Implementation of Load Balancing Algorithms over a Local Area Network and the Internet

J. Ghanem, C. T. Abdallah, M. Hayat, S. Dhakal, J.D. Birdwell, J. Chiasson, and Z. Tang*

Electrical and Computer Engineering Department
MSC01 1100
University of New Mexico
Albuquerque, NM 87131-0001
{jean,chaouki,hayat,dhakal}@eece.unm.edu

*Department of Electrical and Computer Engineering
University of Tennessee
Knoxville, TN 37996-2100 USA
{birdwell,chiasson,ztang}@utk.edu

***Abstract*— In this paper, experimental evaluation of a control-theoretic based load balancing algorithm in real environments is presented. we emphasize on the effects of delays in the exchange of information among nodes, and the constraints these effects impose on the design of a load balancing strategy. two test-beds in two different real environments have been built; The first implementation was over a local area network whereas the second one was over Planet-Lab. The results show the effect of the network delays and variances in the tasks processing time on choosing adequate gain values for the load balancing algorithm.**

I. INTRODUCTION

Parallel computer architectures utilize a set of computational elements (CE) to achieve performance that is not attainable on a single processor, or CE, computer. A common architecture is the cluster of otherwise independent computers communicating through a shared network. To make use of parallel computing resources, problems must be broken down into smaller units that can be solved individually by each CE while exchanging information with CEs solving other problems. For a background on mathematical treatments of load balancing, the reader is referred to [1][2][3]. Effective utilization of a parallel computer architecture requires the computational load to be distributed more or less evenly (accounting for computational and bandwidth limitations) over the available CEs.

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 [4] and gradient methods, stochastic, or optimization based. A comparison of several deterministic methods is provided by Willeback-LeMain and Reeves [5].

The present work focuses on the experimental aspects of a load balancing in real environments, highlighting 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. The load balancing algorithm was inspired by control theoretic results of time-delay systems. Two test-beds were implemented in different environments; The first test-bed involved three machines connected by a switch in local area network (LAN) setting and the second test-bed involved three nodes geographically distributed and connected by the Internet. In the latter setting, the nodes used are part of the Planet-Lab research network. More information about Planet-Lab can be found at www.planet-lab.org.

Section 2 presents our approach to modelling the computer network and load balancing algorithms to incorporate the presence of delay in communicating between nodes and transferring tasks ([6], [7],[8]). Section 3 presents *experimental data* of the load balancing algorithm implemented over a local area network (LAN). Section 4 presents experiments conducted over a distributed environment (Planet-Lab). Both the effect of the network delays and the variances in the tasks processing time on the behavior of the system are assessed. Finally, Section 5 is a summary and conclusion of the present work and a discussion of future work.

II. MATHEMATICAL MODEL

In this section, continuous time models are developed to model load balancing among a network of computers.

This work is supported by the National Science Foundation under Information Technology Research (ITR) grant No. ANI-0312611. Additional support was received from the National Science grant No. IIS-023205. Foundation through grant No. .

To introduce the basic approach to load balancing, consider a computing network consisting of n computers (nodes) all of which can communicate with each other. At start up, the computers are assigned an equal number of tasks. However, 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 $q_j(t)$ to all other computers in the network. A node i receives this information from node j *delayed* by a finite amount of time τ_{ij}; that is, it receives $q_j(t-\tau_{ij})$. Each node i then uses this information to compute its local estimate[1] of the average number of tasks in the queues of the n computers in the network. Based on the most recent observations, the simple estimator $\left(\sum_{j=1}^n q_j(t-\tau_{ij})\right)/n$ $(\tau_{ii}=0)$ of the network average is used by the i^{th} node. Node i then compares its queue size $q_i(t)$ with its estimate of the network average as $\left(q_i(t)-\left(\sum_{j=1}^n q_j(t-\tau_{ij})\right)/n\right)$ and, if this is greater than zero, the node sends some of its tasks to the other nodes. If it is less than zero, no tasks are sent. Further, the tasks sent by node i are received by node j with a delay h_{ij}. The controller (load balancing algorithm) decides how often 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 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 continuous time models developed here represent our effort to capture the effect of the delays in load balancing techniques and were developed so that system theoretic methods could be used to analyze them. The basic mathematical model of a given computing node for load balancing is given by

