*≥0 for all *t*≥0 and all *i*. To see this, suppose without loss of generality that *qi*(*t*)=*xi*(*t*)/*tpi* is the first queue to go to zero, and let *t*1 be the time when *xi*(*t*1)=0. At the time *t*1, $\lambda_i-\mu_i=\lambda_i\geq 0$ by the definition of *μi* and $-\sum_{j=1}^{n}(p_{ij}/t_{p_j})u_j(t-h_{ij})\geq$ 0 for all time by the definition of the *uj*. Further, the term *ui*(*t*1) is negative only if

$$x_i(t_1)>\frac{\left(\sum_{j=1}^{n}x_j\left(t_1-\tau_{ij}\right)\right)}{n}.\tag{4}$$

By supposition (up to time *t*1) all the *xj*(*t*1−*τij*)>0 for *j*≠*i* and *xi*(*t*1)=0 so that *ui*(*t*1)=0 as the right side of (4) is positive at time *t*1. Consequently, at time *t*1 all terms on the right-hand side of (3) are nonnegative. Further, *xi*(*t*) cannot go negative in a neighborhood of *t*1. For if it did, as the right-hand side of (4) is continuous, it follows that:

(5)

$$x_i(t) < 0 < \frac{\left(\sum_{j=1}^{n} x_j(t - \tau_{ij})\right)}{n}$$

for some *t*∈(*t*1,*t*1+*δ*) with *δ*>0. Therefore, *ui*(*t*)=0 for all *t*∈[*t*1,*t*1+*δ*] and the right-hand side of (3) is nonnegative for all *t*∈[*t*1,*t*1+*δ*] which contradicts *xi*(*t*)<0. Note that *t*1+*δ* can be taken to be at least as large as the time at which some *xk* goes to zero, that is, *qk*(*t*1+*δ*)=0 as the right-hand side of (5) must remain positive for *t*∈[*t*1,*t*1+*δ*].

If *xi*(*t*) goes positive after *t*1, then the previous argument is repeated at the next time a queue goes to zero. If *xi*(*t*) remains identically zero in the interval (*t*1, *t*1+*δ*), then the argument is also similar in that at time *t*1+*δ*, both *xi*(*t*1+*δ*), *xk*(*t*1+*δ*) are then zero. As the remaining *n*−2 nodes are still positive, the right-hand side of (5) continues to hold with both *xi* and *xk* zero at time *t*1+*δ* and one again gets a contradiction if either *xi* or *xk* goes negative in an interval (*t*1+*δ*, *t*2). Continuing in this manner, it follows that *qi*(*t*)=*xi*(*t*)/*tpi* cannot go negative for all *i*.

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 (3) from *i*=1,…,*n* to obtain

(6)

$$\frac{d}{dt}\left(\sum_{i=1}^{n} q_i(t)\right) = \sum_{i=1}^{n}\left(\frac{\lambda_i - \mu_i}{t_{p_i}}\right) + \sum_{i=1}^{n} u_i(t)/t_{p_i} - \sum_{i=1}^{n}\sum_{j=1}^{n} \frac{p_{ij}}{t_{p_j}} u_j(t - h_{ij})$$

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

(7)

$$\frac{d}{dt} q_{\mathrm{net_i}}(t) = \sum_{j=1}^{n} \frac{p_{ij}}{t_{p_j}} u_j(t - h_{ij}) - \sum_{j=1}^{n} \frac{p_{ij}}{t_{p_j}} u_j(t).$$

Here *q*neti 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 (*pij*/*tpj*)*uj*(*t*) while the *i*th node is taking these tasks from node *j* off the network at the rate (*pij*/*tpj*)*uj*(*t*−*hij*). Summing (7) over all the nodes, one obtains

(8)

$$\frac{d}{dt}\left(\sum_{i=1}^{n} q_{\mathrm{net_i}}(t)\right) = \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}u_j(t-h_{ij}) - \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}u_j(t)$$

$$= \sum_{i=1}^{n}\sum_{j=1}^{n}\frac{p_{ij}}{t_{p_j}}u_j(t-h_{ij}) - \sum_{j=1}^{n}\frac{u_j(t)}{t_{p_j}}.$$

Adding (6) and (8), one obtains the conservation of queue lengths given by

(9)

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

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 ∑*ni*=1*λi*/*tpi* at all the nodes, or similarly, decrease by the rate of processing of tasks ∑*ni*=1*μi*/*tpi* at all the nodes. The load balancing itself cannot increase or decrease the total number of tasks in all the queues.

SECTION IV. Stability of the Controller

The controller in the model (1) is

$$u_i(t) = -K_i \mathrm{sat}\left(y_i(t)\right)$$

where the gains *Ki*>0 are to be specified. Physically, these gains are limited by the bandwidth constraints in the network. One can also view the *pij* as controller parameters to be specified subject to the constraints given previously.

Interestingly, it turns out that the system (1) is asymptotically stable in the sense of Lyapunov for any set of gains *Ki*>0 and any set of *pij*≥0 with ∑*ni*=1*pij*=1. Specifically, we have the following theorem.

Theorem

Given the system described by (1) and (7) with *λi*=0 for *i*=1,…,*n* and initial conditions *xi*(0)≥0, then (*qi*(*t*),*q*neti(*t*))→0 as *t*→∞.

