Proceedings of the 42nd IEEE
Conference on Decision and Control
Maui, Hawaii USA, December 2003

Dynamical Discrete-Time Load Balancing in Distributed Systems in the Presence of Time Delays

S. Dhakal B. S. Paskaleva M. M. Hayat E. Schamiloglu C. T. Abdallah

Department of Electrical and Computer Engineering
University of New Mexico
Albuquerque, NM 87131-1356, USA

Abstract— **The implementation of a load balancing policy on a continuous basis in a delay-limited distributed computing environment may not only drain the computational resources of each computational element (CE), but can also lead to an unnecessary exchange of loads between the CEs. This degrades the system performance, measured by the overall completion time of the total tasks in the system. Thus, for a given distribution of the load among the CEs, there has to be an optimal number and distribution of discrete balancing instants. This paper focuses on £xing the number of balancing instants and optimizing the completion time over the strength of load balancing, which is controlled by the so-called gain parameter, and the time when the balancing is executed. First, the case when the load balancing is implemented at a single instant per node is presented. Then, a strategy is considered where a second load balancing instant is allowed for each node. The simulations show that both strategies outperform the continuous balancing policy. Moreover, with the double load-balancing strategy the overall completion time is further reduced in comparison to the single load balancing case. It is also seen that the optimal choice of the gain parameter depends on the delay and this dependence becomes more signi£cant as the delays increase. This interplay between the strength of load balancing and the magnitude delay has a direct effect on the performance of the policy and on the sensitivity to the selection of the balancing instants.**

I. INTRODUCTION

The development of effective and computationally ef£cient load balancing techniques is an essential task in parallel and distributed computing environments. Effective load balancing relies on accurate knowledge of the state of the individual computational elements (CEs) whereby such knowledge is used to distribute the incoming computational tasks to appropriate CEs in accordance to a load balancing policy. However, large-scale distributed computing systems with physically and/or logically distant CEs inherently involve time delays. Consequently, the information that a particular node has about other nodes, at any given time, is dated and may not accurately represent their current states. Such time-delay factors can seriously alter the expected performance of load balancing policies designed without taking into account such delays. One source of time delay is the computational limitations of the CEs and the execution of the load balancing policy itself. A more signi£cant source of time delay is the limitations imposed by the communication medium between the CEs. This includes delays in transferring a load between the nodes and delays in the exchange in communications between them. For example, network QoS factors such as latency, congestion and corruption, can signi£cantly contribute to delays during dynamic load balancing when loads are being re-distributed between the CEs. In addition, the delays encountered in dynamic load balancing are actually random due to the uncertainty in the condition of the shared network that connects the CEs. Such network conditions include the level of traf£c and network con£guration and architecture. This uncertainty is particularly prominent in wireless networks, and especially for satellite links that operate at very high transmission rates with low bandwidths and relatively high bit-error rates. Other factors that contribute to the stochastic nature of the distributed-computing problem are: randomness and possible burst-like nature of the arrival of new job requests at each node from external sources, randomness of the load-transfer process itself being queue-size dependent, and randomness in the task completion process at each node. In the recent years, dynamic load balancing has been an area of extensive research and a number of strategies have been proposed [1], [2], [3], [4]. Generally, the developed policies can be categorized according to their operational settings, which include local versus global, static versus dynamic, and centralized versus distributed scheduling. However, there have been very limited focus on the inherent latency involved in geographically-distant distributed systems and the stochastic behavior of this latency.

Thus, if we are designing a load-balancing policy under no delay or £xed-delay assumptions, the policy will not perform as expected in real situation when delays are non-zero or random. To adequately describe and investigate load balancing behavior in such delay-infested environments, we have previously taken a new look at the problem of dynamic load balancing using a dynamical model that captures the stochastic delays discussed above [5], [6]. We incorporated the stochastic dynamics of load balancing and applied the model to predict the impact of random delays on the performance. In particular, we considered the effect of the strength of load balancing, which is governed by the fraction,

