# Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration-Theory Approach

Sagar Dhakal, Majeed M. Hayat, *Senior Member*, *IEEE*, Jorge E. Pezoa, Cundong Yang, and David A. Bader, *Senior Member*, *IEEE*

**Abstract**—A regeneration-theory approach is undertaken to analytically characterize the average overall completion time in a distributed system. The approach considers the heterogeneity in the processing rates of the nodes as well as the randomness in the delays imposed by the communication medium. The optimal one-shot load balancing policy is developed and subsequently extended to develop an autonomous and distributed load-balancing policy that can dynamically reallocate incoming external loads at each node. This adaptive and dynamic load balancing policy is implemented and evaluated in a two-node distributed system. The performance of the proposed dynamic load-balancing policy is compared to that of static policies as well as existing dynamic load-balancing policies by considering the average completion time per task and the system processing rate in the presence of random arrivals of the external loads.

**Index Terms**—Renewal theory, queuing theory, distributed computing, dynamic load balancing.

## 1 Introduction

The computing power of any distributed system can be realized by allowing its constituent computational elements (CEs), or nodes, to work cooperatively so that large loads are allocated among them in a fair and effective manner. Any strategy for load distribution among CEs is called load balancing (LB). An effective LB policy ensures optimal use of the distributed resources whereby no CE remains in an idle state while any other CE is being utilized.

In many of today's distributed-computing environments, the CEs are linked by a delay-limited and bandwidth-limited communication medium that inherently inflicts tangible delays on internode communications and load exchange. Examples include distributed systems over wireless local-area networks (WLANs) as well as clusters of geographically distant CEs connected over the Internet, such as PlanetLab [1]. Although the majority of LB policies developed heretofore take account of such time delays [2], [3], [4], [5], [6], they are predicated on the assumption that delays are deterministic. In actuality, delays are random in such communication media, especially in the case of WLANs. This is attributable to uncertainties associated with the amount of traffic, congestion, and other unpredictable factors within the network. Furthermore, unknown characteristics (e.g., type of application and load size) of the incoming loads cause the CEs to exhibit fluctuations in runtime processing speeds. Earlier work by our group has shown that LB policies that do not account for the delay randomness may perform poorly in practical distributed-computing settings where random delays are present [7]. For example, if nodes have dated, inaccurate information about the state of other nodes, due to random communication delays between nodes, then this could result in unnecessary periodic exchange of loads among them. Consequently, certain nodes may become idle while loads are in transit, a condition that would result in prolonging the total completion time of a load.

Generally, the performance of LB in delay-infested environments depends upon the selection of balancing instants as well as the level of load-exchange allowed between nodes. For example, if the network delay is negligible within the context of a certain application, the best performance is achieved by allowing every node to send all its excess load (e.g., relative to the average load per node in the system) to less-occupied nodes. On the other hand, in the extreme case for which the network delays are excessively large, it would be more prudent to reduce the amount of load exchange so as to avoid time wasted while loads are in transit. Clearly, in a practical delay-limited distributed-computing setting, the amount of load to be exchanged lies between these two extremes and the amount of load-transfer has to be carefully chosen. A commonly used parameter that serves to control the intensity of load balancing is the LB gain.

In our earlier work [7], [8], we have shown that, for distributed systems with realistic random communication delays, limiting the number of balancing instants and optimizing the performance over the choice of the balancing times as well as the LB gain at each balancing instant can result in significant improvement in computing efficiency. This motivated us to look into the so-called one-shot LB strategy. In particular, once nodes are initially assigned a certain number of tasks, all nodes would together execute LB only at one prescribed instant [8]. Monte Carlo studies and real-time experiments conducted over WLAN confirmed our notion that, for a given initial load and average

- *S. Dhakal, M.M. Hayat, J.E. Pezoa, and C. Yang are with the Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM 87131-0001. E-mail: {dhakal, hayat, jpezoa, cundongyang}@eece.unm.edu.*
- *D.A. Bader is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332. E-mail: bader@cc.gatech.edu.*

*Manuscript received 17 Dec. 2005; revised 27 June 2006; accepted 6 July 2006; published online 9 Jan. 2007.*
*Recommended for acceptance by R. Thakur.*
*For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0508-1205.*
*Digital Object Identifier no. 10.1109/TPDS.2007.1007.*

processing rates, there exist an optimal LB gain and an optimal balancing instant associated with the one-shot LB policy, which together minimize the average overall completion time. This has also been verified analytically through our regeneration-theory-based mathematical model [9]. However, this analysis has been limited to only two nodes and has focused on handling an initial load without considering subsequent arrivals of loads.

In practice, external loads of different size (possibly corresponding to different applications) arrive at a distributed-computing system randomly in time and node space. Clearly, scheduling has to be done repeatedly to maintain load balance in the system. Centralized LB schemes [10], [11] store global information at one location and a designated processor initiates LB cycles. The drawback of this scheme is that the LB is paralyzed if the particular node that controls LB fails. Such centralized schemes also require synchronization among nodes. In contrast, in a distributed LB scheme, every node executes balancing autonomously. Moreover, the LB policy can be static or dynamic [2], [12]. In a static LB policy, the scheduling decisions are predetermined, while, in a dynamic load-balancing (DLB) policy, the scheduling decisions are made at runtime. Thus, a DLB policy can be made adaptive to changes in system parameters, such as the traffic in the channel and the unknown characteristics of the incoming loads. Additionally, DLB can be performed based on either local information (pertaining to neighboring nodes) [13], [14] or global information, where complete knowledge of the entire distributed system is needed before an LB action is executed.

Due to the emergence of heterogeneous computing systems over WLANs or the Internet, there is presently a need for distributed DLB policies designed by considering the randomness in delays and processing speeds of the nodes. To date, a robust policy suited to delay-infested distributed systems is not available, to the best of our knowledge [3]. In this paper, we propose a sender-initiated distributed DLB policy where each node autonomously executes LB at every external load arrival at that node. The DLB policy utilizes the optimal one-shot LB strategy each time an LB episode is conducted, and it does not require synchronization among nodes. Every time an external load arrives at a node, only the receiver node executes a locally optimal one-shot-LB action, which aims to minimize the average overall completion time. This requires the generalization of the regeneration-theory-based queuing model for the centralized one-shot LB [9]. Furthermore, every LB action utilizes current system information that is updated during runtime. Therefore, the DLB policy adapts to the dynamic environment of the distributed system.

This paper is organized as follows: Section 2 contains the general description of the LB model in a delay-limited environment. In Section 3, we present the regeneration-based stochastic analysis of the optimal multinode one-shot LB policy and develop the proposed DLB policy. Experimental results as well as analytical predictions and Monte Carlo (MC) simulations are presented in Section 5. Finally, our conclusions are given in Section 6.

## 2 Preliminaries

To introduce the basic LB model, we present a review of the queuing model that characterizes the stochastic dynamics of the LB problem, as detailed in [7]. Consider a distributed system of $n$ nodes, where all nodes can communicate with each other. If $Q_i(t)$ is the queue length of the $i$th node at time $t$, then, after time $\Delta t$, the queue length increases due to the arrival of external tasks, $J_i(t, t+\Delta t)$, as well as the arrival of tasks that have been allocated to node $i$ by other nodes as a result of LB. Moreover, in the interval $[t, t+\Delta t]$, the queue $Q_i(t)$ decreases according to the number of tasks serviced by it, which we denote by $C_i(t, t+\Delta t)$. In addition, node $i$ may send a number of tasks to the other nodes in the system in the same time interval. With these dynamics, the queue length of node $i$ can be cast in differential form as

$$\begin{aligned} Q_i(t+\Delta t) = {} & Q_i(t) - C_i(t, t+\Delta t) + J_i(t, t+\Delta t) \\ & - \sum_{j\neq i} \sum_{l} L_{ji}(t) I_{\{t_l^i = t\}} \\ & + \sum_{j\neq i} \sum_{k} L_{ij}(t-\tau_{ij,k}) I_{\{t_k^j = t-\tau_{ij,k}\}}, \end{aligned} \tag{1}$$

where $\{t_k^i\}_{k=1}^{\infty}$ is a sequence of LB instants for the $i$th node, $C_i(t, t+\Delta t)$ is a Poisson process (with rate $\lambda_{d_i}$) describing the random number of tasks completed in the interval $[t, t+\Delta t)$, and $\tau_{ij,k}$ is the delay in transferring a random load $L_{ij}(t-\tau_{ij,k})$ from node $j$ to node $i$ at the $k$th LB instant of node $j$, and $I_A$ is an indicator function for the event $A$.