Proof

First note that the *q*neti are nonnegative since by (7), it follows that

(10)

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

Under the conditions of the theorem, (9) becomes

(11)

$$\frac{d}{dt}\sum_{i=1}^{n}(q_i(t) + q_{\mathrm{net_i}}(t)) = -\sum_{i=1}^{n}\frac{\mu_i(q_i)}{t_{p_i}}.$$

Let *V*(*t*)=Δ∑*ni*=1(*qi*(*t*)+*q*neti(*t*)) and, as the *qi*(*t*), *q*neti(*t*) are nonnegative, *V*(*t*)≥0 and is equal to zero if and only if *qi*(*t*)=*q*neti(*t*)=0 for every *i*. Further, as *μi*(*qi*(*t*))=1 for *qi*(*t*)>0 and *μi*(*qi*(*t*))=0 if only if *qi*(*t*)=0, it follows that *dV*/*dt*=−∑*ni*=1*μi*(*qi*(*t*))/*tpi*≤0. This then implies that

(12)

$$V(t) = V(0) - \int_{0}^{t}\sum_{i=1}^{n}\frac{\mu_i(q_i(t))}{t_{p_i}}dt \geq 0$$

is monotonically decreasing. As *V*(*t*) is bounded below, we have $V(t) \downarrow V_f \geq 0$, or

(13)

$$\lim_{t\to\infty}\int_{0}^{t}\sum_{i=1}^{n}\frac{\mu_i}{t_{p_i}}dt = V(0) - V_f \geq 0\,.$$

The quantity $\mu_i\big(q_i(t)\big)$ is either 1 or 0 depending on whether $q_i(t)$ is positive or zero, so $\mu_i\big(q_i(t)\big)$can be viewed as a set of pulses of unit height and varying width. The integral $\int_0^\infty \mu_i(q_i(t))dt$ is finite by (13) which implies that the widths of the unit-height pulses making up *μi*(*qi*(*t*)) must go to zero as *t*→∞. So, even if a *qi*(*t*) (=*xi*(*t*)/*tpi*) continues to switch between zero and positive values, the time intervals for which it is nonzero must go to zero as *t*→∞. Summarizing, the *qi*(*t*) are nonnegative, continuous functions, bounded by the nonnegative monotonically decreasing function *V*(*t*), and the intervals for which the *qi*(*t*) are nonzero goes to zero as *t*→∞. More precisely, let $I_{t,h} = [t-h, t]$ and if we define $E_{t,h} = \{s \in I_{t,h}\colon q_i(s) > 0\}$, then the Lebesgue measure of *Et*,*h*, denoted by *m*(*Et*,*h*), converges to 0 as *t*→∞ for every *h*. Further, as *ui*(*t*)⩽ 0 is always true and

$$u_i(t) = -K_i \mathrm{sat}\left(x_i(t) - \frac{\sum_{j=1}^{n} x_j(t-\tau_{ij})}{n}\right)$$
$$= -K_i \mathrm{sat}\left(t_{p_i} q_i(t) - \frac{\sum_{j=1}^{n} (\frac{t_{p_i}}{t_{p_j}})\, q_j(t-\tau_{ij})}{n}\right)$$

it follows that the time intervals for which the *bounded* functions $u_i(t)$ are nonzero must go to zero as $t \to \infty$. This follows from the observation that $u_i(t) < 0$ necessarily implies $q_i(t) > 0$. Thus, the integral in (10) can be upper bounded by $\int_{[t-h_{ij},t]\cap E_{t,h_{ij}}} |K_i| y_{\max} d\tau = |K_i| y_{\max} m(E_{t,h_{ij}})$ (recall that $E_{t,h} \subset I_{t,h}$), which converges to zero as $t \to \infty$. Consequently, by (10), $q_{\mathrm{net_i}}(t) \to 0$ as $t \to \infty$.

We now show that the monotonically decreasing function $V(t)$ must go to zero, that is, $\lim_{t\to\infty} V(t) = V_f = 0$. Suppose not, so that *Vf*>0. As *q*neti(*t*)→0, choose *t*1 large enough so that 0≤∑*ni*=1*q*neti(*t*)<*ϵVf* for *t*>*t*1 where 0<*ϵ*<1. Since

$$V(t) = \sum_{i=1}^{n} (q_i(t) + q_{\mathrm{net_i}}(t)) \geq V_f \text{for all } t$$

we have

$$\sum_{i=1}^{n} q_i(t) \geq (1-\epsilon)V_f > 0 \text{for} t > t_1.$$

For every *t*>*t*1, there exists at least one *i* (which depends on *t*) for which *qi*(*t*)>0. Therefore, $\sum_{i=1}^{n} \mu_i(q_i(t))/t_{p_i} dt \geq min\{1/t_{p_i}\}$ for all *t*>*t*1. By (11), we then have

(14)

$$V(t) = V(t_1) - \int_{t_1}^{t} \sum_{i=1}^{n} \frac{\mu_i(q_i(t))}{t_{p_i}} dt \leq V(t_1) - \int_{t_1}^{t} \min\left\{\frac{1}{t_{p_i}}\right\} dt\,.$$

As the right side of (14) eventually becomes negative, we have a contradiction and therefore $V_f = 0$. As it has already been shown that $q_{\mathrm{net_i}} \to 0$ for all *i*, $V(t) \to 0$ then implies that $q_{\mathrm{net_i}} \to 0$ for all *i*. This completes the proof of the theorem.

SECTION V. Simulation Results

Experimental procedures to determine the delay values are given in [15] and summarized in [16]. These give representative values for a Fast Ethernet network with three nodes of $\tau_{ij}=\tau=200\ \mu s$ for $i\neq j$, $\tau_{ii}=0$, and $h_{ij}=2\tau=400\ \mu s$ for $i\neq j$, $h_{ii}=0$. The initial conditions were $x_1(0)=0.6$, $x_2(0)=0.4$ and $x_3(0)=0.2$. The inputs were set as $\lambda_1=3\mu_1$, $\lambda_2=0$, $\lambda_3=0$, $\mu_1=\mu_2=\mu_3=1$. The t_{pi}'s were taken to be equal to $10\ \mu s$.