$$\begin{aligned}\frac{dx_i(t)}{dt} &= \lambda_i-\mu_i+u_i(t)-\sum_{j=1}^n p_{ij}\frac{t_{p_i}}{t_{p_j}}u_j(t-h_{ij})\\ y_i(t) &= x_i(t)-\frac{\sum_{j=1}^n x_j(t-\tau_{ij})}{n}\\ u_i(t) &= -K_i\mathrm{sat}\left(y_i(t)\right)\\ p_{ij} &\geqslant 0, p_{jj}=0, \sum_{i=1}^n p_{ij}=1\end{aligned} \tag{1}$$

where

$$\begin{aligned}\mathrm{sat}(y) &= y \text{ if } y\geqslant 0\\ &= 0 \text{ if } y<0.\end{aligned}$$

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 $q_jt_{p_i}=x_jt_{p_i}/t_{p_j}$, 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.
- $u_i(t)$ 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 $u_i(t)\leq 0$.
- $p_{ij}u_j(t)$ 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) $\int_{t_1}^{t_2}u_j(t)dt$ in the interval of time $[t_1,t_2]$ to the other nodes is carried out with the i^{th} node receiving the fraction $p_{ij}\left(t_{p_i}/t_{p_j}\right)\int_{t_1}^{t_2}u_j(t)dt$ where the ratio t_{p_i}/t_{p_j} converts the task from waiting time on node j to waiting time on node i. As $\sum_{i=1}^n\left(p_{ij}\int_{t_1}^{t_2}u_j(t)dt\right)=\int_{t_1}^{t_2}u_j(t)dt$, this results in removing *all* of the waiting time $\int_{t_1}^{t_2}u_j(t)dt$ from node j.
- The quantity $-p_{ij}u_j(t-h_{ij})$ 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 h_{ij} $(h_{ii}=0)$ is the time delay for the task transfer from node j to node i.
- The quantities τ_{ij} $(\tau_{ii}=0)$ denote the time delay for communicating the expected waiting time x_j from node j to node i.
- The quantity $x_i^{avg}=\left(\sum_{j=1}^n x_j(t-\tau_{ij})\right)/n$ is the estimate (due to the delays) by the i^{th} node of the average waiting time of the network and is referred to as the *local average* (local estimate of the average).

In this model, all rates are in units of the *rate of change of expected waiting time*, or *time/time* which is dimensionless. As $u_i(t)\leq 0$, 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 $u_i(t)=-K_i\mathrm{sat}(y_i(t))$ states that if the i^{th} node output $x_i(t)$ is above the local average $\left(\sum_{j=1}^n x_j(t-\tau_{ij})\right)/n$, then it sends data to the other nodes, while if it is less than the local average nothing is sent. The j^{th} node receives the fraction $\int_{t_1}^{t_2}p_{ji}\left(t_{p_i}/t_{p_j}\right)u_i(t)dt$ of transferred waiting time $\int_{t_1}^{t_2}u_i(t)dt$

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

delayed by the time h_{ij}.

A. Specifying the p_{ij}

Model (1) is the basic model, but one important detail remains unspecified, namely the exact form of the p_{ij} for each sending node i. One approach is to choose them as constant and equal

$$\begin{aligned} p_{ij} &= 1/(n-1) \text{ for } j \neq i \\ p_{ii} &= 0 \end{aligned} \tag{2}$$

where it is clear that $p_{ji} \geqslant 0, \sum_{i=1}^{n} p_{ij} = 1$.