0-7803-7924-1/03/$17.00 ©2003 IEEE

K, that dictates what portion of any CE's excess load should be assigned to other CEs [7]. In the ideal case where the communication and load-transfer delays are negligible (as in a fast Ethernet environment) and the time required to implement the load-balancing policy is also negligible, the best performance (minimizing the waiting times associated with all CEs) is obtained when the load balancing is executed almost continuously without any reservation. Namely, at almost every instant, each CE compares its queue size to the average queue size of the network and distributes all its excess load to other nodes. Every other node also follows a similar policy. However, in a practical setting such a strategy has two main disadvantages: 1) the implementation of the load balancing policy on a continuous basis can drain the computational resources of each CE; and 2) excessive load balancing, both in frequency and strength, can lead to timely and possibly unnecessary exchange of loads between CEs. This means that valuable time may be unduly waisted exchanging loads back and forth between nodes (as the system is diligently attempting to balance the queues) while this time could have been used to actually execute the tasks submitted! In fact. in our prior work [5] we have shown that if the delays are dominated by the communication and load transfer, then there is an optimal load-balancing strength (viz., an optimal value for the parameter K), that minimizes the waiting time in each CE. In particular, we have shown that the strength of the load-balancing policy must be reduced in a delayed environment to avoid any "over-reaction" consequences that may arise due to such delay factors. In [1], remote communication is minimized by reducing the number of balancing instants between the distant CEs while allowing the local CEs to balance continuously. To improve the network efficiency, Hui and Chanson [4] proposed transferring a single large job by combining several balancing instants.

In a more practical setting, the continuous implementation of load balancing, as we stated earlier, can be very costly (wasteful of computational resources) and more importantly, it can inflict an additional delay, namely, the time needed to implement the load balancing policy. Thus, there is an inherent tradeoff between the strength and frequency of load balancing on one hand, and the need to conserve computational resources used in implementing any load-balancing policy. Motivated by such a fundamental tradeoff, in this paper we investigate whether limiting the number of load balancing instants while optimizing the strength of the load balancing and the actual load-balancing instants is a feasible solution to the problem of load balancing in a delay-limited environment. This paper addresses the performance of such a potentially computationally-efficient load-balancing strategy.

II. Description of the Stochastic Dynamical Model

We begin by briefly describing the queuing model that characterizes the stochastic dynamics of the load balancing problem described so far drawing freely from our prior work [5], [7]. This model is subsequently used as basis for the development of a custom-made simulation software used to generate all the results included in this paper.

Suppose that we have a cluster of n nodes. Let $Q_i(t)$ denote the number of tasks awaiting processing at the ith node at time t. Assume that the ith node completes tasks according to a Poisson process and at a constant rate μ_i. Let the counting process $J_i(t_1, t_2)$ denote the number of external tasks (requests) arriving at node i in the interval $[t_1, t_2)$. We will assume that the process $J_i(t_1, t_2)$ is a compound Poisson process with a constant rate λ_i [8], that is, $J_i(t_1, t_2) = \sum_{k:t_1 \le \xi_k < t_2} H_k$, where ξ_k are arrival times of job requests arriving according to a Poisson process with rate λ_i. The random sequence $H_k, k = 1, 2 \ldots$, is a sequence of integer-valued random variables describing the number of tasks associated with the kth job request. The load balancing mechanism is described as follows: The ith node, at a specific load-balancing instant T_l^i, looks at its own load $Q_i(T_l^i)$ and the loads of other nodes at randomly delayed instants (due to communication delays), and decides whether it should allocate a fraction K of its load to the other nodes according to a deterministic policy. Moreover, at the time when it is not balancing its load, it may receive loads from the neighboring nodes subject to random delays (due to the load-transfer delays).