In this set of simulations, the model (1) is used. Figs. 2 and 3 show the responses with the gains set as $K_1=K_2=K_3=K=1000$ and as $K_1=6667$, $K_2=4167$, $K_3=5000$, respectively.

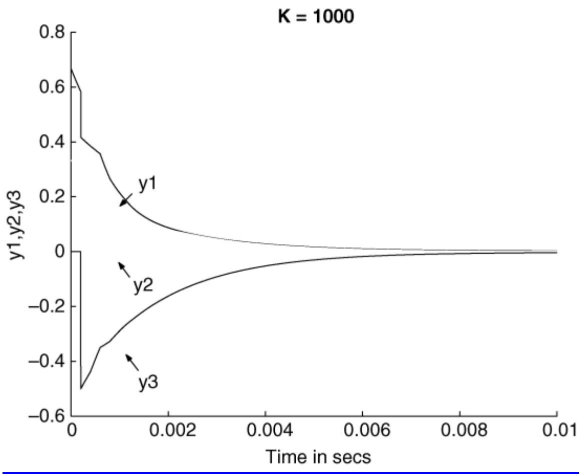


Fig. 2. Constant p_{ij} nonlinear output responses with $K=1000$.

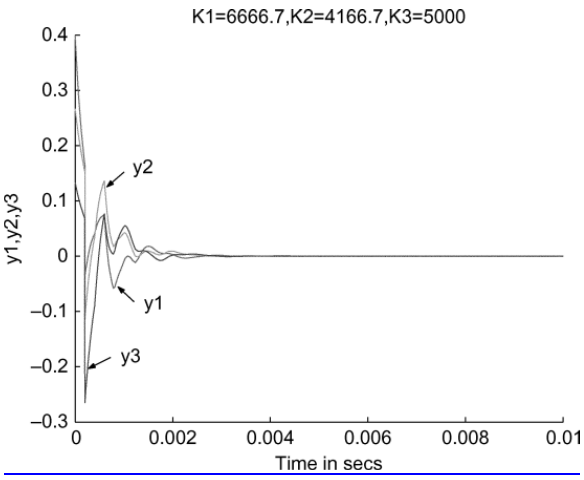


Fig. 3. Nonlinear simulation with constant p_{ij} and $K_1=6666.7$; $K_2=4166.7$; $K_3=5000$.

SECTION VI. Experimental Results

A parallel machine has been built to implement an experimental facility for evaluation of load balancing strategies and parallel databases. 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 (or DNA profile in our specific case) 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 1.4

GHz Athlon MP processors, and the single processor machines use 1.3 GHz Athlon processors. All run the Linux operating system. Our interest here is in the load balancing in any one group of *n* nodes.

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 requests are created not only by the database clients; the search process 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.

An important point is that the actual delays experienced by the network traffic in the parallel machine are *random*. Work has been performed to characterize the bandwidth and delay on unloaded and loaded network switches, in order to identify the delay parameters of the analytic models and is reported in [15] and [16]. The value τ=200 μs used for simulations represents an average value for the delay and was found using the procedure described in [16]. The interest here is to compare the experimental data with that from the three models previously developed.

To explain the connection between the control gain *Ki* and the actual implementation, recall that the waiting time is related to the number of tasks by $x_i(t) = q_i(t)t_{p_i}$ where t_{p_i} is the average time to carry out a task. The continuous time control law is

$$u_i(t) = -K_i \mathrm{sat}\big(y_i(t)\big)$$

where $u_i(t)$ is the rate of decrease of waiting time *xi*(*t*) per unit time. Consequently, the gain *Ki* represents the rate of reduction of waiting time per second in the continuous time model. Also, $y_i(t) = (q_i(t) - n^{-1}\sum_{j=1}^{n} q_j(t-\tau_{ij}))t_{p_i} = r_i(t)t_{p_i}$, where $r_i(t)$ is simply the number of tasks above the estimated (local) average number of tasks and, as the interest here is the case $y_i(t) > 0$, consider $u_i(t) = -K_i y_i(t)$. The implementation must execute the load balancing control law repetitively with a (possibly random) time interval between balancing actions. With Δt the time interval between successive executions of the load balancing algorithm, a discrete time control law is defined that removes a fraction of the queue $K_z r_i(t)(0 < K_z < 1)$ in the time Δt. The rate of reduction of *waiting time* is $-K_z r_i(t)t_{p_i}/\Delta t = -K_z y_i(t)/\Delta t$ so that an equivalent continuous time control law, given the discrete time gain K_z and control interval Δt, is

$$u(t) = -\frac{K_z y_i(t)}{\Delta t} \Longrightarrow K_i = \frac{K_z}{\Delta t}. \tag{15}$$

This shows that the gain *Ki* is related to the actual implementation by how fast the load balancing can be carried out and how much (fraction) of the load is transferred. In the experimental work reported here, Δt actually varies each time the load is balanced. As a consequence, the value of Δ*t* used in (15) is an average value for that run. The average time t_{p_i} to process a task is the same on all nodes used for the experiments (identical processors) and is equal 10 μs, while the time it takes to ready a load for transfer is about 5 μs. The initial conditions were taken as *q*1(0)=6000, *q*2(0)=4000, *q*3(0)=2000 (corresponding to $x_1(0) = q_1(0)t_{pi} = 0.06$, *x*2(0)=0.04, *x*3(0)=0.02). All of the experimental responses were carried out with constant $p_{ij} = 1/2$ for *i*≠*j*.