Another approach is to use the local information of the waiting times $x_i(t), i = 1, .., n$ to set their values. Recall that p_{ij} is the fraction of $u_j(t)$ that node j allocates (transfers) to node i at time t, and conservation of the tasks requires $p_{ij} \geqslant 0, \sum_{i=1}^{n} p_{ij} = 1$ and $p_{jj} = 0$. The quantity $x_i(t-\tau_{ji}) - x_j^{avg}$ represents what node j estimates the waiting time in the queue of node i is with respect to the local average of node j. If queue of node i is above the local average, then node j does not send tasks to it. Therefore $\text{sat}\left(x_j^{avg} - x_i(t-\tau_{ji})\right)$ is an appropriate measure by node j as to how much node i is *below* the local average. Node j then repeats this computation for all the other nodes and then portions out its tasks among the other nodes according to the amounts they are below the local average, that is,

$$p_{ij} \triangleq \frac{\text{sat}\left(x_j^{avg} - x_i(t-\tau_{ji})\right)}{\sum\limits_{i \ni i \neq j} \text{sat}\left(x_j^{avg} - x_i(t-\tau_{ji})\right)}. \tag{3}$$

If the denominator $\sum\limits_{i \ni i \neq j} \text{sat}\left(x_j^{avg} - x_i(t-\tau_{ji})\right) = 0$, then the p_{ij} are defined to be zero and no load is transferred.

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,

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

which implies

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

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

III. Experiments conducted over a local area network

A parallel machine has been built to implement an experimental facility for evaluation of load balancing strategies. A root node communicates with k groups of computer networks. Each of these groups is composed of n nodes (hosts) holding identical copies of a portion of the database. (Any pair of groups correspond to different databases, which are not necessarily disjoint. A specific record, or DNA profile, 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 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 [9][10]. The value $\tau = 200\ \mu\text{sec}$ used for simulations represents an average value for the delay and was found using the procedure described in [10]. 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 K_i and the actual implementation, recall that the waiting time is related to the number of tasks as $x_i(t) = q_i(t)t_{p_i}$ where t_{p_i} is the average time to carry out a task. The continuous time control law is

$$u_i(t) = -K\text{sat}(y_i(t))$$

where $u_i(t)$ is the rate of decrease of waiting time $x_i(t)$ per unit time. Consequently, the gain K_i represents the rate of reduction of waiting time per second in the continuous time model. Also, $y_i(t) = \left(q_i(t) - \left(\sum_{j=1}^{n} q_j(t-\tau_{ij})\right)/n\right) t_{p_i} = r_i(t)t_{p_i}$ 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 $u(t) = -K_i y_i(t)$. With Δt the time interval between successive executions of the load balancing algorithm, the control law says that a fraction of the queue $K_z r_i(t)$ $(0 < K_z < 1)$ is removed in the time Δt so 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

$$u(t) = -\frac{K_z y_i(t)}{\Delta t} \Longrightarrow K_i = \frac{K_z}{\Delta t}. \qquad (4)$$

This shows that the gain K_i 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 Δt used in (4) is an average value for that run. The average time t_{p_i} to process a task is the same on all nodes (identical processors) and is equal 10μ sec while the time it takes to ready a load for transfer is about 5μ sec. 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 $i \neq j$.

Figure 1 is a plot of the responses $r_i(t) = q_i(t) - \left(\sum_{j=1}^{n} q_j(t-\tau_{ij})\right)/n$ for $i = 1,2,3$ (recall that $y_i(t) = r_i(t)t_{p_i}$). The (average) value of the gains were ($K_z = 0.5$) $K_1 = 0.5/75\mu\,\text{sec} = 6667, K_2 = 0.5/120\mu\,\text{sec} = 4167, K_3 = 0.5/100\mu\,\text{sec} = 5000$. Figure 2 shows the plots of the

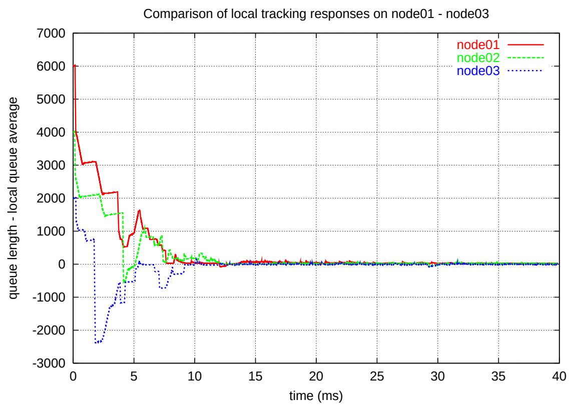

Fig. 1. Experimental response of the load balancing algorithm. The average value of the gains are ($K_z = 0.5$) $K_1 = 6667, K_2 = 4167, K_3 = 5000$ with constant p_{ij}.