### 2.1 Methods for Allocating Loads in Load Balancing

At time $t$, a node ($j$, say) computes its excess load by comparing its local load to the average overall load of the system. More precisely, the excess load, $L_j^{ex}(t)$, is random and is given by

$$L_j^{ex}(t) = \Big( Q_j(t) - \frac{\lambda_{d_j}}{\sum_{k=1}^{n} \lambda_{d_k}} \sum_{l=1}^{n} Q_l(t-\eta_{jl}) \Big)^{+}, \tag{2}$$

where $\eta_{jl}$ is the communication delay from the $l$th to the $j$th node (with the convention $\eta_{ll} = 0$), and $(x)^{+} \triangleq \max(x, 0)$. Note that the second quantity inside the parentheses in (2) is simply the fair share of node $j$ from the totality of the loads in the system. Also, we assume that $Q_l(t-\eta_{jl}) = 0$ if $t < \eta_{jl}$, implying that node $j$ assumes that node $l$ has zero queue size whenever the communication delay is bigger than $t$. This is a more plausible way to calculate the excess load of a node in a heterogeneous computing environment as compared to earlier methods that did not consider the processing speed of the nodes [7], [8], [9]. With the inclusion of the processing speed of the nodes in (2), a slower node would have a larger excess load than that of a faster node. Moreover, the excess load has to be partitioned among the $n-1$ nodes by assigning a larger portion to a node with smaller relative load. To this end, we introduce two different approaches to calculate the partitions, denoted by $p_{ij}$, which represent the fraction of the excess load of node $j$ to be sent to node $i$. Any such partition should satisfy $\sum_{l=1}^{n} p_{lj} = 1$, where $p_{jj} = 0$ by definition.

The fractions $p_{ij}$ for $i \neq j$, can be chosen as

$$p_{ij} = \begin{cases} \frac{1}{n-2} \Big( 1 - \frac{\lambda_{d_i}^{-1} Q_i(t-\eta_{ji})}{\sum_{l\neq j} \lambda_{d_l}^{-1} Q_l(t-\eta_{jl})} \Big), & \sum_{l\neq j} Q_l(t-\eta_{jl}) > 0 \\ \lambda_{d_i} / \sum_{k\neq j} \lambda_{d_k}, & \text{otherwise}, \end{cases} \tag{3}$$

where $n \geq 3$. Clearly, a node assigns a larger partition of its excess load to a node with a small load relative to all other candidate recipient nodes. Indeed, it is easy to check that $\sum_{l=1}^{n} p_{lj} = 1$. For the special case when $n = 2$, $p_{ij} = 1$

whenever $i \neq j$. But observe that $p_{ij} \leq \frac{1}{n-2}$ for any node $i$. This means that the maximum size of the partition decreases as the number of nodes in the system increases, irrespective of the processing rates of the nodes. Therefore, this partition may not be effective in a scenario where some nodes may have very high processing rates as compared to most of the nodes in the system. This observation prompted us to consider a second partition, which is described below.

In the second approach, the sender node locally calculates the excess load for each node in the system and calculates the portions to be transferred accordingly. For convenience, define $m_{i(j)}(t) \triangleq Q_i(t - \eta_{ji})$ and let $L^{ex}_{i(j)}(t)$ be the excess load at node $i$, as calculated by node $j$. Then, by using a rationale similar to that used in (2), we obtain the locally computed excess load

$$L^{ex}_{i(j)}(t) \triangleq m_{i(j)}(t) - \frac{\lambda_{d_i}}{\sum_{k=1}^{n} \lambda_{d_k}} \sum_{l=1}^{n} m_{l(j)}(t). \tag{4}$$

It is straightforward to verify that $\sum_{i=1}^{n} L^{ex}_{i(j)}(t) = 0$ almost surely. The idea here is that node $j$ may transfer loads only to those nodes that are below the average load of the system. Therefore, the partition $p_{ij}$ can be defined as

$$p_{ij} = \begin{cases} L^{ex}_{i(j)}(t) / \sum_{l \in \mathcal{I}_j} L^{ex}_{l(j)}(t), & i \in \mathcal{I}_j \\ 0, & \text{otherwise}, \end{cases} \tag{5}$$

where

$$\mathcal{I}_j \triangleq \{i : L^{ex}_{i(j)}(t) < 0\}.$$

The above partition is most effective when delays are negligible, $m_{i(j)}(t)$ are deterministic, and tasks are arbitrarily divisible. In this case, if LB is executed together by all the nodes that do not belong to $\mathcal{I}_j$, each node finishes its tasks together, thereby minimizing the overall completion time. The proof of optimality of this partition is shown in Appendix A.

When delays are present, the partitions defined by (3) or (5) may not be effective in general, and the proportions $p_{ij}$ must be adjusted. To incorporate this adjustment, the *adjusted* load to be transferred to node $i$ must be defined as

$$L_{ij}(t) = \lfloor K_{ij} p_{ij} L^{ex}_j(t) \rfloor, \tag{6}$$

where $\lfloor x \rfloor$ is the greatest integer less than or equal to $x$, and the parameters $K_{ij} \in [0, 1]$ constitute the *user-specified LB gains*. To summarize, the $j$th node first compares its load to the average overall load of the system, then partitions its excess load among $n-1$ available nodes using the fractions $K_{ij}p_{ij}$, and dispatches the integral parts of the adjusted excess loads to other nodes.

## 3 Theory and Optimization of Load Balancing

In this section, we characterize the expected value of the overall completion time for a given initial load under the centralized one-shot LB policy for an arbitrary number of nodes. The overall completion time is defined as the maximum over completion times for all nodes. We use the theory to optimize the selection of the LB instant and the LB gain. A distributed and adaptive version of the one-shot is also developed and used to propose a sender-initiated DLB policy. Throughout the paper, a *task* is the smallest (indivisible) unit of load and *load* is a collection of tasks.

### 3.1 Centralized One-Shot Load Balancing

The centralized one-shot LB policy is a special case of the model described in (1) with only one LB instant permitted (i.e., $t^i_l = \infty$, for any $i$ and any $l \geq 2$) and no task arrival is permitted beyond the initial load $(J_i(t, t + \Delta t) = 0, t > 0)$. The objective is to calculate the optimal values for the LB instant $t_b$ and LB gains $K_{ij}$ to minimize the average overall completion time (AOCT). We assume that each node broadcasts its queue size at time $t = 0$ and, for the moment, we will assume that all nodes execute LB together at time $t_b$ with a common gain $K_{ij} = K$. This latter assumption is relaxed in Section 3.2 to a setting where nodes execute LB autonomously.

#### 3.1.1 The Notion of Knowledge State

We begin with our formal definition of the *knowledge state* of the distributed system. In a system of $n$ nodes, each node receives $n-1$ communications, each of which carries queue-size information of the respective nodes. Depending upon the choice of the balancing instant $t_b$ and the realizations of the random communication delays, any node may or may not receive a communication by the time LB takes place. For each node $j$, we assign a binary vector $\mathbf{i}_j$ of size $n$ that describes the knowledge state of the node. A "1" entry for the $k$th component $(k \neq j)$ of $\mathbf{i}_j$ indicates that node $j$ has already received the communication from node $k$. By definition, the $j$th component of $\mathbf{i}_j$ is always "1." Clearly, at $t = 0$, all the entries of $\mathbf{i}_j$ are set to 0, with the exception of the $j$th entry, which is "1." The *system knowledge state* is the concatenated vector $\mathbf{I} = (\mathbf{i}_1, \ldots, \mathbf{i}_n)$. For example, in a three-node distributed system $(n = 3)$, state $\mathbf{I} = (100, 011, 111)$ corresponds to the configuration for which node 1 has no knowledge of nodes 2 and 3 (i.e., $\mathbf{i}_1 = (100)$), while node 2 has knowledge of node 3 $(\mathbf{i}_2 = (011))$, and node 3 has knowledge of both nodes 1 and 2 $(\mathbf{i}_3 = (111))$. Clearly, a total of $n \times (n-1)$ binary bits ($n-1$ bits per node) are needed to describe all possible $\mathbf{I}$. An all-ones $\mathbf{I}$ (all-zeros $\mathbf{I}$) refers to the so-called *informed knowledge state* (*null knowledge state*). Any other $\mathbf{I}$ is said to be hybrid. Intuitively, the LB resulting from an informed state should perform best; this is verified in Section 5.