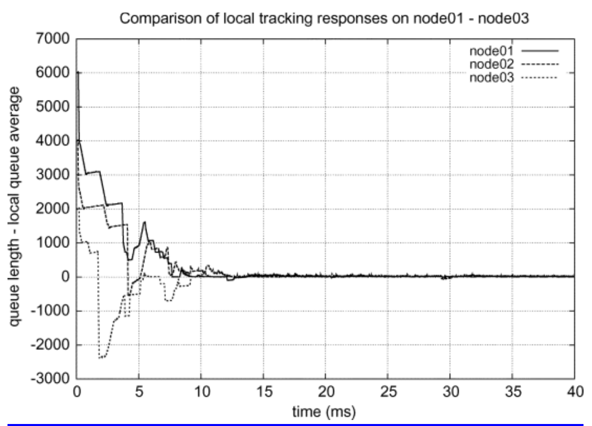


Fig. 4. Experimental response of the load balancing algorithm. The average value of the gains are (*Kz*=0.5) *K*1=6667, *K*2=4167, *K*3=5000 with constant p_{ij}.

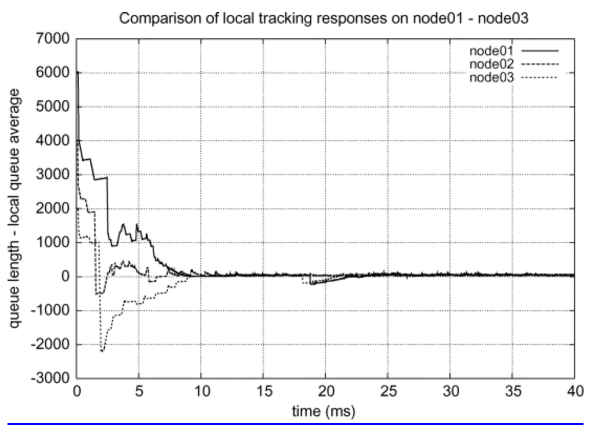


Fig. 5. Experimental response of the load balancing algorithm. The average value of the gains are (*Kz*=0.3) *K*1=2400, *K*2=7273, *K*3=2500 with constant p_{ij}.

Fig. 4 is a plot of the responses $r_i(t) = q_i(t) - (\sum_{j=1}^{n} q_j(t - \tau_{ij}))/n$ for $i = 1$, 2,3 (recall that *yi*(*t*)=*ri*(*t*)*tpi*). The (average) value of the gains were (*Kz*=0.5) *K*1=0.5/75 μs=6667, *K*2=0.5/120 μs=4167, *K*3=0.5/100 μs=5000. This figure compares favorably with Fig. 3 except for the time scale being off; that is, the experimental responses are slower. The explanation for this is that the discrete load balancing implementation is *not* accurately modeled in the continuous time simulations, only its average effect is represented in the gains *Ki*. That is, the continuous time model does not stop processing jobs (at the average rate *tpi*) while it is transferring tasks to do the load balancing. Fig. 5 shows the plots of the response for the (average) value of the gains given by (*Kz*=0.3) *K*1=0.3/125 μs=2400, *K*2=0.3/110 μs=7273, *K*3=0.3/120 μs=2500.

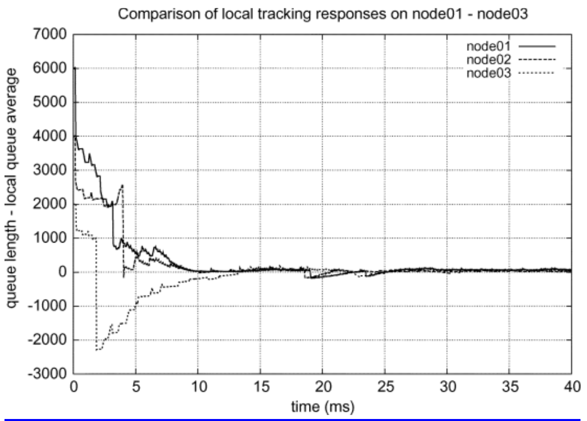


Fig. 6. Experimental response of the load balancing algorithm. The average value of the gains are (*Kz*=0.2) *K*1=1600, *K*2=2500, *K*3=2857 with constant p_{ij}.

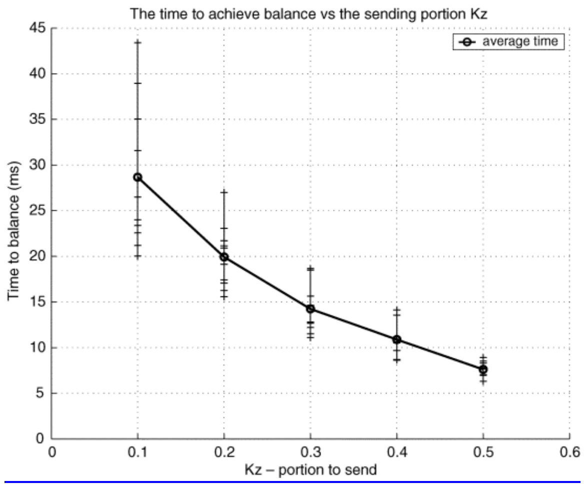


Fig. 7. Summary of the load balance time as a function of the feedback gain *Kz*.

Fig. 6 shows the plots of the response for the (average) value of the gains given by (*Kz*=0.2) *K*1=0.2/125 μs=1600, *K*2=0.2/80 μs=2500, *K*3=0.2/70 μs=2857. The initial conditions were *q*1(0)=6000, *q*2(0)=4000, *q*3(0)=2000 (*x*1(0)=*q*1(0)*tpi*=0.06, *x*2(0)=0.04, *x*3(0)=0.02).