response for the (average) value of the gains given by ($K_z = 0.3$) $K_1 = 0.3/125\mu\,\text{sec} = 2400, K_2 = 0.3/110\mu\,\text{sec} = 7273, K_3 = 0.3/120\mu\,\text{sec} = 2500$. Figure 3 shows the plots of the response for the (average) value of the gains given by ($K_z = 0.2$) $K_1 = 0.2/125\mu\,\text{sec} = 1600, K_2 = 0.2/80\mu\,\text{sec} = 2500, K_3 = 0.2/70\mu\,\text{sec} = 2857$. The initial conditions were $q_1(0) = 6000, q_2(0) = 4000, q_3(0) = 2000$ ($x_1(0) = q_1(0)t_{pi} = 0.06, x_2(0) = 0.04, x_3(0) = 0.02$).

Figure 4 summarizes the data from several experimental runs of the type shown in Figures 1, 3, 2. For $K_z = 0.1, 0.2, 0.3, 0.4, 0.5$, ten runs were made and the settling time (time to load balance) were determined. These are marked as small horizontal ticks on Figure 4. (For all such runs, the initial queues were the same and equal to

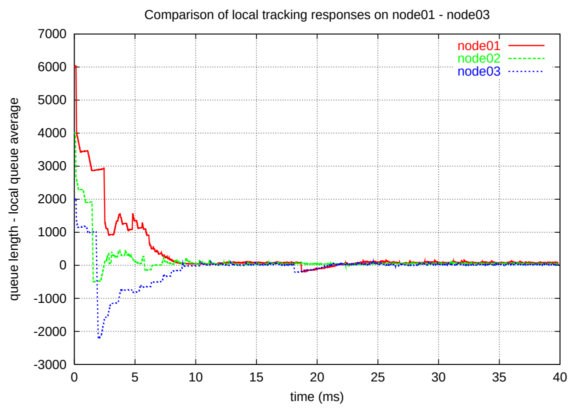

Fig. 2. Experimental response of the load balancing algorithm. The average value of the gains are ($K_z = 0.3$) $K_1 = 2400, K_2 = 7273, K_3 = 2500$ with constant p_{ij}.

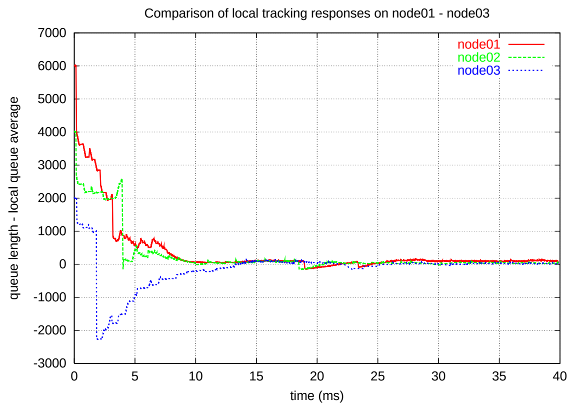

Fig. 3. Experimental response of the load balancing algorithm. The average value of the gains are ($K_z = 0.2$) $K_1 = 1600, K_2 = 2500, K_3 = 2857$ with constant p_{ij}.

$q_1(0) = 600, q_2(0) = 400, q_3(0) = 200$. For each value of K_z, the average settling time for these ten runs was computed and is marked as a dot on given on Figure 4. For values of $K_z = 0.6$ and higher (with increments of 0.1 in K_z), consistent results could not be obtained. In many cases, ringing extended throughout the experiment's time interval (200 miliseconds).

For example, Figure 5 shows the plots of the queue length less the local queue average for an experimental run with $K_z = 0.6$ where the settling time is approximately 7 milliseconds. In contrast, Figure 6 shows the experimental results under the same conditions where persistent ringing regenerates for 40 milliseconds. it was found the response was so oscillatory that a settling time was not possible to determine accurately. However, Figure 4 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.

