2005 American Control Conference
June 8-10, 2005. Portland, OR, USA

Closed-loop Load Balancing: Comparison of a Discrete Event Simulation with Experiments

Zhong Tang, John White, John Chiasson, J. Douglas Birdwell, Chaouki T. Abdallah and Majeed M. Hayat

Abstract— **Load balancing for parallel computations is modeled as a deterministic dynamic nonlinear time-delay system. This model accounts for the trade-off between using processor time/network bandwidth and the advantage of distributing the load evenly between the nodes to reduce overall processing time. A distributed closed-loop controller is presented to balance load dynamically at each node by using not only the local estimate of the queue size of other nodes, but also estimates of the number of tasks in transit. A discrete event simulation using OPNET Modeler is presented and compared with experimental data, and results indicate good agreement between the nonlinear time-delay model and the behaviors observed on a parallel computer network. Moreover, both simulations and experiments show a dramatic increase in performance obtained using the proposed closed-loop controller.**

I. INTRODUCTION

The objectives of parallel processing are to reduce wall-clock time, increase throughput, and increase the size of solvable problems by dividing the software into multiple fragments that can be executed simultaneously on a set of computational elements (CE) interconnected via a high bandwidth network. To effectively utilize a parallel computer architecture, the computational loads 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.

Various taxonomies of load balancing algorithms exist in the literature [1][2]. 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 [3] and gradient methods, stochastic, or optimization based. A comparison of several deterministic methods is provided by Willebeek-LeMair and Reeves [4]. Approaches to modeling and iterative load balancing are given in [5][6][7]. In recent years, there has been active work on load balancing using control theory, especially for database applications and web services [8][9][10][11]. A queuing theory [12] approach is well-suited to the modeling requirements and has been used in the literature by Spies [13] 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. Previous results by the authors appear in [14][15][16][17].

There is a trade-off between using processor time/network bandwidth and the advantage of distributing the load between nodes to reduce overall processing time. Our work in [18] discusses a mathematical model that captures the processor resource constraints in load balancing. The open-loop experiments and Simulink simulations correspond well. The work has been extended to the closed-loop control of a load balancing network, and some initial results are presented in [19]. However, Simulink does not easily handle time varying delays which arise in the closed-loop case. This motivated the authors to develop a new discrete event simulation based on OPNET Modeler.

This work presents the closed-loop control of a load balancing network with time delays and processor resource constraints. The closed-loop controller at each node uses not only the local estimate of the queue sizes of other nodes, but also estimates of the number of tasks in transit to it. A discrete event simulation using OPNET Modeler is presented and compared with the experiments on a parallel computer network. The OPNET Modeler simulations indicate good agreement of the nonlinear time-delay model with the actual implementation. Both OPNET simulations and experimental results show the superiority of using the controller based on the anticipated queue sizes to using the controller based on the local queue sizes only.

Section II presents a model of a load balancing algorithm in the computer network that incorporates the presence of time delays in communicating between nodes and transferring tasks. Section III addresses the feedback control law on a local node and how a node portions out its tasks to other nodes. Feedback controllers based on the actual

Z. Tang, J. White, J. Chiasson and J. D. Birdwell are with ECE Dept, The University of Tennessee, Knoxville, TN 37996-2100, USA, ztang@utk.edu, jwhite@lit.net, chiasson@utk.edu, birdwell@utk.edu

C. T. Abdallah and M. M. Hayat are with ECE Dept, University of New Mexico, Albuquerque, NM 87131-1356, USA, chaouki@ece.unm.edu, hayat@ece.unm.edu

The work of J. D. Birdwell, J. Chiasson, Z. Tang and J. White was supported by the National Science Foundation under grant number ANI-0312182. The work of C. T. Abdallah and M. M. Hayat was supported by the National Science Foundation through grant ANI-0312611. We also express our gratitude to the OPNET Technologies, Inc. for providing us with an academic license for their OPNET Modeler network simulation software. 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.