#### 3.1.2 Regenerative Equations

The concept of regeneration[1] has proved to be a powerful tool in the analysis of complex stochastic systems [15], [16], [17]. The idea of our approach is to define a certain special random variable, called the *regeneration time*, $\tau$, defined as *the time to the first completion of a task by any node or the first arrival of a communication, whichever comes first*. The key feature of the event $\{\tau = s\}$ is that its occurrence will regenerate queues at time $s$ that have similar statistical properties and dynamics as their predecessors, but possibly with different initial configurations, viz., different initial

1. Consider a game where a gambler starts with fortune $x \in \{0, 1, 2, \ldots, 20\}$ dollars and bids a dollar at every hand, either winning or losing a dollar. The game is over if he hits 0 or 20 dollars. Given the outcome of first bidding, the process of regeneration can be seen as follows: If the gambler wins (loses), the game starts again with $x+1$ dollars ($x-1$ dollars). Therefore, at every bidding, the same game regenerates itself, but with a different initial condition.

load distribution if the initial event is a task completion or a different knowledge state if the initial event is an arrival of communication. We use the notions described above to derive integral equations describing the expected time of load completion under a predefined LB policy of Section 2.

Consider an $n$-node distributed computing system and suppose that the service time (execution time for one task) of the $i$th node follows exponential distribution with parameter (inverse of the mean) $\lambda_{d_i}$. Although somewhat restrictive, this is a meaningful assumption in order to obtain an analytically tractable result. The communication delays between the nodes, say the $i$th node and the $j$th node, are also assumed to follow an exponential distribution with rates $\lambda_{ij}$. Let $W_i$ and $X_{ij}$ be the random variables representing the time of the first task completion at the $i$th node and the time of arrival of communication from node $j$ to node $i$, respectively. Note that the regeneration random variable can now be written as

$$\tau = \min(\min_i(W_i), \min_{j\neq i}(X_{ij})).$$

From basic probability, $\tau$ is also an exponential random variable with rate $\lambda = \sum_{i=1}^{n}(\lambda_{d_i} + \sum_{j\neq i}\lambda_{ij})$.

To see how the idea of regeneration works, consider the example for which the initial event occurs at time $s$ happens to be the execution of a task at node 1. This corresponds to the occurrence of the event $\{\tau = s, \tau = W_1\}$. In this case, queue dynamics remains unchanged except that node 1 will now have one task less from its initial load. Thus, upon the occurrence of this particular realization of the initial event, the queues will reemerge at time $s$ with a different initial load. A similar behavior is observed if the initial event is the arrival of a communication from node 2 to node 1 or, equivalently, when the event $\{\tau = s, \tau = X_{12}\}$ occurs. In this case, the newly emerged queues will have a new knowledge state, where the second component of $\mathbf{i}_1$ is set to "1."

Let $T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b)$ be the overall completion time given that the balancing is executed at time $t_b$, where the $i$th node has $m_i \geq 0$ tasks at time $t = 0$ and the system knowledge state is $\mathbf{I}$ at time $t = 0$. Exploiting the properties of conditional expectation, we can write the AOCT as

$$\begin{aligned}\mathrm{E}[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b)] &= \mathrm{E}\Big[\mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) \mid \tau\Big]\Big] \\ &= \int_0^\infty \mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) \mid \tau = s\Big] f_\tau(s)ds,\end{aligned} \tag{7}$$

where $f_\tau(t)$ is the probability density function (pdf) of $\tau$. Splitting the above integral, we get

$$\begin{aligned}\mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b)\Big] = &\int_0^{t_b} \mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) \mid \tau = s\Big] f_\tau(s)ds \\ &+ \int_{t_b}^\infty \mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) \mid \tau = s\Big] f_\tau(s)ds.\end{aligned} \tag{8}$$

For $s > t_b$, the occurrence of the event $\{\tau = s\}$ implies that no change occurred in initial configuration of the queues until $t_b$. So, conditional on the occurrence of $\{\tau = s\}$ with $s > t_b$, we can imagine new queues emerging independently at $t_b$, which are identically distributed to the queues that originally emerged at time 0. Therefore,

$$\mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) \mid \tau = s\Big] = t_b + \mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(0)\Big]$$

as long as $s > t_b$.

On the other hand, for $s \leq t_b$, we have

$$\begin{aligned}&\mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) \mid \tau = s\Big] = \\ &\sum_{i=1}^{n}\sum_{j\neq i}\mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) \mid \tau = s, \tau = X_{ij}\Big]\mathrm{P}\Big\{\tau = X_{ij} \mid \tau = s\Big\} \\ &+ \sum_{i=1}^{n}\mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) \mid \tau = s, \tau = W_i\Big]\mathrm{P}\Big\{\tau = W_i \mid \tau = s\Big\}.\end{aligned}$$

Suppose that, for $s \leq t_b$, the event $\{\tau = s, \tau = W_i\}$ occurs. In this case, we can think of new queues emerging at time $s$, independently of the original queues, which have the same statistics as the original queues, had node $i$ in the original queue had $m_i - 1$ tasks instead of $m_i$ tasks. Thus, the queue has reemerged, or regenerated itself, with a different initial load and, therefore,

$$\begin{aligned}&\mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) \mid \tau = s, \tau = W_i\Big] = \\ &\qquad s + \mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_i-1\ldots,m_n}(t_b - s)\Big].\end{aligned}$$

Similarly, if $\{\tau = s, \tau = X_{ij}\}$ occurs, we obtain

$$\mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) \mid \tau = s, \tau = X_{ij}\Big] = s + \mathrm{E}\Big[T^{\mathbf{I}^{ij}}_{m_1,\ldots,m_n}(t_b - s)\Big],$$

where $\mathbf{I}^{ij}$ is identical to $\mathbf{I}$ with the exception that the $j$th component of $\mathbf{i}_i$ is 1.

Let $\mu^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) := \mathrm{E}\Big[T^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b)\Big]$. In light of the regeneration-event decomposition and the conditional expectations described above, the quantities $\mu^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b)$ can be characterized by the following set of $2^{n(n-1)}$ (one for each initial knowledge state $\mathbf{I}$) integro-difference equations:

$$\begin{aligned}\mu^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) = &\int_{t_b}^\infty \Big(\mu^{\mathbf{I}}_{m_1,\ldots,m_n}(0) + t_b\Big) f_\tau(s)\,ds \\ &+ \int_0^{t_b}\Bigg(\sum_{i=1}^{n}\Big(s + \mu^{\mathbf{I}}_{m_1-\delta_{1,i},\ldots,m_n-\delta_{n,i}}(t_b - s)\Big)\mathrm{P}\Big\{\tau = W_i \mid \tau = s\Big\} \\ &+ \sum_{i=1}^{n}\sum_{j\neq i}\Big(s + \mu^{\mathbf{I}^{ij}}_{m_1,\ldots,m_n}(t_b - s)\Big)\mathrm{P}\Big\{\tau = X_{ij} | \tau = s\Big\}\Bigg) \\ &\times f_\tau(s)\,ds.\end{aligned} \tag{9}$$

Here, $\delta_{j,i} = 1$ is the Kronecker delta. By direct differentiation of (9), we obtain

$$\begin{aligned}\frac{d\mu^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b)}{dt_b} = &\sum_{i=1}^{n}\lambda_{d_i}\mu^{\mathbf{I}}_{m_1-\delta_{1,i},\ldots,m_n-\delta_{n,i}}(t_b) \\ &+ \sum_{i=1}^{n}\sum_{j\neq i}\lambda_{ij}\mu^{\mathbf{I}^{ij}}_{m_1,\ldots,m_n}(t_b) - \lambda\mu^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) + 1.\end{aligned} \tag{10}$$

Each of these equations involves a recursion in the variable appearing in the subscripts and superscripts of $\mu^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b)$, which has been exploited to solve them by writing an efficient code. We also point out that, while solving each of these equations, we need to solve for its corresponding initial conditions, namely, $\mu^{\mathbf{I}}_{m_1,\ldots,m_n}(0)$. For simplicity, we will provide explicit solution of (10) to compute the optimal LB gains and the optimal LB instant

for $n = 2$. Nonetheless, this will demonstrate the fundamental technique to calculate the initial condition for a multinode system.