Fig. 7 summarizes the data from several experimental runs of the type shown in Figs. 4 –6. For *Kz*=0.1, 0.2,0.3,0.4,0.5, ten runs were made and the settling time (time to load balance) were determined. These are marked as small horizontal ticks on Fig. 7. (For all such runs, the initial queues were the same and equal to *q*1(0)=600, *q*2(0)=400, *q*3(0)=200. For each value of *Kz*, the average settling time for these ten runs was computed and is marked as a dot on given on Fig. 7. For values of *Kz*=0.6 and higher (with increments of 0.1 in *Kz*), consistent results could not be obtained. In many cases, ringing extended throughout the experiment's time interval (200 ms).

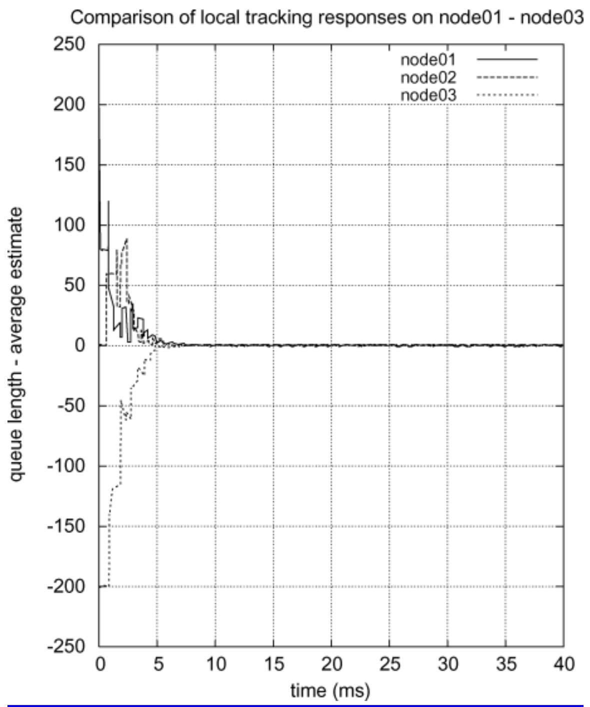


Fig. 8. *Kz*=0.6—settling time is approximately 7 ms.

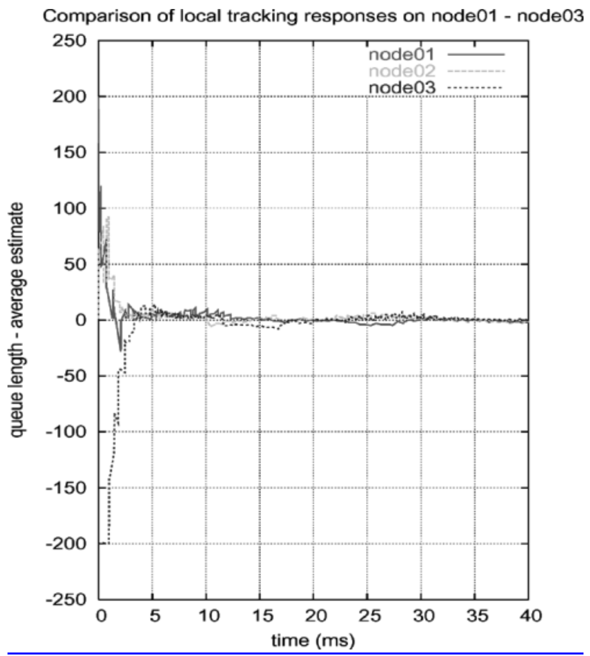
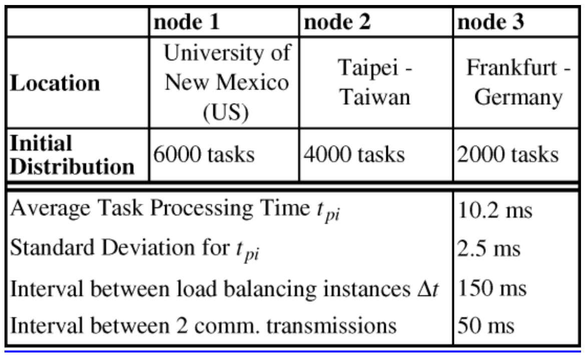


Fig. 9. *Kz*=0.6 These are the same conditions as Fig. 8, but now the ringing persists.

For example, Fig. 8 shows the plots of the queue length less the local queue average for an experimental run with *Kz*=0.6 where the settling time is approximately 7 ms. In contrast, Fig. 9 shows the experimental results under the same conditions where persistent ringing regenerates for 40 ms. It was found the response was so oscillatory that a settling time was not possible to determine accurately. However, Fig. 7 shows that one desires to choose the gain to be close to 0.5 to achieve a faster response time without breaking into oscillatory behavior.

SECTION VII. Experiments Over PlanetLab

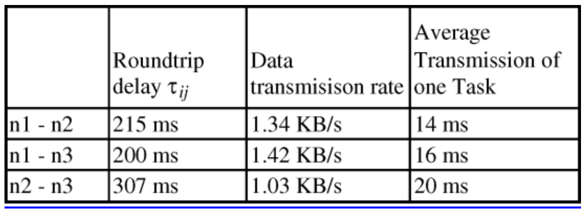
A geographically distributed system was developed to validate the theoretical work for large delays and to assess different load balancing policies in a real environment. The system consists of several nodes running the same code. The nodes are part of PlanetLab, a planetary-scale network involving more than 350 nodes positioned around the globe



	node 1	**node 2**	**node 3**
Location	University of New Mexico (US)	Taipei - Taiwan	Frankfurt - Germany
Initial Distribution	6000 tasks	4000 tasks	2000 tasks
Average Task Processing Time t_{pi}			10.2 ms
Standard Deviation for t_{pi}			2.5 ms
Interval between load balancing instances Δt			150 ms
Interval between 2 comm. transmissions			50 ms

	Node 1	Node 2	Node 3
Location	University of New Mexico (US)	Taipei - Taiwan	Frankfurt - Germany
Initial Distribution	6000 tasks	4000 tasks	2000 tasks
Average Task Process Time t_{pi}			10.2 ms
Standard Deviation for t_{pi}			2.5 ms
Interval between load balancing instances Δt			150 ms
Interval between 2 comm. transmissions			50 ms

Fig. 10. Parameters and settings of the experiment.