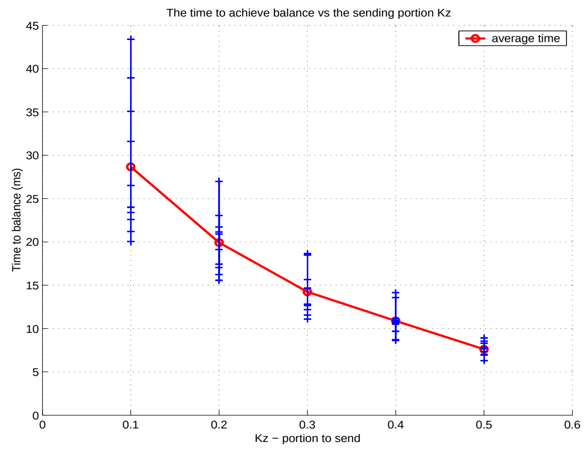

Fig. 4. Summary of the load balance time as a function of the feeback gain K_z

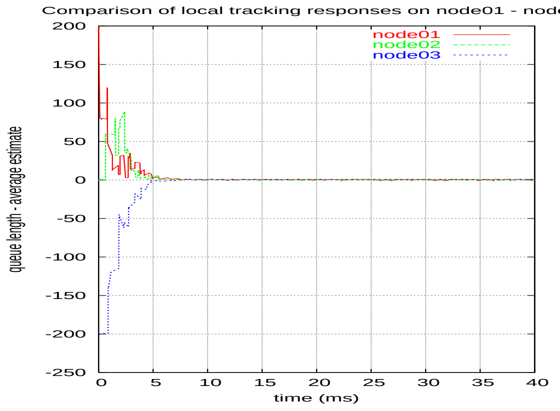

Fig. 5. $K_z = 0.6$ - Settling time is approximately 7 milliseconds.

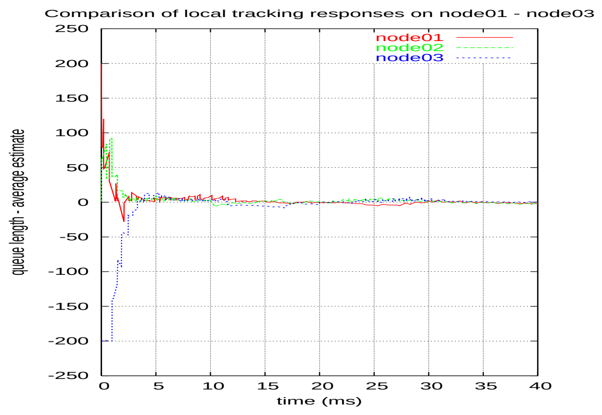

Fig. 6. $K_z = 0.6$ These are the same conditions as Figure 5, but now the ringing persists.

	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

TABLE I
PARAMETERS AND SETTINGS OF THE EXPERIMENT

IV. EXPERIMENTS OVER PLANET-LAB

Another distributed system has been developed to validate the theoretical work 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 Planet-Lab, a planetary-scale network involving more than 350 nodes positioned around the globe and connected via the Internet (www.palnetlab.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 size of the elements in each row was generated randomly from a specified range which made the execution time of a task variable. As for the communication part of the program, UDP was used to exchange queue size information among the nodes and TCP was used to transfer the data or tasks 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. As for the load balancing policy, the same parameter values were also used; for instance all p_{ij} were set to $1/2$ for $i \neq j$. The initial parameters and settings for the experiment are summarized in table I.

Throughout the experiment, network statistics related to transmission rates and delays were collected. The averages of the parameters are shown in table II. 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 made the observed transmission rates not exactly accurate in the presence of large communication delays.

In order to observe the behavior of the system under various gains, several experiments were conducted for different gain values K_z ranging from 0.1 to 1. Fig. 7 is a plot of the responses $r_i(t)$ corresponding to each node i where the gain K_z was set to 0.3. Similarly, Fig. 8 shows the system response for gain K_z equal to 0.5. Figure 9 summarizes several runs corresponding to different gain values. For each $K_z = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7$, ten runs were made and the settling times (time to load balance)

	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

TABLE II

AVERAGE NETWORK DELAYS AND TRANSMISSION RATES.