#### 3.1.3 Special Case: $n = 2$

In this case, (10) yields four equations involving $\mu_{m_1,m_2}^{(1k_1,k_21)}(t_b)$ for $k_i \in \{0,1\}$. In [9], a brute-force method (based on conditional probabilities) was used to calculate $\mu_{m_1,m_2}^{(1k_1,k_21)}(0)$. Now, we solve this more efficiently using the concept of regeneration. Without loss of generality, suppose $m_1 > m_2$. Using (2) and (6), and with $p_{21} = 1$,

$$L_{21}(0) = \begin{cases} \left\lfloor \frac{K(\lambda_{d_2} m_1 - \lambda_{d_1} m_2)}{\lambda_{d_1} + \lambda_{d_2}} \right\rfloor & \text{if } (k_1,k_2) \in \{(1,0),(1,1)\} \\ \left\lfloor \frac{K\lambda_{d_2} m_1}{\lambda_{d_1} + \lambda_{d_2}} \right\rfloor, & \text{otherwise.} \end{cases} \tag{11}$$

$L_{12}(0)$ can be calculated similarly. For convenience, we define $L_{21} := L_{21}(0)$ and $L_{12} := L_{12}(0)$. The delay in transferring load $L_{ij}$ is termed as load-transfer delay from the $j$th to the $i$th node. The load-transfer delay is assumed to follow an exponential pdf with rate $\lambda_{t_{ij}}$, which is a function of $L_{ij}$ (see Section 5.1). Suppose $T_1$ is the waiting time at node 1 before all the tasks (including that sent from node 2) are served. Let the cumulative distribution function (cdf) of $T_1$ be denoted as $F_{T_1}(r_1; L_{12}; t)$, where $r_1$ is the number of tasks at node 1 just after LB is performed at time $t = 0$, i.e., $r_1 = m_1 - L_{21}$, and $L_{12}$ is the number of tasks in transit. Applying the regeneration principle (for details, refer to Appendix B), we obtain

$$\begin{aligned} \frac{dF_{T_1}(r_1; L_{12}; t)}{dt} = \\ -(\lambda_{d_1} + \lambda_{t_{12}}) F_{T_1}(r_1; L_{12}; t) + \lambda_{d_1} F_{T_1}(r_1 - 1; L_{12}; t) \\ + \lambda_{t_{12}} F_{T_1}(r_1 + L_{12}; 0; t). \end{aligned} \tag{12}$$

The initial conditions $F_{T_1}(0; L_{12}; t)$ and $F_{T_1}(r_1 + L_{12}; t)$ can be further decomposed into simpler recursive equations by invoking the regeneration theory once again. For simplicity of notation, let $F_{T_1}(t) := F_{T_1}(r_1; L_{12}; t)$. We can also calculate $F_{T_2}(t)$ using similar recursive differential equations. Now, the overall completion time is $T_C = \max(T_1, T_2)$ and recall that its average $\mathrm{E}[T_C]$ is $\mu_{m_1,m_2}^{(1k_1,k_21)}(0)$. By exploiting the independence of $T_1$ and $T_2$, we obtain the explicit solution

$$\begin{aligned} \mu_{m_1,m_2}^{(1k_1,k_21)}(0) &= \mathrm{E}[\max(T_1, T_2)] \\ &= \int_0^\infty t[f_{T_1}(t) F_{T_2}(t) + F_{T_1}(t) f_{T_2}(t)]\, dt, \end{aligned} \tag{13}$$

where $f_{T_1}(t)$ and $f_{T_2}(t)$ are the pdfs of $T_1$ and $T_2$, respectively.

### 3.2 A Policy for Dynamic Load Balancing

In this section, we modify the centralized one-shot LB strategy to a distributed, adaptive setting and use it to develop a sender-initiated DLB policy. The distributed one-shot LB policy is different from the centralized one-shot LB policy described in Section 3.1 in two ways: 1) It adapts to varying system parameters such as load variability, randomness in the channel delay, and variable runtime processing speed of the nodes, and 2) the LB is performed in an autonomous fashion, that is, each node selects its own optimal LB instant and gain. (Recall that, according to the centralized one-shot LB policy described in Section 3.1, after the initial load assignment to nodes, all the nodes execute LB synchronously using a common LB instant and gain.) Each time an external load arrives at a node, the node seeks an optimal one-shot LB action that minimizes the load-completion time of the entire system, based on its present load, its knowledge of the loads of other nodes, and its knowledge of the system parameters at that time. For clarity, we use the term *external load* to represent the loads submitted to the system from some external source and not the loads transferred from other nodes due to LB. We will assume external load arrivals of random sizes. Each time an external load is assumed to arrive randomly at any of the nodes, independently of the arrivals of other external loads to it and other nodes.

Consider a system of $n$ distributed nodes with a given initial load and assume that external loads arrive randomly thereafter. We assume that nodes communicate with each other at so-called "sync instants" on a regular basis. Upon the arrival of each batch of external loads, the receiving node and only the receiving node prompts itself to execute an optimal distributed one-shot LB. Namely, it finds the optimal LB instant and gain and executes an LB action accordingly. Since load balancing is performed locally at the external-load-receiving node, say, node $j$, the policy depends only on its knowledge state vector $\mathbf{i}_j$, rather than the system knowledge state $\mathbf{I}$. Consequently, the number of possible knowledge states become $2^{(n-1)}$. Further, considering the periodic sync-exchanges between nodes, each node in the system is continually assumed to be informed of the states of other nodes. Hence, the only possible choice for the knowledge state vector of each node $j$ is $\mathbf{i}_j = (1 \cdots 1) \equiv \mathbf{1}$, leading to a simpler optimization problem than the one detailed earlier.

Suppose that an external arrival occurs at node $j$ at time $t = t_a$. We need to compute the optimal LB gain and optimal LB instant for node $j$ based on knowledge-state vector $\mathbf{1}$. Clearly, according to the knowledge of node $j$ at time $t_a$, the effective queue length of node $k$ is $m_{k(j)}(t_a)$. To recall, $m_{k(j)}(t_a) = Q_k(t_a - \eta_{jk^*})$, where $\eta_{jk^*}$ refers to the delay in the most recent communication received by node $j$ from node $k$. The goal is to minimize $\mu^{\mathbf{1}}_{m_{1(j)},\ldots,m_{n(j)}}(t_a + t_b)$, where $t_b$ is the LB instant of node $j$ measured from the time of arrival $t_a$. By setting $t_a = 0$, the system of queues, in the context of node $j$, at time $t_a$ becomes statistically equivalent to the system of queues at time 0 with initial load distribution $m_{k(j)}$ for all $k \in \{1,\ldots,n\}$. Therefore, we utilize the regeneration theory to obtain the following difference-differential equation that can be solved to calculate the optimal LB instant and the optimal LB gain.

$$\begin{aligned} \frac{d\mu^{\mathbf{1}}_{m_{1(j)},\ldots,m_{n(j)}}(t_b)}{dt_b} = \sum_{k=1}^{n} \lambda_{d_k} \mu^{\mathbf{1}}_{m_{1(j)}-\delta_{1,k},\ldots,m_{n(j)}-\delta_{n,k}}(t_b) \\ - \lambda \mu^{\mathbf{1}}_{m_{1(j)},\ldots,m_{n(j)}}(t_b) + 1, \end{aligned} \tag{14}$$

where $\lambda = \sum_{k=1}^{n} \lambda_{d_k}$. In addition, the optimization over $t_b$ becomes unnecessary since node $j$ is already in the informed knowledge state $\mathbf{1}$. This claim will be verified in Section 5.1, where the theoretical and experimental results show that a node should perform LB immediately after it gets informed. It simplifies our analysis as we can now set

$t_b = 0$ and the LB gains that minimize $\mu^{\mathbf{1}}_{m_{1(j)},\ldots,m_{n(j)}}(0)$ can be computed using difference equations. Therefore, in practice, the optimal LB gains are calculated online by the receiver node $j$ and LB is performed instantly at time $t_a$.

The initial condition $\mu^{\mathbf{1}}_{m_{1(j)},\ldots,m_{n(j)}}(0)$ can also be solved based on similar techniques that were used to obtain (13). But one notable difference here is that the local LB action taken by node $j$ at time 0 (measured from $t_a$) does not consider future load arrivals at node $j$ due to past or future LB actions at other nodes. In general, $L_{kj}(0)$, for all $k \neq j$, are calculated based on (2), (5), and (6), while setting $L_{jk}(0) = 0$ for all $k$. Therefore, we would expect to obtain a different solution for locally optimal $K$ than the one provided by (10).

The system parameter, namely, the average processing time per task $\lambda_{d_i}^{-1}$, is updated locally by each node $i$. At every sync instant, the node broadcasts its current processing rate and the current queue size. The added overhead in transferring and processing the knowledge state information grows in proportion to the arrival rates since the sync periods are adjusted according to the arrival rates. The second adaptive parameter is the mean transfer delay per task $\theta_{ji}$, which is updated by

$$\theta_{ji}^{(k)} = \alpha\left(\frac{\tau_{ji,k}}{L_{ji,k}}\right) + (1-\alpha)\theta_{ji}^{(k-1)}, \tag{15}$$

where $\tau_{ji,k}$ is the actual delay incurred in sending $L_{ji,k}$ tasks to node $j$ at the $k$th successful transmission of node $i$ and $\alpha \in [0,1]$ is the so-called "forgetting factor" of the previous estimation [18]. Also, $\theta_{ji}^{(0)}$ is calculated empirically from many experimental realizations of delays in transferring tasks from node $i$ to node $j$. The forgetting factor can be adjusted dynamically in order to accommodate drastic changes in transfer delay per task. Steps for the DLB policy are described in Appendix C.