	Roundtrip delay τ_{ij}	Data transmisison rate	Average Transmission of one Task
n1 - n2	215 ms	1.34 KB/s	14 ms
n1 - n3	200 ms	1.42 KB/s	16 ms
n2 - n3	307 ms	1.03 KB/s	20 ms

	Roundtrip delay	Data transmission rate	Average Transmission of one Task
n1 – n2	215 ms	1.34 KB/s	14 ms
n1 – n3	200 ms	1.42 KB/s	16 ms
n2 – n3	307 ms	1.03 KB/s	20 ms

Fig. 11. Average network delays and transmission rates.

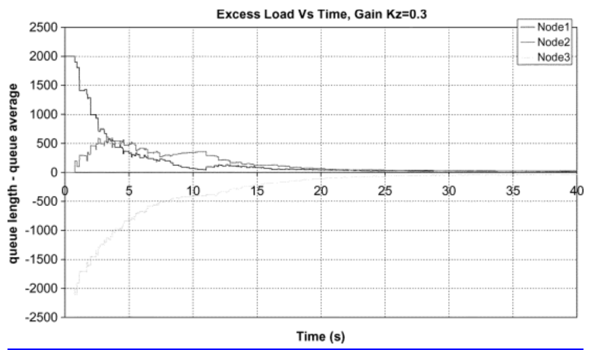


Fig. 12. Experimental response of the load balancing algorithm under large delays. Gain *Kz*=0.3 and *pij*=0.5.

and connected via the Internet (www.planetlab.org). The application used to illustrate the load balancing process was matrix multiplication, where one task is defined as the multiplication of one row by a static matrix duplicated on all nodes (3 nodes in our experiment). The number of elements in each row were generated randomly from a specified range, which made the execution time of a task variable. The network protocol UDP was used to exchange queue size information among the nodes, and TCP (connection-based) was used to transfer task data from one machine to another.

To match the experimental settings of the previous sections, 3 nodes were used; node1 at the University of New Mexico, node2 in Taipei, Taiwan and node3 in Frankfurt, Germany. The *pij* were set to 1/2 for *i≠j*. The initial parameters and settings for the experiment are summarized in the Table given in Fig. 10.

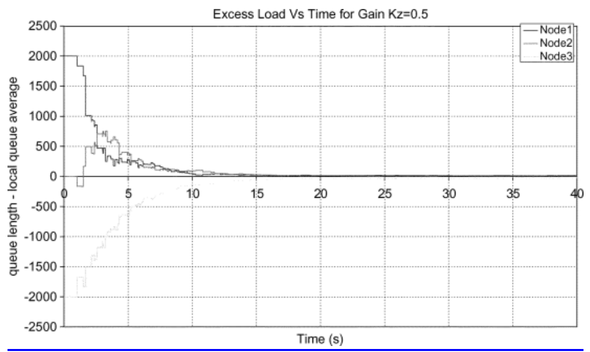


Fig. 13. Experimental response of the load balancing algorithm under large delays. Gain *Kz*=0.5 and *pij*=0.5.

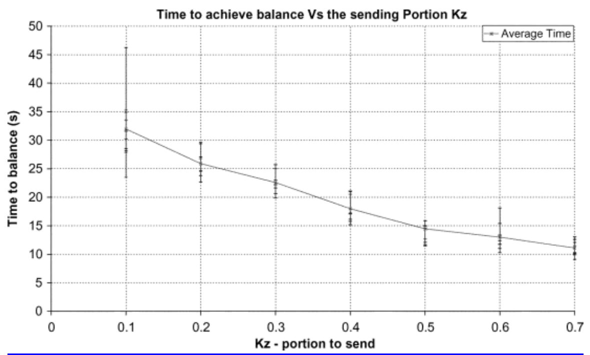


Fig. 14. Summary of the load balancing time as function of the gain *Kz*.

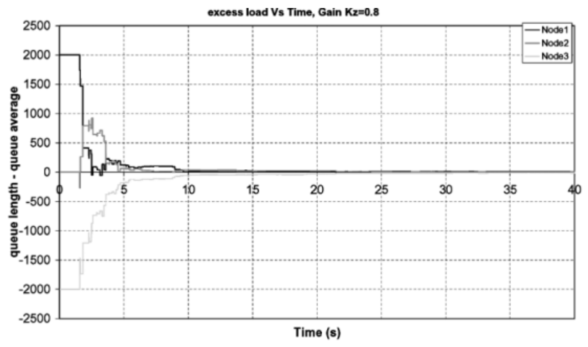


Fig. 15. Experimental response of the load balancing algorithm under large delays. Gain *Kz*=0.8 and *pij*=0.5.

Throughout the experiments, network statistics related to transmission rates and delays were collected. The averages of these statistics are shown in the Table given in Fig. 11. Large delays were observed in the network due to the dispersed geographical location of the nodes. Moreover, the transmission rates detected between the nodes were very low mainly because the amount of data exchanged (in bytes) is small. Indeed, the average size of data needed to transmit a single task was 20 bytes, which caused variability in the observed transmission rates due to the large communication delays and their variation. In order to observe the behavior of the system under various gains, several experiments were conducted for different gain values *Kz* ranging from 0.1 to 1.0. Fig. 12 is a plot of the responses *ri*(*t*) corresponding to each node *i* where the gain *Kz* was set to 0.3.

