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

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

March 9, 2003

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

1 Introduction

The development of effective and computationally efficient 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 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-delays 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 significant source of time delay is the limitations imposed by the communication medium between 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 significantly 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 traffic and network configuration and architecture. Moreover, 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.

Thus, if we are designing a load-balancing policy under no delay or fixed-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 [1], [2]. 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, $K$, that dictates what portion of any CE’s excess load should be assigned to other CEs [3]. In the ideal case where the communication and load-transfer delays are negligible (as in an 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 wasted 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 [1] 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 the delay factors.

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 instances while optimizing the strength of the load balancing and the actual load-balancing instance 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.

## 2 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 [1], [3]. 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 $i$th node at time $t$. Assume that the $i$th node completes tasks according to a Poisson process and at a constant rate $\mu_i$. 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 $\lambda_i$ [4], that is, $J_i(t_1, t_2) = \sum_{k: t_1 < \xi_k \leq t_2} H_k$, where $\xi_k$ are arrival times of job requests arriving according to a Poisson process with rate $\lambda_i$. 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 $k$th job request. The load balancing mechanism is described as follows: The $i$th node, at a specific load-balancing instant $T_i^t$, looks at its own load $Q_i(T_i^t)$ 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 $i$th queue in a differential form as (in $\Delta t$ time increments):

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

where $C_i(t + \Delta t)$ is a Poisson process (with rate $\mu_i$) 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 at the same interval, $\tau_j(t)$ is the delay in transferring load from node $j$ to node $i$ at the same interval, and $L_{ij}(t)$ 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 $L_{kl}(t) = g_{kl}(Q_i(t), Q_k(t - \eta_j(t)), \ldots, Q_l(t - \eta_j(t)), \ldots)$, where for any $j \neq k$, $\eta_j(t) = \eta_j(t)$ is the communication delay between the $k$th and $j$th nodes at time $t$. The function $g_{kl}$ dictates the load-balancing policy between the $k$th and $l$th nodes. One common example is

$$g_{kl}(Q_i(t), Q_k(t - \eta_j(t)), \ldots, Q_l(t - \eta_j(t)), \ldots) = K_{kl} p_{kl} \cdot \left( Q_i(t) - n^{-1} \sum_{j=1}^{n} Q_j(t - \eta_j(t)) \right) \cdot u\left( Q_i(t) - n^{-1} \sum_{j=1}^{n} Q_j(t - \eta_j(t)) \right),$$

where $u(\cdot)$ is the unit step function with the obvious convention $\eta_j(t) = 0$, and $K_{kl}$ is a parameter that controls the “strength” or “gain” of load balancing at the $k$th (load distributing) node. In this example, the $i$th node simply compares its load to the average over all nodes and sends out a fraction $p_{ik}$ of its excess load to the $l$th node. (Of course, $\sum_{l=1}^{n} p_{il} = 1$.)
3 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 $\mu$s 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).

![Graph of Queue Length and Completion Time](image1.png)

**Figure 1:** Left: 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. Right: The empirical variance of the queue length normalized by the mean-square values.

Some of our earlier experimental results that motivated the present study are summarized in Fig. 1. The left graph in this figure 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 60ms. 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 right 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 30ms). We observed that even in the fastest completion period, 95% of the tasks were completed around 15ms 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 workload 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.

![Graph of Variance](image2.png)

**Figure 2:** Optimal single load-balancing scheduling for the short-delay case. Left: completion time vs. load-balancing instant, $t_{bal}$; Right: queue lengths and cumulative tasks completed by each node.
3.1 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 $t_{bal}$ is 0.02 ms, and the corresponding completion time $t_{compl}$ is 47.57 ms, as seen in Fig. 2 (left). Now from the right graph in Fig. 2, we can see that the queue lengths change abruptly as a result of load-balancing events associated with the three 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 represent 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 first 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 $t_{bal}$ 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 significantly 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 left plot in Fig. 3, the shortest completion time possible is approximately 52 ms for
Figure 5: Double load-balancing scheduling for the long-delay case. Left: $K = 0.5$, $t_{bal1} = 0.01\text{ms}$, $t_{bal2} = 0.02\text{ms}$; Right: queue length evolution and the cumulative tasks done by each node.

Figure 6: Double load-balancing scheduling showing the task completion time as a function of the load-balancing strength parameter $K$. Left: small-delay case; Right: large-delay case.

$t_{bal1} = 0.01\text{ms}$ 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 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 efficiency. 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 right plot in the same figure we can see that CE1 and CE2 complete their work 5ms 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 66.51ms at $K = 0.725$ and for $t_{bal1} = 0.63\text{ms}$. 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 65ms. 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.

3.2 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 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 0.8ms, and a similar average load-transfer delay, then the
best \( t_{\text{compl}} \) is found to be 43.15ms, which occurs when \( t_{\text{bal1}} = 0.01\text{ms} \) and \( t_{\text{bal2}} = 0.02\text{ms} \) 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 \( t_{\text{bal1}} = 3.87 \) and \( t_{\text{bal2}} = 3.88 \) yields one of the best completion times.

From the left 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 0.03ms-2ms is sensitive to error in the scheduling time. In particular, a small deviation of \( t_{\text{bal}} \) 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 \( t_{\text{bal1}} = 0.01\text{ms} \) 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 45ms. 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 \( K = 0.5 \), \( t_{\text{bal1}} = 0.01\text{ms} \) and \( t_{\text{bal2}} = 0.02\text{ms} \). While a completion time of approximately 16ms is slightly higher than that in the previous case, it is still close to its optimal value. We see from Fig. 5 (left) 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 left plot shows the dependence of the minimum \( t_{\text{compl}} \) as a function of the load-balancing strength parameter \( K \) for small delays, and the right plot shows the same dependency for the long-delay case.

4 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 proviso 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.

References