0-7803-9098-9/05/$25.00 ©2005 AACC

queue size and on the *anticipated* queue size are discussed in this section. Section IV presents the OPNET model of a load balancing system. Simulations and experiments are presented to demonstrate the feedback controller based on the anticipated queue size. Section V is summarizes the present work.

II. 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 may be 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.

A simple approach to load balancing would be to have each computer in the network broadcast 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 can then use this information to compute its local estimate[1] 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, can be used as the network average. Node i would 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. 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. It was shown in [19] that this straightforward controller leads to unnecessary task transfers (the queue lengths oscillate) due to the time delays. A modification of this controller is proposed below that avoids unnecessary task transfers.

As 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. The model used here captures the effect of the delays in load balancing techniques as well as the processor constraints so that system theoretic methods can be used to analyze them.

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

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

where

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

In this model

- 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.
- $\lambda_i \ge 0$ 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).
- $\mu_i \ge 0$ 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.
- $\eta_i = 0$ or 1 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.

In this model, all rates are in units of the rate of change of expected waiting time, or *time/time* which is dimensionless.

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

As $\eta_i = 1$ or 0, node i can only send tasks to other nodes and cannot initiate transfers from another node to itself. A delay is experienced by transmitted tasks before they are received at the other node. Model (1) is the basic model, the p_{ji} defines how to portion the tasks to be transferred on each sending node i. One approach is to choose them as constant and equal

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

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

The model (1) is shown in [18] 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). The model is only (Lyapunov) stable, and asymptotic stability must be insured by the choice of the feedback law.

III. FEEDBACK CONTROL

In [18], a 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. However, there is additional information that can be made available to the nodes — specifically, the information on q_{net_i}, which is the number of tasks that are in the network being sent to the i^{th} node, 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} in transit 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

$$z_i \triangleq x_i + x_{net_i} = t_{p_i}\left(q_i + q_{net_i}\right) \tag{3}$$

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

$$z_{i\_avg} \triangleq \left(\sum_{j=1}^{n} z_j(t-\tau_{ij})\right)/n \tag{4}$$

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,

$$w_i(t) \triangleq x_i(t) - z_{i\_avg}(t) = x_i(t) - \frac{\sum_{j=1}^{n} z_j(t-\tau_{ij})}{n} \tag{5}$$

to be the expected waiting time relative to the estimate of average (anticipated) waiting time in the network by the i^{th} node. By using the waiting times $z_i(t)$ in (5) (rather than $x_i(t)$) unnecessary task transfers are avoided (see [19]). A control law based on the anticipated waiting times is chosen as

$$\eta_i(t) = h\left(w_i(t)\right). \tag{6}$$

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

$$h(w) = \begin{cases} 1 & \text{if} \quad w \geqslant 0 \\ 0 & \text{if} \quad w < 0. \end{cases}$$

The control law (6) states a balancing action is needed on node i if its local waiting time is above the estimate of the anticipated network average waiting time. How a sending node portions out its excess load to transfer to other nodes is determined by the p_{ij}. Rather than set the p_{ij} constant as in (2), they are specified by equation (7) below.

The quantity p_{ij} is the fraction of waiting time above the network average to be transferred from node j to node i. The p_{ij} can be specified using the anticipated waiting times 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 $\text{sat}\left(z_{j\_avg} - z_i(t-\tau_{ji})\right)$ 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,

$$p_{ij} = \frac{\text{sat}\left(z_{j\_avg} - z_i(t-\tau_{ji})\right)}{\sum\limits_{i \ni i \neq j} \text{sat}\left(z_{j\_avg} - z_i(t-\tau_{ji})\right)}. \tag{7}$$

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.

IV. EXPERIMENTAL RESULTS

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. In the experimental facility, all machines 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 engine threads access tree-structured indices to locate database records that match search or store requests. 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.