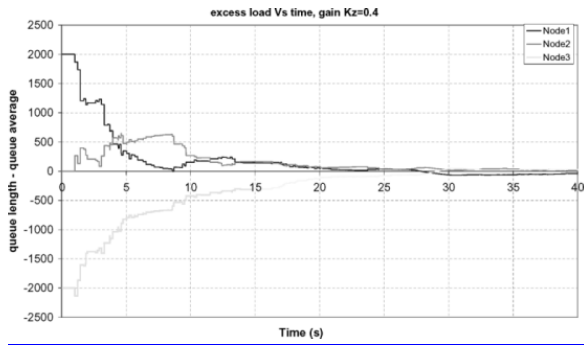


Fig. 16. Experimental response of the load balancing algorithm under large delays. Gain *Kz*=0.4 and *pij*=0.5.

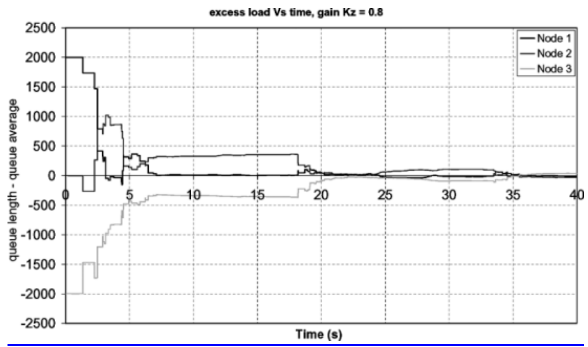


Fig. 17. Experimental response of the load balancing algorithm under large delays. Gain *Kz*=0.8 and *pij*=0.5.

Similarly, Fig. 13 shows the system response for gain *Kz* equal to 0.5. Fig. 14 summarizes several runs corresponding to different gain values. For each *Kz*=0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, ten runs were made, and the settling times (time to load balance) were determined. For gain values higher than 0.8, consistent results could not be obtained. For instance, in most of the runs no settling time could be achieved. However, when the observed network delays were stable, the system response was steady and converged quickly to a balanced state when *Kz* was equal to 0.8 (Fig. 15). As previously mentioned, this scenario was not frequently seen. The system behaviors in these experiments do not exactly match, and the results obtained in the previous sections. This is due to the difference in network topology and delays (and very likely due to the random nature of the delay [17], [18]). For instance, the ratio between the average delay and the task process time is 20 (200 μs/10 μs) for the local area network (LAN) setting and 12 (120 ms/10 ms) for the distributed setting. This fact is one of the reasons ringing is observed earlier (for *Kz*=0.6) in the LAN experiment whereas under PlanetLab the ringing responses were observed starting at *Kz*=0.8. Interestingly, we have previously observed a similar behavior, using a Monte Carlo simulation of a 3-node distributed system with *random* delays in [17] as well as experiments on a wireless LAN in [18].

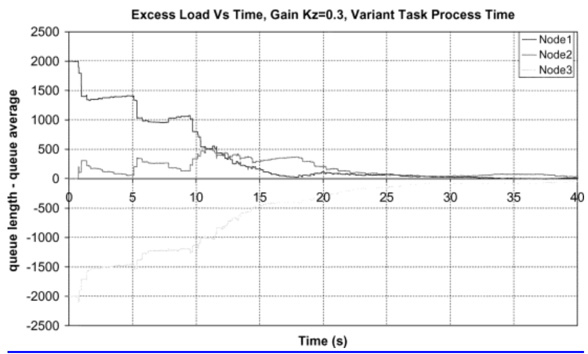


Fig. 18. Experimental response of the load balancing algorithm under large variance in the tasks processing time. Gain *Kz*=0.3 and *pij*=0.5.

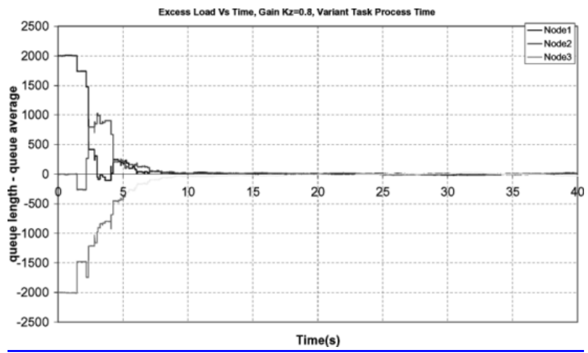


Fig. 19. Experimental response of the load balancing algorithm under large variance in the tasks processing time. Gain *Kz*=0.8 and *pij*=0.5.

The previous experiments have been conducted under normal network conditions stated in the Table given in Fig. 11. However, another set of experiments were carried out where the network condition worsens and larger delays were observed. In particular, the data transmission rate between node 2 (Taiwan) and node 3 (Germany) dropped from 1.03 KB/s to 407 KB/s. Figs. 16 and 17 show the system responses for gains *Kz*=0.4 and *Kz*=0.8, respectively. These experiments clearly show the negative effect of the delay on the stability of the system. Nevertheless, we see that with a low gain (*Kz*=0.4), the settling time is approximately 22 ms. On the other hand, when the gain was set to 0.8, the system did not reach an equilibrium as shown by the nodes' responses *ri*(*t*) in Fig. 17. At this point, only the effect of delay on the stability of the system was tested. In order to study the effect of the variability of the task processing time on system behavior, the matrix multiplication application was adjusted in a way to obtain the following results; the average task processing time was kept at 10.2 ms, but the standard deviation became 7.15 ms instead of 2.5 ms. Figs. 18 and 19 show the respective system responses for gains *Kz*=0.3 and *Kz*=0.8. Comparing Figs. 12 and 18, we can see that in the latter case, some ringing persists and the system response did not completely settle out. On the other hand, setting the gain *Kz* to 0.8 led the system to accommodate the variances in the task processing time.