With the above description of task assignments between nodes, we can write the dynamics of the ith queue in a differential form as (in Δt time increments):

$$Q_i(t+\Delta t) = Q_i(t) - C_i(t+\Delta t) - \sum_{j \neq i} L_{ji}(t) + \sum_{j \neq i} L_{ij}(t - \tau_{ij}(t)) + J_i(t, t+\Delta t), \quad (1)$$

where $C_i(t+\Delta t)$ is a Poisson process (with rate μ_i) describing the random number of tasks completed in the interval $[t, t+\Delta t)$, $J_i(t, t+\Delta t)$ is a random number of new, external tasks arriving in the same interval, $\tau_{ij}(t)$ is the delay in transferring load from node j to node i at the same interval, and $L_{ji}(t)$ is the load transferred from node i to j in the interval $[t, t+\Delta t)$. More precisely, for any $k \neq l$, the random load L_{kl} diverted from node l to node k has the form $L_{kl}(t) \triangleq g_{kl}(Q_l(t), Q_k(t - \eta_{lk}(t)), \ldots, Q_j(t - \eta_{lj}(t)), \ldots)$, where for any $j \neq k$, $\eta_{kj}(t) = \eta_{jk}(t)$ is the communication delay between the kth and jth nodes at time t. The function g_{kl} dictates the load-balancing policy between the kth and lth nodes.

One common example is

$$g_{lk}(Q_l(t), Q_k(t-\eta_{lk}(t)), \ldots, Q_j(t-\eta_{lj}(t)), \ldots)$$
$$= K_k p_{lk} \cdot \Big(Q_l(t) - n^{-1}\sum_{j=1}^{n} Q_j(t-\eta_{lj}(t))\Big) \cdot$$
$$u\Big(Q_l(t) - n^{-1}\sum_{j=1}^{n} Q_j(t-\eta_{lj}(t))\Big), \quad (2)$$

where $u(\cdot)$ is the unit step function with the obvious convention $\eta_{ii}(t) = 0$, and K_k is a parameter that controls the "strength" or "gain" of load balancing at the kth (load distributing) node. In this example, the lth node simply compares its load to the average over all nodes and sends out a fraction p_{lk} of its excess load to the lth node. (Of course, $\sum_{l \neq k} p_{lk} = 1$.)

III. Simulation Results

Consider a cluster of three nodes with equal computing power (i.e., the task completion rates, $\mu_i, i = 1,2,3$, are all the same), and let us assume that each node is allowed to execute load balancing at only two scheduling times. Throughout this paper, we will assume that the average task completion time is 10 μs per task, and the load-balancing policy is implemented according to the policy described in the previous section. The initial load for these experiments was distributed unevenly among the three nodes as 7000, 4500, and 500 tasks, with no additional external arrival of tasks (in this paper we only consider the zero-input response).

Some of our earlier experimental results that motivated the present study are summarized in Fig. 1. The top graph in this £gure shows the empirical average of the queue size (dashed curves show the number tasks cumulatively performed). It is seen that approximately only 87% of the total tasks were completed within 60 ms. The fact that the total number of tasks performed by each CE are not the same indicates that load-balancing has not been effective (since all nodes have the same computing capability), which is attributed mainly to the presence of delay. To have better insight into the time elapsed before all the tasks are computed, we generated the empirical variance of the queues, as shown by the bottom graph in Fig. 1. The graph shows a high-degree of uncertainty in the smallest queue and, more importantly, near the tail of the queues (beyond 30 ms). We observed that even in the fastest completion period, 95% of the tasks were completed around 15 ms faster then the time taken to complete the last 5% the tasks. This is an indicator that the nodes are continuing to exchange tasks back and forth near the tail of the queue even when load-balancing seems unnecessary. The more often we try to equalize the work load between the nodes, the more often portions of loads are transferred between the CEs. As a result, the CEs are not able to complete their assigned tasks by the time of the new load balancing policy execution. The net effect is that loads are bouncing between the nodes with little actual work being performed.