The OPNET Modeler [20] is a tool suite for the creation and analysis of discrete event simulations of computer networks. A network in OPNET Modeler is built of many components including network models, node models, link models, packet formats, and process models. By configuring specific parameters for each of these components and writing C code to describe the behavior of the component, a model was created to simulate the load balancing algorithm behavior with time-varying delays over different network topologies.

In the OPNET model of a load balancing system, three nodes are connected by a model of a gigabit switch. Each node simulates the load balancing algorithm using a process model with a given set of initial conditions. Figure 1 shows the process model at each node, which is the lowest level of the OPNET model hierarchy. A process model is a finite state machine, in which events from the simulation kernel produce conditions that cause transitions between states. The `init` state initializes system variables, schedules `SEND_STATS` and `PROC_JOB` transitions, and immediately transitions to the `idle` state. Here the `idle` state waits for interrupts from the simulation kernel to cause transitions to other states. The `proc_job` state pulls a task from the queue, optionally generates additional tasks, updates its local average estimate, and schedules another `PROC_JOB` transition for $\sim 400\mu\,\text{sec}$ in the future. Transitions to the `send_stats` state are scheduled at regular intervals, sending the node's queue size to all other nodes. Additionally, the `send_stats` state will schedule a `LOADBAL` transition when it detects an imbalance in load based on the node's queue size and its local average estimate.

The `loadbal` state implements the load balancing algorithm by determining the number of tasks to send to each node, sending each node a notification of the number of tasks in transit to it, and then sending the actual tasks. The last action of the `loadbal` state is to schedule a `LOADBAL_FINISHED` transition, and to set a flag that will block transitions to the `proc_job` and `send_stats` states until the `LOADBAL_FINISHED` transition occurs. This is done to account for the fact that the load balancing algorithm and search tasks contend for the same processing resources. When packets are sent from a node, they are passed to the simulation kernel where the delay to transfer the packet to another node is determined based on the packet size, the link bandwidth, and optionally, a random propagation delay. Once the total delay is determined, a `PK_ARRVL` transition is scheduled at the receiving node for the appropriate time. When this transition occurs, the packet type is determined by the `det_pk_type` state to be a statistic update of another node's load, a notification of incoming tasks, or incoming tasks themselves and are handled by the `rcv_stats`, `rcv_job_info`, and `rcv_jobs` states, respectively.

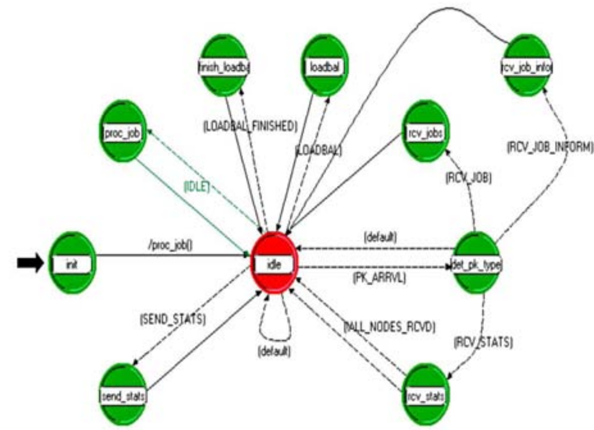

Fig. 1. OPNET simulation: process model in a load balancing system.

The node that transfers tasks computes the amounts to be sent to other nodes using (7). This amount is computed and sent before actually transferring tasks. Such communications are efficient; the communication delay of each transferred measurement is much smaller than the task transfer delays. Thus knowledge of the anticipated queue sizes can be used to compensate the effect of delays. OPNET simulations and experimental results on the parallel computer are presented here to illustrate the effects of time delays in closed loop load balancing.