## 4 Distributed Computing System Architecture

The LB policy has been implemented on a distributed computing system to experimentally determine its performance. The system consists of CEs that are processing jobs in a cooperative environment. The software architecture of the distributed system is divided in three layers: application, load-balancing, and communication. The application used to illustrate the LB process is matrix multiplication, where the processing of one task is defined as the multiplication of one row by a static matrix duplicated on all nodes. To achieve variability in the processing speed of the nodes, the randomness is introduced in the size of each task (row) by independently choosing its arithmetic precision with an exponential distribution. In addition, the application layer needs to update the queue size information of each node. The LB policy is implemented at the load-balancing layer with a software using a multithreaded process, where the POSIX-threads programming standard is used. One of the threads schedules and triggers the LB instants at predefined or calculated amount of times. In our implementation, when an external load arrives at a node that is transferring load, the required LB action is delayed until the node completes the transfer. The communication layer of each node handles the transfer of data from (to) that node to (from) the other nodes within the system. Each node uses the UDP transport protocol to transfer its current state information to the other nodes, while the TCP transport protocol is used to transfer the application data (tasks) between the CEs.

## 5 Results

We present the theoretical, MC simulation, and experimental results on the LB policies applied to the matrix multiplication performed on a distributed system comprising two nodes that are connected over 1) the Internet and 2) the UNM EECE infrastructure-based IEEE 802.11b WLAN. Over the Internet, we employed a 650 MHz Intel Pentium III processor-based computer (node 1) and a 2.66 GHz Intel P4 processor-based computer (node 2). For the WLAN setup, node 1 was replaced with a 1 GHz Transmeta Crusoe processor-based computer.

At first, experiments were performed to estimate the system parameters, namely, the processing speed of the nodes ($\lambda_{d_i}$), the communication rate ($\lambda_{ij}$), and the load-transfer rate per task ($\lambda_{t_{ij}}$). In Fig. 1, we show the empirical pdfs for the communication delay over the Internet as well as the WLAN, each of which can be approximated with an exponential pdf. In the experiments, each information packet had a fixed size of 30 Bytes. In Fig. 2a, we see that the average transfer delay grows linearly with the increase in number of tasks. Further, in Fig. 2b, the transfer delay per task can also be approximated as an exponential random variable. These empirical results are in agreement with the assumptions made in Section 3.

### 5.1 Centralized One-Shot LB Policy

In the experiments conducted over the Internet, node 1 and node 2 were initially assigned 100 and 60 tasks, respectively, where each task had a mean size of 120 Bytes. In this context, the processing rates per task of node 1 and node 2 were found to be 0.69 and 1.85, respectively. First, fixing the LB gain at $K = 1$, we optimized the AOCT by triggering the LB action at different instants. The analytical and experimental results of this optimization are shown in Fig. 3a. The experimental results are plotted by taking the AOCTs obtained from 20 experiments for each $t_b$. It can be seen that the AOCT becomes small after $t_b = 1$s. This behavior is attributed to the communication delay imposed by the channel. The empirically calculated average communication delay from node 1 to node 2 was 0.7 s, and from node 2 to node 1 was 0.9 s. Therefore, any LB action performed before 0.7 s is blind in the sense that there is no knowledge of the initial load of the other node; both nodes exchange tasks in this case. This behavior is evident from the experimental results shown in Fig. 3b, which depicts the mean number of tasks transferred as a function of $t_b$. Further, when LB action is taken between 0.7s and 0.9s, then node 1 will most likely have knowledge of node 2, while node 2 would not have knowledge of node 1. Consequently, according to (6), node 1 sends a smaller portion of its load to node 2 while node 2 still sends the same amount of load to node 1. This means that the slower node (node 1) would eventually execute more tasks than the faster node (node 2); hence, a larger AOCT is expected. On the other hand, any LB action taken after 1 s is not advantageous

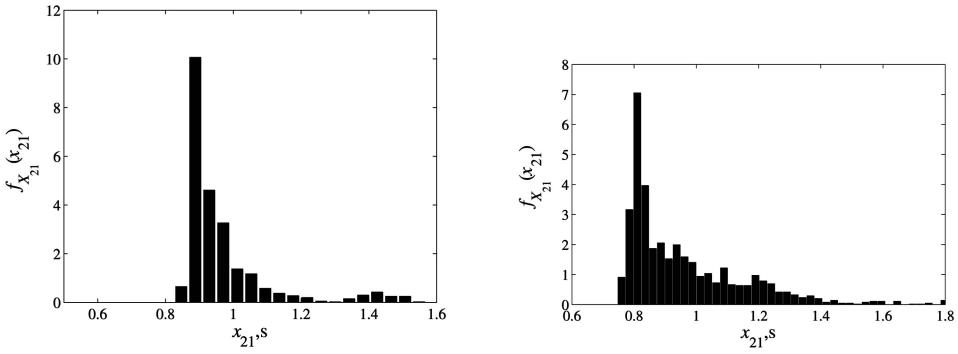



Fig. 1. Empirical pdfs of the communication delay from node 1 to node 2 obtained (a) on the Internet and (b) on the EECE WLAN.

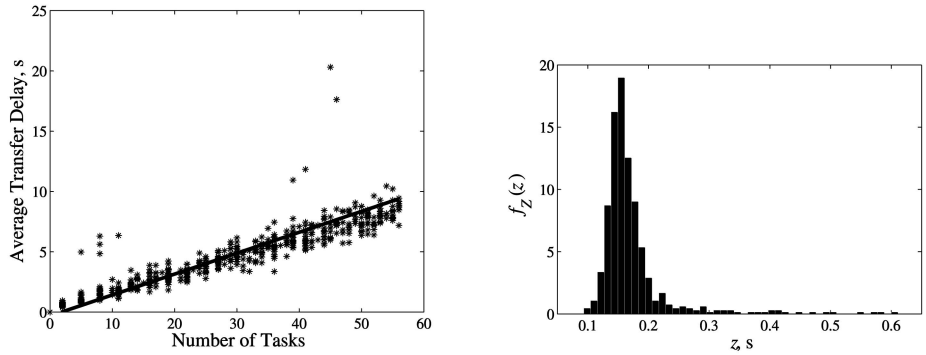



Fig. 2. (a) Mean delay as a function of the number of tasks transferred between nodes. The stars are the actual realizations from the experiments. (b) Empirical pdf of the transfer delay per task on the Internet under a normal work-day operation.

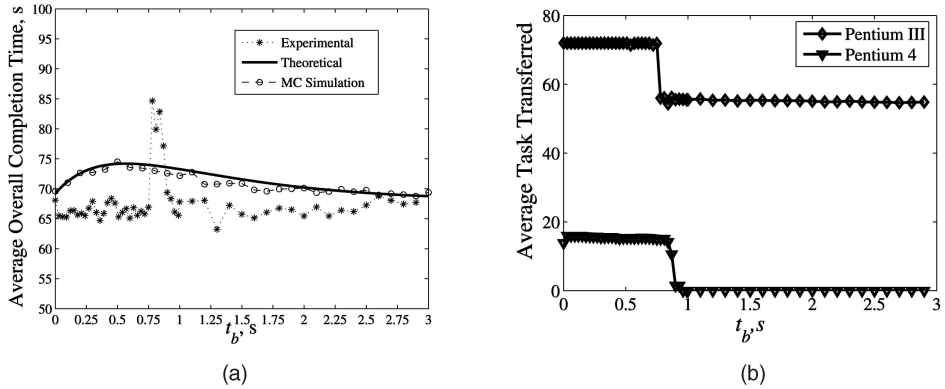



Fig. 3. (a) The AOCT as a function of LB instants for the experiments over the Internet. The LB gain was fixed at 1. (b) The amount of load transferred between nodes at different LB instants.

because there would be a low probability for information to arrive. If $t_b$ is delayed for too long, the slower node ends up computing more tasks, resulting in a larger AOCT (not shown in the figure).