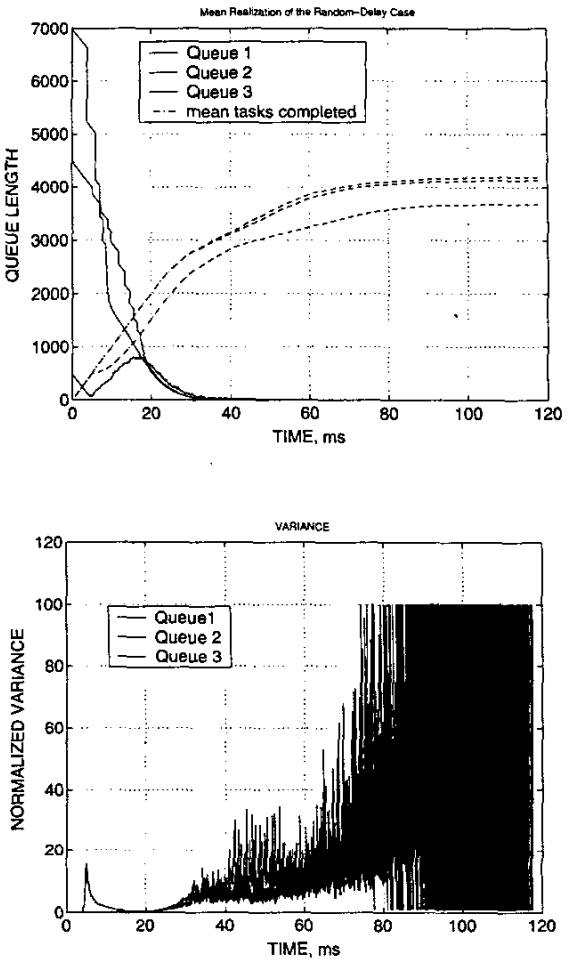

Fig. 1. Top: The empirical mean queue length using 100 realizations of the queues for each node (solid curves). Dashed curves are empirical averages of the tasks performed by each node cumulatively in time. Bottom: The empirical variance of the queue length normalized by the mean-square values.

A. Single Load-balancing Strategy

We now present the results for the case when load-balancing is implemented at a single instant only per node. We assumed initial loads of $Q_1(0) = 7000$, $Q_2(0) = 4500$, and $Q_3(0) = 500$, and an average communication and load-transfer delays of 8 ms (corresponding to relatively short load balancing transfer delays). The results showed that the optimal value for the load-balancing strength parameter K_{opt} is 0.8 ms, the optimal load balancing instant t_{bal1} is 0.02 ms, and the corresponding completion time t_{compl} is 47.57 ms, as seen in Fig. 2 (top). Now from the bottom graph in Fig. 2, we can see that the queue lengths change abruptly as a result of load-balancing events associated with the three

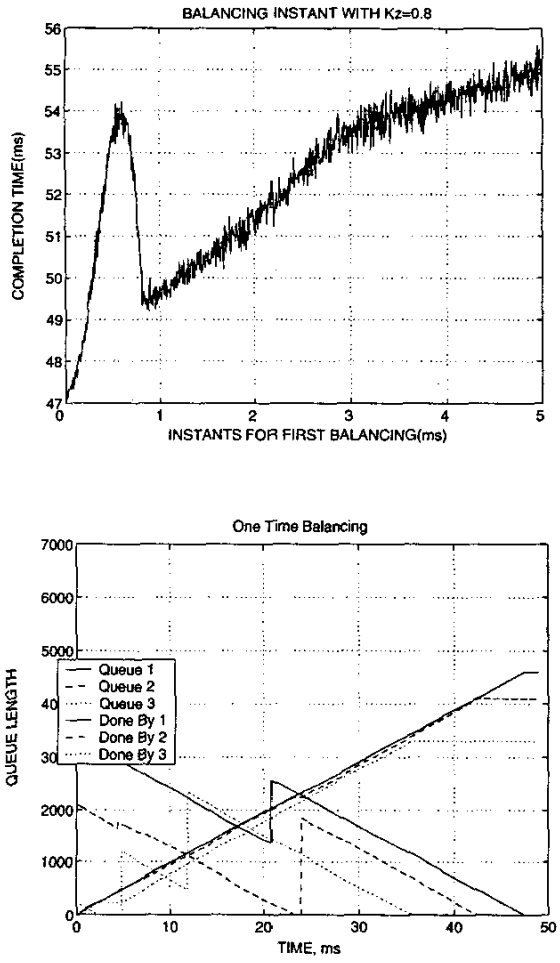

Fig. 2. Optimal single load-balancing scheduling for the short-delay case. Top: completion time vs. load-balancing instant, t_{bal1}; Bottom: queue lengths and cumulative tasks completed by each node.