The experiments presented in this section support the ones reported in the previous section where a local area network was used. In particular, high gains were shown to lead to persistent ringing. Conversely, systems with low gain values lead to slow responses in the load balancing.

SECTION VIII. Summary and Conclusion

A load balancing algorithm was modeled as a nonlinear time-delay system. The model was shown to be consistent in that the total number of tasks was conserved and the queues were always nonnegative. Further, the system was shown to be always stable, but the delays do create a limit on the size of the controller gains in order to ensure performance (fast enough response without oscillatory behavior). Experiments demonstrated a correlation of the continuous time model with the actual implementation. Future work will consider the fact that the load balancing operation involves processor time which is not being used to process tasks. There is a tradeoff between using processor time/network bandwidth and the advantage of distributing the load evenly between the nodes to reduce overall processing time, which has not been fully captured in the present work.

References

1. E. Altman, H. Kameda, "Equilibria for multiclass routing in multi-agent networks", *Proc. 40th IEEE Conf. Decision and Control*, pp. 604-609, 2001-Dec.

2. C. K. Hisao Kameda, J. Li, Y. Zhang, Optimal Load Balancing in Distributed Computer Systems, New York:Springer-Verlag, 1997.

3. H. Kameda, I. R. El-Zoghdy Said Fathy, J. Li, "A performance comparison of dynamic versus static load balancing policies in a mainframe", *Proc. 39th IEEE Conf. Decision and Control*, pp. 1415-1420, 2000-Dec.

4. J. D. Birdwell, R. D. Horn, D. J. Icove, T. W. Wang, P. Yadav, S. Niezgoda, "A hierarchical database design and search method for CODIS", *Proc. 10th Int. Symp. Human Identification*, 1999-Sep.

5. J. D. Birdwell, T. W. Wang, R. D. Horn, P. Yadav, D. J. Icove, "Method of indexed storage and retrieval of multidimensional information", *Proc. 10th SIAM Conf. Parallel Processing for Scientific Computation*, 2000-Sep.

6. J. D. Birdwell, T.-W. Wang, M. Rader, "The university of Tennessee's new search engine for CODIS", *Proc. 6th CODIS Users Conf.*, 2001-Feb.

7. T. W. Wang, J. D. Birdwell, P. Yadav, D. J. Icove, S. Niezgoda, S. Jones, "Natural clustering of DNA/STR profiles", *Proc. 10th Int. Symp. Human Identification*, 1999-Sep.

8. H. G. Rotithor, "Taxonomy of dynamic task scheduling schemes in distributed computing systems", *Inst. Elect. Eng. Proc. Comput. Dig. Techniques*, vol. 141, no. 1, pp. 1-10, 1994.

9. A. Corradi, L. Leonardi, F. Zambonelli, "Diffusive load-balancing polices for dynamic applications", *IEEE Concurrency*, vol. 22, no. 1, pp. 979-993, Jan.-Feb. 1999.

10. M. H. Willebeek-LeMair, A. P. Reeves, "Strategies for dynamic load balancing on highly parallel computers", *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 9, pp. 979-993, Sep. 1993.

11. C. T. Abdallah, J. D. Birdwell, J. Chiasson, V. Chupryna, Z. Tang, T. W. Wang, "Load balancing instabilities due to time delays in parallel computation", *Proc. 3rd IFAC Conf. Time Delay Systems*, pp. 198-202, 2001-Dec.

12. J. D. Birdwell, J. Chiasson, Z. Tang, C. T. Abdallah, M. Hayat, T. Wang, "Dynamic time delay models for load balancing part I: deterministic models", *Proc. CNRS-NSF Workshop: Advances in Control of Time-Delay Systems*, pp. 355-370, 2003-Jan.

13. J. D. Birdwell, J. Chiasson, Z. Tang, C. T. Abdallah, M. M. Hayat, "The effect of feedback gains on the performance of a load balancing network with time delays", *Proc. IFAC Workshop on Time-Delay Systems (TDS'03)*, pp. 371-385, 2003-Sep.

14. J. D. Birdwell, J. Chiasson, C. T. Abdallah, Z. Tang, N. Alluri, T. Wang, "The effect of time delays in the stability of load balancing algorithms for parallel computations", *Proc. 42nd IEEE Conf. Decision and Control*, pp. 582-587, 2003-Dec.

15. P. Dasgupta, *Performance evaluation of fast ethernet ATM and myrinet under PVM*, 2001.

16. P. Dasgupta, J. D. Birdwell, T. W. Wang, "Timing and congestion studies under PVM", *Proc. 10th SIAM Conf. Parallel Processing for Scientific Computation*, 2001-Mar.

17. S. Dhakal, B. Paskaleva, M. Hayat, E. Schamiloglu, C. Abdallah, "Dynamical discrete-time load balancing in distributed systems in the presence of time delays", *Proc. IEEE Conf. Decision Control*, pp. 5128-5134, 2003-Dec.

18. J. Ghanem, S. Dhakal, M. M. Hayat, H. Jerez, C. T. Abdallah, J. Chiasson, "On load balancing in distributed systems with large time delays: theory and experiments", *Proc. IEEE Mediterranean Control Conf. (MED'04)*, 2004-Jun.-6�shed'.