Our next goal is to minimize the AOCT over $K$ while keeping $t_b$ fixed. The experiments were performed with the same initial configurations and the LB was triggered at 1 s using different gains. The results obtained over the Internet and WLAN are shown in Fig. 4. It is seen that the theoretical, MC-simulation, and experimental results are in good agreement and the optimal $K$ is approximately 1. This is almost equivalent to the hypothetical case when transfer delay is absent, in which case, perfect LB is achieved when $K = 1$ (or when, on average, 55 tasks are transferred from node 1 to node 2, as given by (6)). For experiments over the Internet, the empirically calculated average transfer delay per task was found to be 0.17 s and the average delay to transfer 55 tasks from node 1 to node 2 is therefore approximately 9 s. On the other hand, node 2 does not finish its initial load until 32 s, which means that there are no idle times at node 2 before the arrival of the transfer. Therefore, any transfer incurring a delay less than 32 s is effectively equivalent, as far as node 2 is concerned, to an instantaneous transfer. For experiments over WLAN, the initial load at node 1 and node 2 were set to 100 and 60 tasks, respectively, while the processing rates per task

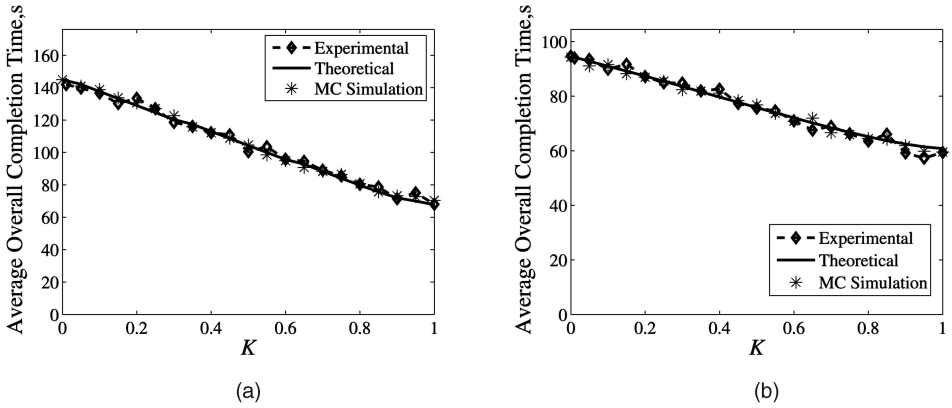



Fig. 4. The AOCT under different LB gains for (a) the Internet and (b) the WLAN. The LB instant was fixed at 1 s.

were estimated to be 1.07 and 1.85, respectively. The average delay to transfer 55 tasks was 5.5 s and the optimal performance was obtained for $K = 1$, as expected.

These results motivate us to look further into the effect of $K$ on the AOCT. Specifically, we consider the types of applications that impose a mean transfer delay greater than the mean processing time of the initial load at the receiver node, thereby resulting in an idle time for the receiving node. This kind of situation can arise in real applications, like processing of satellite images, where the images are large in size and, thus, the time to transfer them is greater than their processing time [19]. We simulated this type of behavior by means of our matrix-multiplication setup by increasing the mean size (in Bytes) of each row and simultaneously reducing the number of columns to be multiplied in the static matrix. Clearly, a larger row size increases the mean transfer delay per row (task) as well as the mean processing time per task. However, by reducing the number of columns in the static matrix, the mean processing time per task can be reduced. By using this approach, we were able to achieve a mean delay per task of 0.72 s while keeping the processing rates at 1.06 and 3.78 tasks per second for node 1 and node 2, respectively. The initial loads were still 100 and 60 tasks at nodes 1 and 2, respectively. Now, according to (6), with $K = 1$, the load to be transferred from node 1 is 64 tasks, producing a delay of 46 s. On the other hand, node 2, on average, finishes its initial load around 16 s, and it would therefore have long idle time while it is awaiting the arrival of load. This discussion is also supported by our theoretical and experimental results shown in Fig. 5a, where the AOCT is at minimum when $K = 0.7$, which holds for both experimental and theoretical curves. The error between the theoretical and experimental minima is approximately 12 percent. Finally, Fig. 5b shows the analytical optimal gain as a function of the mean transfer delay per task.

## 5.2 Proposed DLB Policy

In this section, we present the results on DLB policy for the experiments conducted over the Internet, whereby external loads of random sizes arrive randomly in time at any node in the distributed system. To recall, each instant an external load arrives to a node, the receiving node (and only the receiving node) takes a local, optimal one-shot LB action to minimize the AOCT of the total load in the system at that instant. As external tasks arrive with a certain rate, the total load and the overall completion time of the total load in the system change with time. The performance of DLB policy is now evaluated in terms of the average completion time per task (ACTT) corresponding to all tasks that are executed within a specified time-window, where the completion time of each task is defined as the sum of the processing time, the queuing time, and the transfer time of the task.

For all the experiments, the tasks are generated independently according to a compound (or generalized)

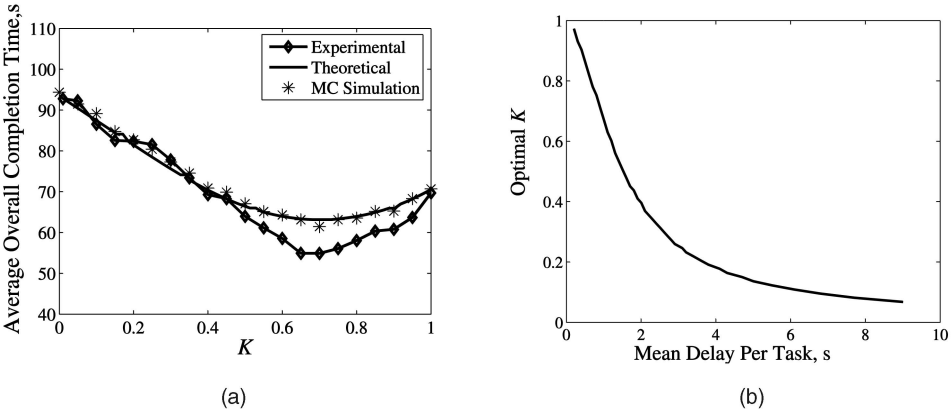



Fig. 5. (a) The AOCT as a function of the LB gain in presence of large transfer delay. The LB instant was fixed at 2 s. (b) The theoretical result on the optimal LB gain for mean transfer delays per task.

TABLE 1
Experimental Results

| Experiment Number | Average Completion Time per Task (s) | | | System Processing Rate (task $s^{-1}$) | | |
|---|---|---|---|---|---|---|
| | DLB | $K$=0.1 | $K$=1 | DLB | $K$=0.1 | $K$=1 |
| 1 | 22.55 | 73.87 | 49.76 | 1.75 | 1.69 | 1.11 |
| 2 | 8.61 | 15.82 | 11.67 | 3.3 | 3.06 | 2.92 |
| 3 | 9 | 10.56 | 10.77 | 3.29 | 3.73 | 2.84 |

Poisson process with Poisson-distributed marks [20]. More precisely, the external loads arrive according to a Poisson process, and the numbers of tasks at the load-arrival instants constitute a sequence of independent and identically distributed Poisson random variables. (Recall that the task size, in terms of Bytes per task, is also random, according to a geometric distribution.) Note that, since the proposed DLB policy is triggered by the arrival of tasks and it is based on the actual realization of the task number in each arrival, it is independent of the statistics of the number of tasks per arrival as well as the statistics of the underlying task-arrival process.

The experiments were conducted for three different cases: Experiment 1: Node 1 receives, on average, 55 external tasks at each arrival and the average interarrival time is set to be 40 s, while no external tasks are generated at node 2. Experiment 2: Node 2 receives, on average, 22 external tasks at each arrival and the average interarrival time is 9 s, while no external tasks are generated at node 1. Experiment 3: Nodes 1 and 2 independently receive, on average, 16 and 40 external tasks, respectively, at each arrival and the average interarrival times are 20 s and 18 s for nodes 1 and 2, respectively. The empirical estimates of the processing rates of nodes 1 and 2 were found to be 1.06 and 3.78 tasks per second, respectively. The estimate of the average transfer delay per task, $\theta_{ji}^{(k)}$, is updated after every transfer of tasks according to (15), with $\theta_{ji}^{(0)} = .85$ s and $\alpha = .05$.

Each experiment was conducted for a period (time-window) of 1 hour and the ACTT corresponding to each case is listed in Table 1. We also show the ACTT obtained using static policies that perform LB with fixed gains of $K = 0.1$ and $K = 1$ at all arrival instants. It is clear from Table 1 that the ACTT is the minimum for the DLB policy for all three experiments. Considering Experiment 1, note that the average rate of arrival at node 1 is 1.37 tasks per second since the interarrival times are independent of arrival sizes. Therefore, the average arrival rate of node 1 is greater than its processing rate (1.06 tasks per second), but it is smaller than the combined processing rates of the nodes. With LB, some portion of the arriving tasks is diverted to node 2, which reduces the effective arrival rate at node 1 and thus avoids load accumulation. In the static LB policy with $K = 0.1$, node 1 keeps 90 percent of its excess load and, hence, the effective arrival rate at node 1 remains larger than its processing rate. Therefore, the queue-length accumulates with every arrival, which results in a greater queuing delay, and thus, excess ACTT. In contrast, in the static policy with $K = 1$, node 1 sends all of its excess load to node 2 at every LB instant. However, each batch of transferred load undergoes a large delay, resulting in an increase in ACTT.