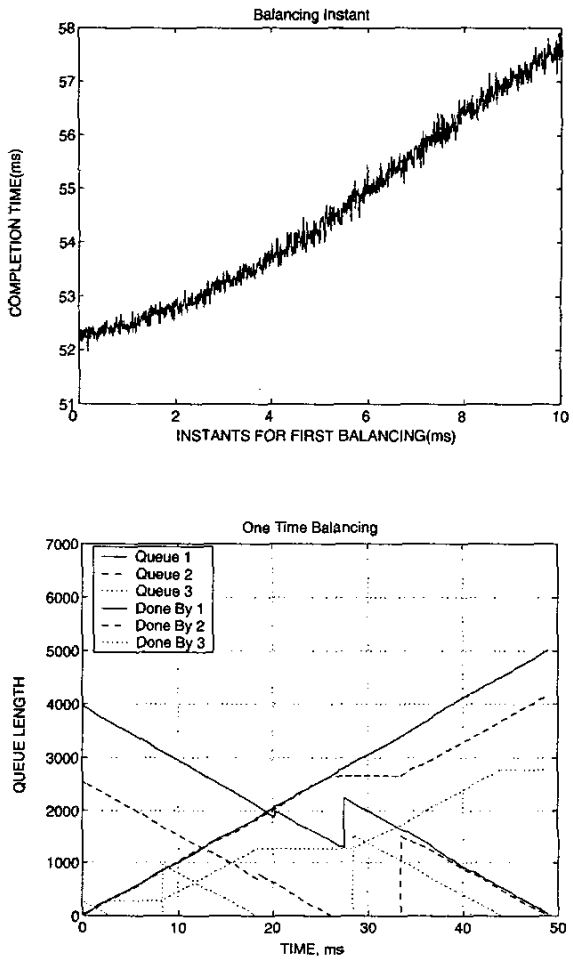

Fig. 3. Single load-balancing scheduling for the long-delay case. Top: completion time as a function of t_{bal1}; Bottom: queue evolution and the cumulative tasks done by each node.

nodes (a total of six transitions and two transitions per node in this case: one transition when a node transmits tasks to other nodes, and once when it receives the tasks that were sent to it). The group of increasing curves represents the tasks completed cumulatively in time by each node. We also noticed that when K ranges between 0.4 and 0.9, the completion time £rst decreases to a minimum of 47.57 ms, and then increases to 55 ms. The optimal range of the gain parameter is between 0.7 and 0.8. Within this range t_{bal1} is changing from 0.01 ms to 3.68 ms. Therefore, for relatively small communications delays, we can execute the load balancing policy either before the present states of the neighboring nodes are known, or after we receive this information. Nevertheless, there is a tradeoff involved in choosing one choice over the other. If completion time is the primary optimization goal, then it is advantageous to execute the load balancing policy at the very beginning, combined with a large value of the gain parameter. However, this comes at the price of sensitivity to any delay in executing the load balancing. For example, if the execution is delayed to just before the time when communication from other nodes arrive, then the completion is signi£cantly prolonged, as can be seen from the peak near $t_{bal1} = 0.6$ ms. On the other hand, if maintaining a stable (i.e., less sensitivity to error in the execution time) is sought, then it would be advantageous to execute the load balancing after receiving information from the neighboring nodes at a slight price of prolonged task completion time.

Next we consider a case where the delays are relatively long, both in communication and load transfer. As can be seen from the top plot in Fig. 3, the shortest completion time possible is approximately 52 ms for $t_{bal1} = 0.01$ ms and the optimal value of K is found to be 0.65. Like in the above scenario, there is no reason for CEs to wait for the information to reach them, because if