A. Load Balancing with Initial Tasks

The OPNET Model is configured to match the characteristics of the actual load balancing experiments with an initial queue distribution of $q_1(0) = 600$, $q_2(0) = 200$ and $q_3(0) = 100$ tasks. The average time to do a search task is $400\mu\,\text{sec}$, and the inputs are set as $\lambda_1 = 0, \lambda_2 = 0, \lambda_3 = 0$. The actual delays experienced by the network traffic in the parallel machine are *random*. Experiments were performed and the resulting time delays were measured and analyzed. Those values were used in the simulation for comparison with the experiments.

Figure 2 shows the OPNET results of a 3-node load balancing using p_{ij} based on the anticipated waiting times z_i. The load balancer on each node uses the anticipated queue sizes which include both the queue information of other nodes and the task information in transit through the network. As the communication delay of each transferred measurement is much smaller than the actual task transfer delays, the node controller gets more up to date information to make its decision on load balancing. Figure 2 shows such a closed-loop controller which reduces unnecessary transfers (due to delayed information of other nodes, see [19]) resulting in a faster settling time.

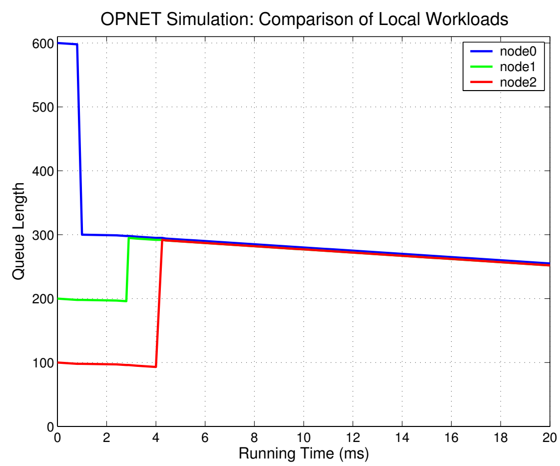

Fig. 2. OPNET 3-node simulation using controller (6) and p_{ij} by (7).

Figure 3 shows the experimental data of the responses of the queues versus time for load balancing with initial queues of $q_1(0) = 600$, $q_2(0) = 200$ and $q_3(0) = 100$ tasks. In Figure 3, the system reaches the balanced state quickly using the anticipated waiting times. Although all trajectories contain random effects and therefore can not be compared point by point, the qualitative behaviors of the OPNET simulation and the experiment are quite similar.

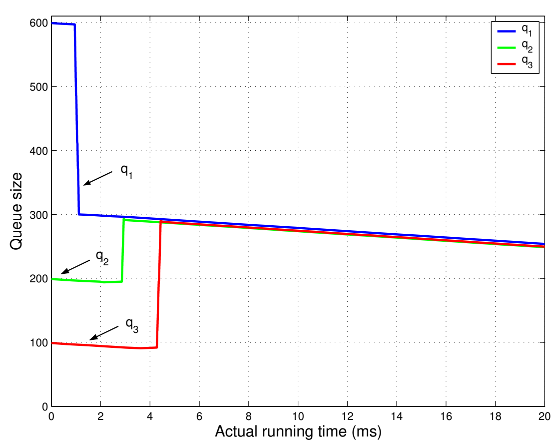

Fig. 3. Plot of queue sizes using controller (6) and p_{ij} by (7).

Figure 4 shows node 2's estimates of the anticipated queue sizes in the network under closed-loop load balancing with the p_{ij} based on the z_i. Node 1 broadcasts the number of tasks it is sending to each node before actually transferring the tasks to the other nodes. Specifically, $q_1 + q_{net_1}$ in Figure 4 is what node 2 estimates the total tasks at node 1 or in transit to node 1, and $q_3 + q_{net_3}$ is what node 2 estimates the total tasks at node 3 or in transit to node 3. Node 2 uses the (anticipated) estimates $q_1 + q_{net_1}$ and $q_3 + q_{net_3}$ in the controller (5)(6) to balance its load. From Figure 4, the anticipated estimates are used by the controller to compensate the effect of delays in the task transfers so that no unnecessary task transfers are initiated. This method quickly balances computational workloads across all nodes and results in a shorter job completion time.