In the case of Experiment 2, the average rate of arrival at node 2 is 2.44 tasks per second, which is smaller than the processing speed of node 2. As a result, the static LB with $K = 1$ gives a reduced ACTT compared to $K = 0.1$, meaning that the increase in ACTT due to queuing delay at node 2 for $K = 0.1$ is greater than the increase in ACTT caused by the transfer delay when $K = 1$. However, the DLB outperforms the static case of $K = 1$ due to excessive delay in load transfer associated to this static LB case. For Experiment 3, the ACTTs are evidently similar under both $K = 0.1$ and $K = 1$ static LB policies. This is because ACTT is dominated by queuing delay in the $K = 0.1$ (at the slower node 1) case while it is dominated by transfer delay in the $K = 1$ case. On the other hand, the DLB policy effectively uses the system resources, viz., the nodes and the channel, to avoid excessive queuing delay as well as the transfer delay.

We now look at the effect of LB policies on the *system processing rate* (SPR), which is calculated as the total number of tasks executed by the system in a certain time-window divided by the active time of the system. The *active time* of the system within a time-window is defined as the aggregate of all times for which there is at least one task in the system that is either being processed or being transferred. The SPR achieved under different LB policies are listed in Table 1. It is interesting to note that, in the case of Experiment 1, better SPR is achieved with $K = 0.1$ than with $K = 1$, despite the fact that the latter performs better in terms of ACTT. To explain this behavior, we first need to look at one extreme case when no LB is performed. In this case, the SPR is always equal to $\lambda_{d_1}$ independently of the size of time window. However, as we increase the time window, the ACTT diverges to infinity since the average rate of arrival is bigger than the average processing rate of node 1. The performance for the case of a weak LB action with $K = 0.1$ is found to be similar to the extreme case of no LB. In the second case, when LB is performed with $K = 1$, the active time of the system gets dominated by times when there are tasks in transfer while both nodes are idle. Consequently, the number of tasks processed by the system is less while the active time of the system may increase, resulting in a reduced SPR. However, the LB action taken by node 1 reduces the effective arrival rate at node 1 below its processing rate. As a result, the ACTT of the system is bounded.

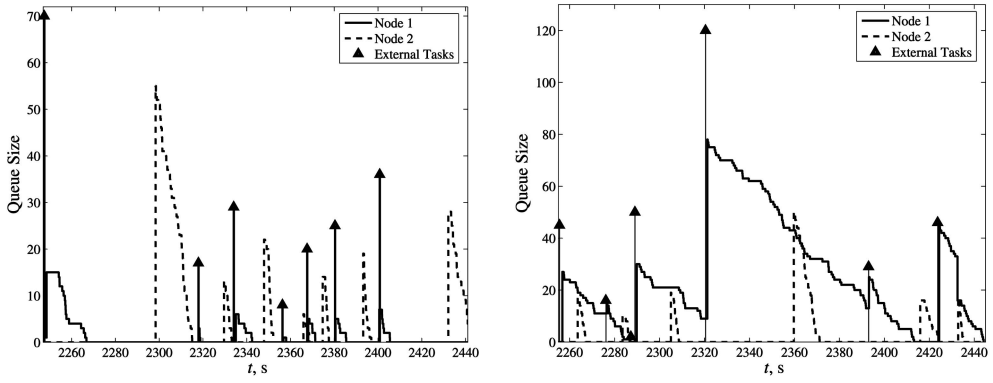



Fig. 6. One realization of the queues under a static LB policy using (a) a fixed gain $K = 1$ and (b) DLB policy.

In the case of DLB policy, LB gains are chosen small enough to avoid large transfer delays but large enough to lower the effective arrival rate at node 1. Therefore, for Experiment 1, the DLB policy achieves the maximum SPR and the minimum ACTT. The fact that nodes have large idle times while there are tasks in transfer for the case of $K = 1$ is depicted in Fig. 6. Observe that, when there is an arrival of 70 tasks at node 1 around 2,250 s, 55 tasks are transferred to node 2. On the other hand, node 2 has an empty queue at the arrival instant of node 1 and, due to the transfer delay, it must wait another 50 s to receive the tasks. Further, node 1 finishes the remaining 15 tasks and becomes idle by the time node 2 gets the transferred load. This behavior is repeated at all arrival instants, which are marked by arrows in Fig. 6a. In contrast, from Fig. 6b, it can be seen that the transfer delay mostly overlaps with the working times of the sender node, which results in smaller idle times on both nodes. Similar results are observed for Experiment 2.

In the case of Experiment 3, node 1 and node 2 receive external loads at a rate of 0.8 and 2.2 tasks per second, respectively. This means that, even if no LB is performed, both nodes process their own tasks without being idle for a long time. Therefore, the SPR is expected to be close to the sum of the processing rates of the nodes. However, when LB is performed, nodes may become idle due to the transfer delay, resulting in smaller SPR. This is evident from our results of Experiment 3 where the static LB policy with $K = 0.1$ achieves maximum SPR. On the other hand, the DLB policy transfers the right amount of tasks at every LB instant, so that the transfer delays plus the queuing delays at the receiving node are smaller than the queuing delays for those tasks at the sender node. This reduces the ACTT but may or may not increase SPR depending on the resulting active time.

### 5.3 Comparison to Other DLB Policies

Next, we will compare the performance of our DLB policy to versions of two existing LB policies for heterogeneous and dynamic computing, namely, the shortest-expected-delay (SED) policy [21] and the never-queue (NQ) policy [22], which we have adapted to our distributed-computing setting. Suppose that external arrival of $x$ tasks occurs at node $i$ at time $t$. Let $m_{j(i)}(t)$ be the queue lengths of node $j$ as per the knowledge of node $i$ at time $t$. Let $l_{j(i)}(t)$ be the ACTT for the batch of $x$ external tasks if all the external tasks join the queue of node $j$. The average completion time per task (per batch of $x$ arriving tasks) can be expressed as

$$l_{j(i)}(t) = \frac{1}{x}\sum_{r=1}^{x}\left(\frac{m_{j(i)}(t)+r}{\lambda_{d_j}} + \theta_{ji}^{(k)}x\right) = \frac{m_{j(i)}(t)}{\lambda_{d_j}} + \frac{x+1}{2\lambda_{d_j}} + \theta_{ji}^{(k)}x, \quad (16)$$

where $\theta_{ji}^{(k)}$ is the $k$th update of average transfer delay per task sent from node $i$ to node $j$ (with $\theta_{ii}^{(k)} = 0$). In the SED policy, the batch of $x$ tasks is assigned to the node that achieves the minimum ACTT. Therefore, the receiver node is identified as $\mathrm{argmin}_j(l_{j(i)}(t))$. On the other hand, in the NQ policy, all external loads are assigned to a node that has an empty queue. If more than one node have an empty queue, the SED policy is invoked among the nodes with the empty queues to choose a receiver node. Similarly, if none of the queues is empty, the SED policy is invoked again to choose the receiver node among all the nodes.

We implemented the SED and the NQ policies to perform the distributed computing experiments on our testbed. The experiments were conducted between two nodes connected over the Internet (keeping the same processing speeds per task). We performed three types of experiments for each policy: 1) node 1 receiving, on average, 20 tasks at each arrival and the average interarrival time set to 12 s while no external tasks were generated at node 2, 2) node 2 receiving, on average, 25 tasks at each arrival and the average interarrival time set to 8 s, and 3) node 1 and node 2 independently receiving, on average, 10 and 15 external tasks at each arrival and the average interarrival times set to 8 s and 7 s, respectively. Each experiment was conducted for a two-hour period. The results, shown in Table 2, suggest that the ACTT achieved

TABLE 2
Experimental Results of ACTT

| Experiment Number | Average Completion Time per Task (s) | | |
|---|---|---|---|
| | DLB | SED | NQ |
| i | 7.61 | 15.19 | 15.55 |
| ii | 6.09 | 13.68 | 13.77 |
| iii | 4.71 | 6.92 | 7.44 |

from the DLB policy is approximately half the ACTT achieved from either the SED or NQ policies.