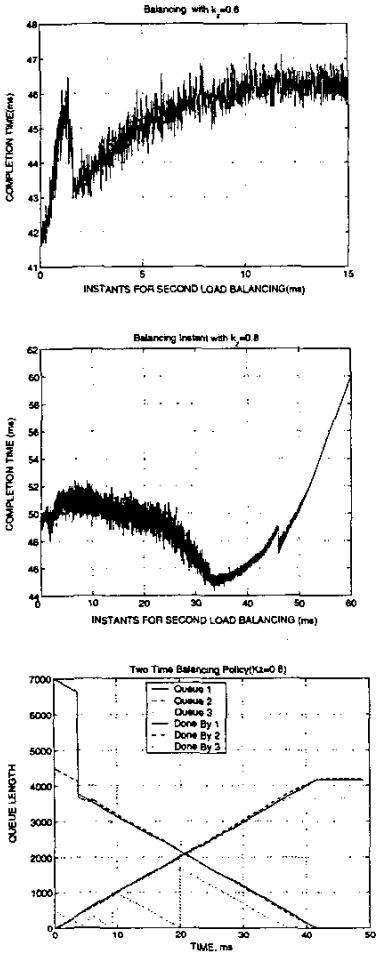

Fig. 4. Double load-balancing scheduling for the short-delay case. Top: $K = 0.6$, $t_{bal1} = 0.01$ ms, $t_{bal2} = 0.02$ ms; Middle: $K = 0.8$, $t_{bal1} = 0.01$ ms, $t_{bal2} = 3.69$ ms; Bottom: queue evolution length and the cumulative tasks done by each node.

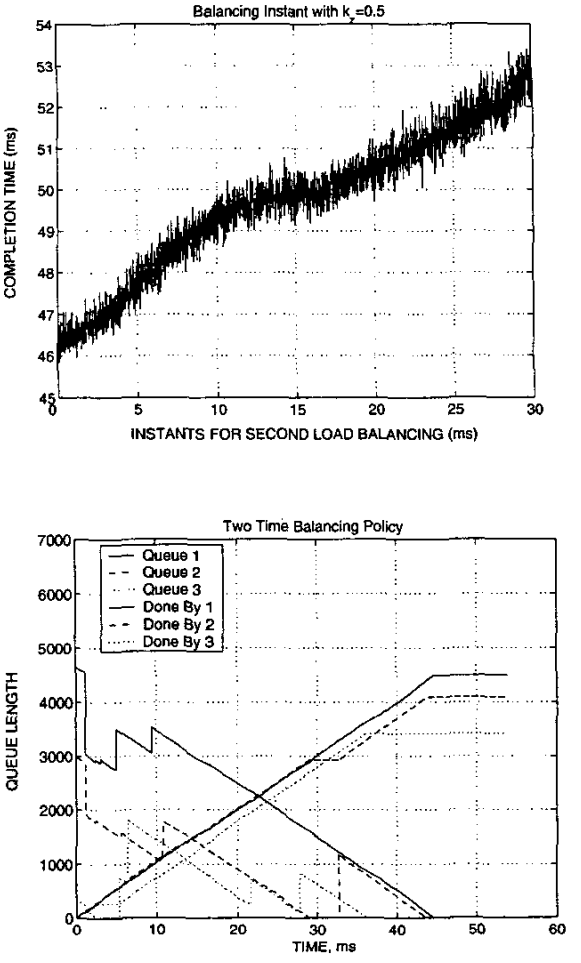

Fig. 5. Double load-balancing scheduling for the long-delay case. Top: $K = 0.5$, $t_{bal1} = 0.01$ ms, $t_{bal2} = 0.02$ ms; Bottom: queue length evolution and the cumulative tasks done by each node.

they do valuable time will be wasted (due to the large communication delay) since one node is idle. Thus, in this case, "informed" load balancing does not render effciency. Moreover, the optimal value of the balancing strength parameter has to be smaller compared to the case with short load-transfer delays. The reason is that in the present situation it will take longer for most of the information to reach its destination, and consequently, the overall completion time will increase. In addition, our simulations show that even with the optimal value of K, the task-completion time cannot reach the one corresponding to the short-delay case considered earlier. From the bottom plot in the same £gure we can see that CE1 and CE2 complete their work 5 ms after CE3. Thus, the system's load was not totally balanced.