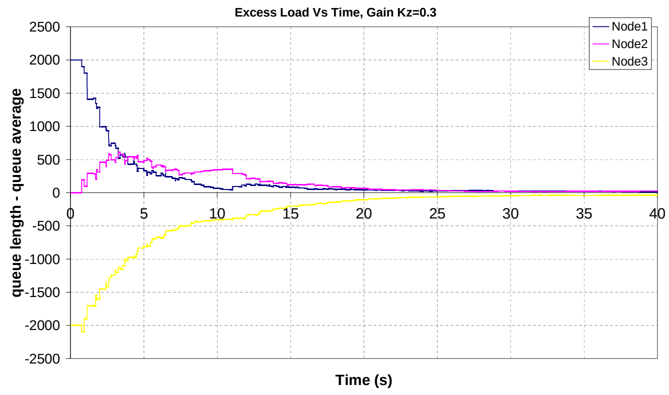

Fig. 7. Experimental response of the load balancing algorithm under large delays. gain $K_z = 0.3$ and $p_{ij} = 0.5$.

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 K_z is equal to 0.8 (Figure 10. As previously indicated, this scenario wasn't frequently seen. The system behavior in these set of experiments does not exactly match, for the same gain value, the results obtained in the previous sections, due to the difference in network topology and delays. For instance, the ratio between the average delay and the task process time is 20 $(200\mu s/10\mu s)$ for the LAN setting and 12 $(120ms/10ms)$ for the distributed setting. This fact is one of the reasons why ringing is observed earlier (for $K_z = 0.6$) in the LAN experiment whereas under Planet-Lab unstable responses were observed starting $K_z = 0.8$.

The previous experiments have been conducted under normal network conditions stated in Table II. However, another set of experiments was conducted at a different time where the network condition worsens and larger delays were observed. In particular, the data transmission rate between node 2 (Taiwan) and node 3 dropped from 1.03KB/s to 407B/s. Figures 11 and 12 show the system responses for gains $K_z = 0.4$ and $K_z = 0.8$ respectively. These experiments clearly show the negative effect of the delay on the stability of the system. Nevertheless, we can see that with a low gain namely $K_z = 0.4$, a settling time can be identified at around

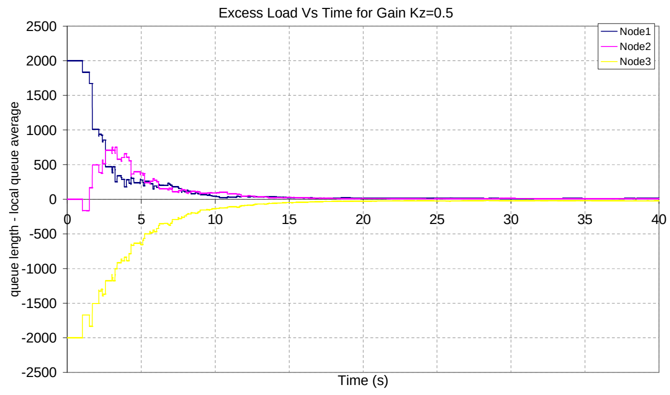

Fig. 8. Experimental response of the load balancing algorithm under large delays. gain $K_z = 0.5$ and $p_{ij} = 0.5$.

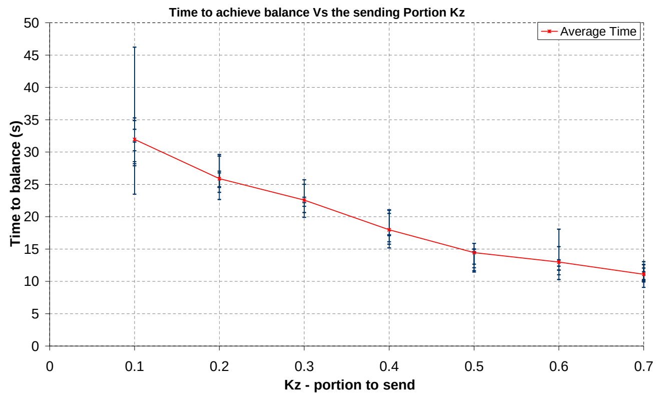

Fig. 9. Summary of the load balancing time as function of the gain K_z.