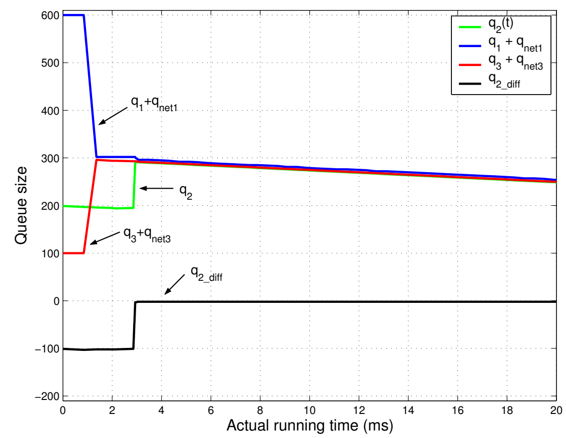

Fig. 4. Estimated queue sizes by node 2 using (6) and (7).

B. Load Balancing with Task Generation

In this experiment, the initial queue distribution is $q_1(0) = 600$ 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, and the inputs were set as $\lambda_1 = 3, \lambda_2 = 0, \lambda_3 = 0$. That is, a number of new tasks is generated at the rate λ_1 on node 1 in each processing loop. It is interesting to see how the load balancing algorithm performs as tasks are dynamically generated.

Figures 5 shows the OPNET simulation of a 3-node load balancing using the p_{ij} specified by (7) with task generation in the process of execution. Figure 6 shows the responses of the queues versus time in an actual experimental run. The staircase-like increases of queue size corresponds to new task insertions in node 1 at the rate $\lambda_1 = 3$. Figures 5 and 6 show that the control system quickly acts to balance the nodes using the anticipated waiting times. The OPNET simulations indicate good agreement between the event-based nonlinear time delay model and the actual implementation.

V. Summary

In this work, a load balancing algorithm for parallel computing was modeled as a nonlinear dynamic system in the presence of time delays and processor resource constraints. A closed-loop controller was presented based on the local queue size and an estimate of the tasks being sent to the queue from other nodes. The proposed control law on each node used not only its estimate of the queue

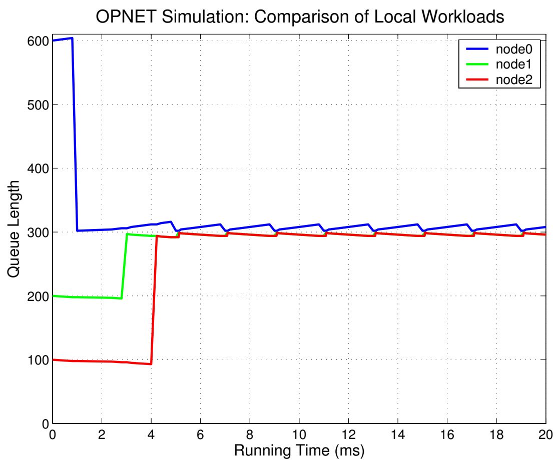

Fig. 5. OPNET 3-node simulation of load balancing with task generation.

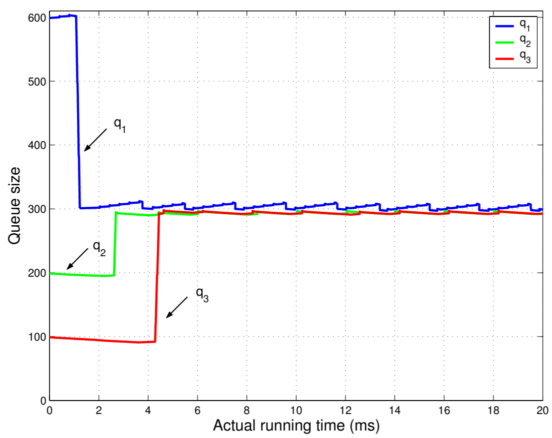

Fig. 6. Plot of queue sizes in load balancing with task generation.

sizes at other nodes, but also an estimate of the number of tasks in transit to it. The system achieved faster settling times than reported in [18] by using this information to avoid unnecessary transfers. An OPNET simulation model was presented to include the time varying delays arising in the closed-loop load balancing process. The OPNET simulations indicated good agreement of the nonlinear time delay model with the actual implementation. Both simulations and experimental results showed a substantial improvement in performance obtained using the closed-loop controller based on anticipated queue sizes.