To investigate the relationship between the initial load distribution and the optimal values for the system parameters, we considered a case where the initial loads are almost equally distributed between the nodes. In particular, we considered $Q_1(0) = 7000, Q_2(0) = 6500$, and $Q_3(0) = 6000$. For this setting, the shortest completion time was 66.51 ms at $K = 0.725$ and for $t_{bal1} = 0.63$ ms. These values are very close to the ideal case when no time delays are present and the minimum completion time for a total of 19500 tasks is 65 ms. From our empirical measurements we can conclude that when we have only one load-balancing execution per node in a small-delay environment, the best time to implement the load balancing is almost right at the beginning with a relatively large K (that actually depends on the initial load distribution). For the longer-delay case, however, K has to be decreased.

B. Double Load-balancing Strategy

Next, we consider a strategy for which a second load balancing instant, denoted by t_{bal2}, is allowed for each node. From the point of view of each node, t_{bal2} can

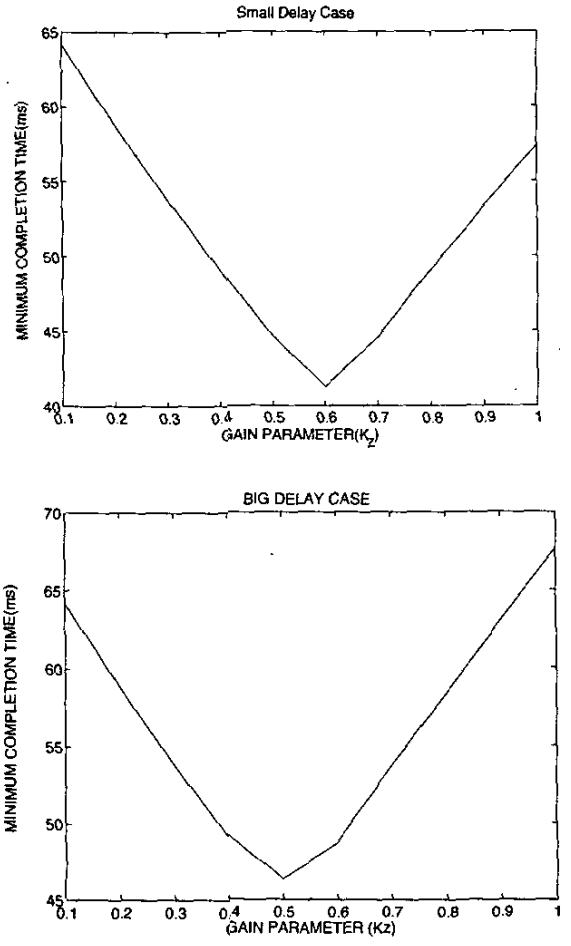

Fig. 6. Double load-balancing scheduling showing the task completion time as a function of the load-balancing strength parameter K. Top: small-delay case; Bottom: large-delay case.

be chosen using several options. For example, t_{bal2} can be chosen just after the first load balancing instant, between the moments in which the nodes are receiving loads from their neighbors, or at the end of the load exchange. If we choose $Q_1(0) = 7000$, $Q_2(0) = 4500$, $Q_3(0) = 500$, an average communication delay of 0.8 ms, and a similar average load-transfer delay, then the best t_{compl} is found to be 43.15 ms, which occurs when $t_{bal1} = 0.01$ ms and $t_{bal2} = 0.02$ ms with $K = 0.6$. A similar completion time can be achieved by executing the two load-balancing instants at a later time, after the nodes have received information. We found that this requires two load balancing instants following each other. In particular, our experiment shows that $t_{bal1} = 3.87$ and $t_{bal2} = 3.88$ yields one of the best completion times.

From the top plot in Fig. 4 we see that balancing in the beginning of the process leads to shorter completion times. The same plot indicates that execution of the load balancing within the range 0.03–2 ms is sensitive to error in the scheduling time. In particular, a small deviation of t_{bal} leads to a substantial increase in completion time. This time interval coincides with the time when every node receives information from its neighbors in a random way. Therefore, reliable load balancing is not possible during this time interval due to the communication delays. For the same reason, when $K = 0.8$, the best execution strategy is to execute the first load balancing policy right at the beginning with $t_{bal1} = 0.01$ ms and after that to wait until each one of the nodes received information from its neighbors before executing the second load balancing, as seen from Fig. 4 (middle). The completion time achieved in this case is 45 ms. Thus, qualitatively speaking, when we have two load-balancing instants in a small-delay environment, the optimal way to place them is either in the beginning, or immediately after the CEs have completed the information exchange.