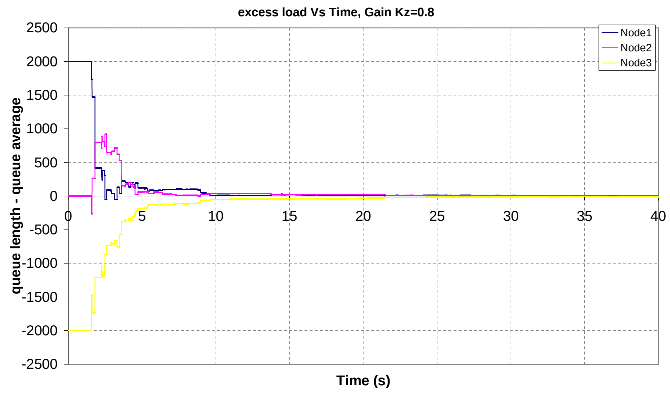

Fig. 10. Experimental response of the load balancing algorithm under large delays. gain $K_z = 0.8$ and $p_{ij} = 0.5$.

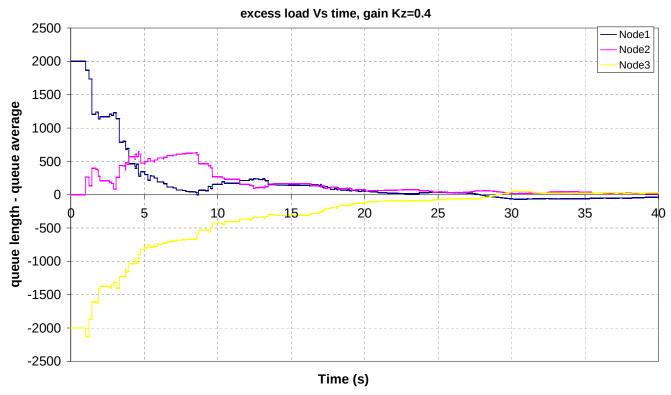

Fig. 11. Experimental response of the load balancing algorithm under large delays. gain $K_z = 0.4$ and $p_{ij} = 0.5$.

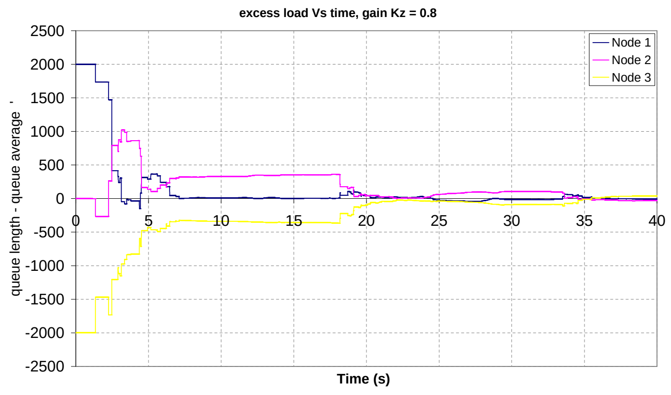

Fig. 12. Experimental response of the load balancing algorithm under large delays. gain $K_z = 0.8$ and $p_{ij} = 0.5$.

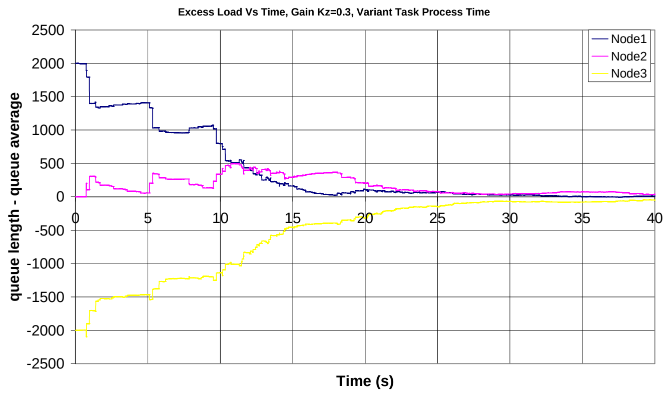

Fig. 13. Experimental response of the load balancing algorithm under large variance in the tasks processing time. gain $K_z = 0.3$ and $p_{ij} = 0.5$.