References

[1] H. G. Rotithor, "Taxonomy of dynamic task scheduling schemes in distributed computing systems," *IEE Proceedings on Computer and Digital Techniques*, vol. 141, no. 1, pp. 1–10, 1994.

[2] M. A. Senar, A. Corteś, A. Ripoll, L. Hluchý, and J. Astalos, "Dynamic load balancing," in *Parallel Program Development for Cluster Computing: methodology, tools and integrated environments* (J. C. Cunha, P. Kacsuk, and S. C. Winter, eds.), Advances in Computation: Theory and Practice, pp. 69–95, Nova Science, 2001. New York.

[3] A. Corradi, L. Leonardi, and F. Zambonelli, "Diffusive load-balancing policies for dynamic applications," *IEEE Concurrency*, vol. 7, no. 1, pp. 22–31, 1999.

[4] 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.

[5] H. Kameda, J. Li, C. Kim, and Y. Zhang, *Optimal Load Balancing in Distributed Computer Systems*. Springer, 1997.

[6] H. Kameda, E.-Z. S. Fathy, I. Ryu, 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.

[7] 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.

[8] J. L. Hellerstein, "Challenges in control engineering of computing systems," in *Proceedings of the 2004 American Control Conference*, pp. 1970–1979, June 2004. Boston, MA, USA.

[9] Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. Pareky, and C. Garcia-Arellano, "Using MIMO linear control for load balancing in computing systems," in *Proceedings of the 2004 American Control Conference*, pp. 2045–2050, June 2004. Boston, MA, USA.

[10] A. Robertsson, B. Wittenmark, M. Kihl, and M. Andersson, "Design and evaluation of load control in web server systems," in *Proceedings of the 2004 American Control Conference*, pp. 1980–1985, June 2004. Boston, MA, USA.

[11] T. Abdelzaher, Y. Lu, R. Zhang, and D. Henriksson, "Practical application of control theory to web services," in *Proceedings of the 2004 American Control Conference*, pp. 1992–1997, June 2004. Boston, MA, USA.

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

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

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

[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 *Advances in Time-Delay Systems* (S.-I. Niculescu and K. Gu, eds.), vol. 38 of *Lecture Notes in Computational Science and Engineering*, pp. 355–370, Springer-Verlag, 2003.

[16] M. Hayat, S. Dhakal, 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 *Advances in Time-Delay Systems* (S.-I. Niculescu and K. Gu, eds.), vol. 38 of *Lecture Notes in Computational Science and Engineering*, pp. 371–385, Springer-Verlag, 2003.

[17] 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*, pp. 582–587, December 2003. Maui, HI, USA.

[18] Z. Tang, J. D. Birdwell, J. Chiasson, C. T. Abdallah, and M. Hayat, "A time delay model for load balancing with processor resource constraints," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, pp. 4193–4198, December 2004. Paradise Island, Bahamas.

[19] Z. Tang, J. D. Birdwell, J. Chiasson, C. Abdallah, and M. Hayat, "Closed loop control of a load balancing network with time delays and processor resource constraints," in *Advances in Communication Control Networks* (S. Tarbouriech, C. Abdallah, and J. Chiasson, eds.), vol. 308 of *Lecture Notes in Control and Information Sciences*, pp. 245–268, Springer, 2004.

[20] OPNET Technologies Inc., "OPNET Modeler," 2004. Available: http://www.opnet.com/products/modeler/home.html.