For the case of large delays, the optimal solution with two load balancing instants is $K = 0.5$, $t_{bal1} = 0.01$ ms and $t_{bal2} = 0.02$ ms. While a completion time of approximately 46 ms is slightly higher than that in the previous case, it is still close to its optimal value. We see from Fig. 5 (top) that the two instants are in the beginning of the process. Long delays will cause nodes to use dated information to determine the load redistribution. We also found that the value of K_{opt} is lower compared to the short-delay case. The long time delays require smaller values of K_{opt} because it takes longer time to transfer larger packets of data between the nodes, and selecting a high value for K will be "over-reactive." For example, for $K = 0.9$ the cluster behavior is unstable and small perturbations in the load balancing instant cause increase in the completion time. The behavior of the double-balancing case is summarized in Fig. 6. The top plot shows the dependence of the minimum t_{compl} as a function of the load-balancing strength parameter K for small delays, and the bottom plot shows the same dependency for the long-delay case.

IV. Conclusions

Our simulations indicate that with a double-load-balancing strategy, it is possible to achieve improved overall performance, measured by the completion time of the total tasks in the system, in comparison to the single-load-balancing strategy. In either case, a performance almost comparable to the continuous-load-balancing strategy can be achieved. The optimal selection of the load-balancing instants is shown to be in the beginning of the work process with the provision that the gain parameter should be selected more conservatively as the delay becomes more pronounced. However, if the delays are relatively small, it is possible to delay the execution of the load balancing until the information about the state of other nodes is collected. This "better informed" balancing will have the advantage of reduced sensitivity to errors in the selection of load-balancing instants.

Our future work will include the implementation and performance analysis of the load-balancing strategies proposed here on real systems. We have successfully implemented the balancing algorithm on three geographically distributed CEs located respectively in Spain, Argentina and the US. The results have been encouraging and a detailed study of the real system is underway. This work was supported by the National Science Foundation (under an Information Technology Research Grant).

V. REFERENCES

[1] Z. Lan, V. E. Taylor, and G. Bryan, "Dynamic load balancing for adaptive mesh re£nement application," in *Proc. ICPP'2001*, Valencia, Spain, 2001.

[2] T. L. Casavant and J. G. Kuhl, "A taxonomy of scheduling in general-purpose distributed computing systems," *IEEE Trans. Software Eng.*, vol. 14, pp. 141–154, Feb. 1988.

[3] G. Cybenko, "Dynamic load balancing for distributed memory multiprocessors," *IEEE Trans. Parallel and Distributed Computing*, vol. 7, pp. 279–301, Oct. 1989

[4] C-C. Hui and S. T. Chanson,"Hydrodynamic load balancing," *IEEE Trans. Parallel and Distributed Systems*, vol. 10, Issue 11, Nov. 1999.

[5] M. M. Hayat, S. Dhakal, C. T. Abdallah " Dynamic time delay models for load balancing. Part II: Stochastic analysis of the effect of delay uncertainty, *CNRS-NSF Workshop: Advances in Control of Time-Delay Systems*, Paris France, January 2003. Also to appear in an edited book by Springer, Keqin Gu and Silviu-Iulian Niculescu, Editors.

[6] J. D. Bridwell, J. Chisson, Z. Tang, T. Wang, C. T. Abdallah, and M. M. Hayat, "Dynamic time delay models for load balancing. Part I: Deterministic models," *CNRS-NSF workshop: Advances in Control of Time-Delay Systems*, Paris France, Jan. 2003. Also to appear in an edited book by Springer, K. Gu and S-I. Niculescu, Editors.

[7] 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*, to appear, 2003.

[8] D. J. Daley and D. Vere-Jones, *An introduction to the theory of point processes*. Springer-Verlag, 1988.