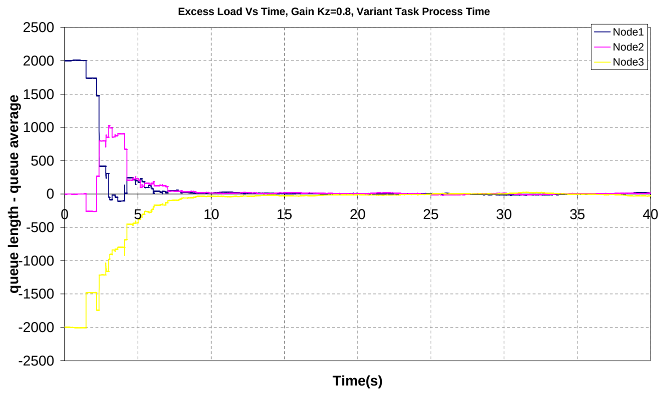

Fig. 14. Experimental response of the load balancing algorithm under large variance in the tasks processing time. gain $K_z = 0.8$ and $p_{ij} = 0.5$.

22ms. On the other hand, when the gain was set to 0.8, the system did not reach a stable point as shown by the nodes' ringing responses $r_i(t)$ in Figure 12.

As 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 the system behavior, the matrix multiplication application was adjusted in a way to get the following results; the average task processing time was kept at 10.2ms but the standard deviation became 7.15 ms instead of 2.5 ms. Figures 13 and 14 show the respective system responses for gains $K_z = 0.3$ and $K_z = 0.8$. Comparing Figures 7 and 13, we can see that in the latter case, some ringing persists and the system did not completely stabilize. On the other hand, setting the gain K_z to 0.8 led the system to accommodate with the variances in the task processing time.

The experiments presented in this section validate the ones done in the previous section where a local area network was involved.

V. Summary and Conclusions

In this paper, a nonlinear time-delay model of the load balancing algorithm was presented. Experiments have indicated a correlation between the continuous time models and the actual implementation. Further, the results drawn from the two test-beds were consistent with each other. In particular, high gains were shown to be inefficient and therefore introduce drawbacks in systems with large delays. Conversely, systems with low gain values could not cope with the variability introduced by the tasks processing time. Therefore, one should avoid the limits and carefully choose an adequate gain value. Future work will consider incorporating the network delays as detected by the system in the fraction coefficients p_{ij} in order to quickly stabilize the system and therefore decrease the overall completion time.

References

[1] E. Altman and H. Kameda, "Equilibria for multiclass routing in multi-agent networks", December 2001.Orlando, FL USA.

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

[3] H. Kameda, I. R. El-Zoghdy Said Fathy, and J. Li, A performance comparison of dynanmic versus static load balancing policies in a mainframe, in Proceedings of the 2000 IEEE Conference on Decision and Control, pp. 14151420, December 2000. Sydney, Australia.

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

[5] M. Willebeek-LeMair and A. Reeves, Strategies for dynamic load balancing on highly parallel computers, IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 9, pp. 979993, 1993.

[6] J. D. Birdwell, J. Chiasson, Z. Tang, C. T. Abdallah, M. Hayat, Z. Tang, 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. Also, to appear in an edited book by Springer-Verlag, Keqin Gu and Silviu-Iulian Niculescu, editors.

[7] C. Abdallah, J. Birdwell, J. Chiasson, V. Churpryna, Z. Tang, and T. 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.

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

[9] P. Dasgupta, Performance Evaluation of Fast Ethernet, ATM and Myrinet under PVM, MS Thesis. University of Tennesse, 2001.

[10] P. Dasgupta, J. D. Birdwell, and T. W. Wang, Timing and congestion studies under PVM, in Tenth SIAM Conference on Parallel Processing for Scientific Computation, March 2001. Portsmouth, VA.