It should be noted that the complexity of solving (14) grows with the number of nodes and the added computational overhead needs to be considered as well. Specifically, when the delays imposed by the channel differ according to paths between nodes, the LB gains $K_{ij}$, for all $i$, can no longer be parameterized by one value $K$. In such cases, it is not computationally efficient to perform the online optimization required by the DLB policy. While this analysis is not within the scope of this paper, we would like to suggest a suboptimal solution for the LB gains that can easily be obtained based on the solution for a two-node system. Suppose that, in an $n$-node distributed system, node $j$ receives external load at time $t_a$ and an LB action needs to be triggered instantly. Based on the knowledge of node $j$ about the queue lengths of all other nodes, the excess load of node $j$ as well as the partitions $p_{ij}$ can easily be calculated using the equations given in Section 2. In order to calculate the optimal LB gain $K_{ij}$, for each $i \neq j$, fix a node pair $(i,j)$ and assume that $K_{kj} = 1$ for all $k \neq i,j$, meaning node $j$ could send full partition $p_{kj}$ of the excess load to all other nodes except node $i$. Now, the problem reduces to finding the optimal gain $K_{ij}$ for a two-node system $(i,j)$, where, after the execution of LB, nodes $i$ and $j$ have loads $m_{i(j)}(t_a)$ and $m_{j(j)}(t_a) - \sum_{k\neq i,j}\lfloor p_{kj}L_j^{ex}(t_a)\rfloor - \lfloor K_{ij}p_{ij}L_j^{ex}(t_a)\rfloor$, respectively, while $\lfloor K_{ij}p_{ij}L_j^{ex}(t_a)\rfloor$ tasks are in transit to node $i$. Regeneration theory can now be utilized to obtain difference equations that can be solved easily to compute the optimal $K_{ij}$. In summary, we would need to solve at most $n-1$ independent two-dimensional difference equations, one equation for each $i \neq j$, as compared to solving one $n$-dimensional difference equation given by (14). Therefore, in this suboptimal approach, an efficient automated code can be used to compute the optimal gains online.

## 6 Conclusion

A continuous-time stochastic model has been formulated for the queues' dynamics of a distributed computing system in the context of load balancing. The model takes into account the randomness in delay and allows random arrivals of external loads. At first, the model was simplified by relaxing external arrivals of loads and an optimization problem was formulated for minimizing the average overall completion time. Based on the theory of regeneration, we showed that a one-shot load balancing policy can be optimized over the balancing gain and the balancing instant that together minimize the average overall completion time for a certain initial load. We also looked at the interplay between the balancing gain and the size of the random delay in the channel. The theoretical predictions, MC simulations, and the experimental results all showed that, when the average transfer delay per task is large compared to the average processing time per task, reduced load-balancing strength (or gain) minimizes the average overall completion time.

The optimal one-shot load-balancing approach was then adapted to develop a distributed and dynamic load-balancing policy in which, at every external load arrival, the receiver node executes load balancing autonomously. Further, the optimal gains are calculated on-the-fly, based on the system parameters that are adaptively updated. Thus, the dynamic-load-balancing policy can adapt to the changing traffic conditions in the channel as well as the change in task processing rates induced from the type of applications. We have shown experimentally that the proposed dynamic-load-balancing policy minimizes the average completion time per task while improving the system processing rate. The interplay between the queuing delays and the transfer delays as well as their effects on the average completion time per task and system processing rate were investigated. In particular, the average completion time per task achieved under the proposed dynamic-load-balancing policy is significantly less than those achieved by the commonly used SED and NQ policies. This is attributable to the fact that the dynamic-load-balancing policy achieves a higher success, in comparison to the SED and NQ policies, in reducing the likelihood of nodes being idle while there are tasks in the system, comprising tasks in the queues as well as those in transit.

Our future work considers the implementation and evaluation of the proposed suboptimal solution on a multinode system. To this end, we will consider a wireless sensor network where the nodes are constrained in computing power as well as power consumption.

## Appendix A

## Optimality of Partitions in the Ideal Case

By ideal case, we mean that there are no delays, the queues are deterministic, and the tasks are arbitrarily divisible. This effectively means that each node in the system has the exact queue size of other nodes. Consequently, it follows that $m_{i(j)}(t) = Q_i(t)$, $\mathcal{I}_j = \mathcal{I}$, and $p_{ij} \equiv p_i$, independently of $j$. Assume further that LB actions are executed together at time $t$ at all the nodes that do not belong to $\mathcal{I}$. Let $Q_i^f(t)$ be the total load at node $i \in \mathcal{I}$ after the execution of LB. Then,

$$\begin{aligned} Q_i^f(t) &= Q_i(t) + p_i \sum_{j\in\mathcal{I}^c} L_j^{ex}(t) \\ &= Q_i(t) + \frac{L_i^{ex}(t)}{\sum_{j\in\mathcal{I}} L_j^{ex}(t)} \sum_{j\in\mathcal{I}^c} L_j^{ex}(t). \end{aligned} \tag{17}$$

Since $\sum_{j=1}^n L_j^{ex}(t) = 0$, we have

$$\sum\nolimits_{j\in\mathcal{I}} L_j^{ex}(t) = -\sum\nolimits_{j\in\mathcal{I}^c} L_j^{ex}(t).$$

Therefore,

$$Q_i^f(t) = Q_i(t) - L_i^{ex}(t) = \lambda_{d_i} \frac{\sum_{l=1}^n Q_l(t)}{\sum_{l=1}^n \lambda_{d_l}}. \tag{18}$$

Clearly, the overall completion time is $\frac{\sum_{l=1}^n Q_l(t)}{\sum_{l=1}^n \lambda_{d_l}}$ for all the nodes.

## Appendix B

## Derivation of Renewal Equations

Consider the integro-difference equation given in (9). By exploiting the fact that the minimum of independent exponential random variables is also an exponential random variable, we obtain $f_\tau(t) = \lambda e^{-\lambda t}u(t)$, where $\lambda = \sum_{i=1}^n(\lambda_{d_i} + \sum_{j\neq i}\lambda_{ij})$. Further, $\mathrm{P}\{\tau = W_i \mid \tau = s\} = \frac{\lambda_{d_i}}{\lambda}$ and $\mathrm{P}\{\tau = X_{ij} \mid \tau = s\} = \frac{\lambda_{ij}}{\lambda}$. Therefore, (9) can be written as

$$\mu^{\mathbf{I}}_{m_1,\ldots,m_n}(t_b) = \left(\mu^{\mathbf{I}}_{m_1,\ldots,m_n}(0) + t_b\right) \int_{t_b}^{\infty} \lambda e^{-\lambda s}\, ds + \int_0^{t_b} s e^{-\lambda s} ds \int_0^{t_b} \left[ \sum_{i=1}^{n} \lambda_{d_i} \mu^{\mathbf{I}}_{m_1-\delta_{1,i},\ldots,m_n-\delta_{n,i}}(t_b - s) + \sum_{i=1}^{n} \sum_{j \neq i} \lambda_{ij} \mu^{\mathbf{I}^{ij}}_{m_1,\ldots,m_n}(t_b - s) \right] e^{-\lambda s} ds. \quad (19)$$

Using the Leibnitz integral rule and change of variables, it is easy to show that

$$\frac{d}{dt_b} \int_0^{t_b} \lambda_{d_i} \mu^{\mathbf{I}}_{m_1-\delta_{1,i},\ldots,m_n-\delta_{n,i}}(t_b - s) e^{-\lambda s} ds = - \lambda \int_0^{t_b} \lambda_{d_i} \mu^{\mathbf{I}}_{m_1-\delta_{1,i},\ldots,m_n-\delta_{n,i}}(t_b - s) e^{-\lambda s} ds + \lambda_{d_i} \mu^{\mathbf{I}}_{m_1-\delta_{1,i},\ldots,m_n-\delta_{n,i}}(t_b). \quad (20)$$

Differentiating (19) with $t_b$, using identities similar to (20) and arranging the terms, we get (10).

Next, we present the integro-difference equations to characterize $F_{T_1}(r_1; L_{12}; t)$, which will lead to (12) after differentiation with respect to $t$. Let $T_1(r_1; L_{12}) \equiv T_1$ be the total completion time of node 1, and we are interested in calculating $F_{T_1}(r_1; L_{12}; t) = \mathrm{P}\{T_1(r_1; L_{12}) \leq t\}$. With LB at time $t = 0$, the regeneration event at node 1 can either be the arrival of $L_{12}$ load sent by node 2 or the execution of a task by node 1 (if $r_1 > 0$). If the regeneration event at time $s \in [0, t]$ is the arrival of $L_{12}$ load, using the memoryless property of exponential r.v., we obtain a new queue at node 1 having $r_1 + L_{12}$ load with exponential service time for each task, while there is no load in transit. Therefore, we need to calculate $\mathrm{P}\{T_1(r_1 + L_{12}; 0) \leq t - s\}$. Instead, if the regeneration event is the task execution at node 1, we need to look at $\mathrm{P}\{T_1(r_1 - 1; L_{12}) \leq t - s\}$. Therefore,

$$\mathrm{P}\{T_1(r_1; L_{12}) \leq t\} = \int_0^t f_\tau(s) \left[ \mathrm{P}\{T_1(r_1 - 1; L_{12}) \leq t - s\} \frac{\lambda_{d_1}}{\lambda} + \mathrm{P}\{T_1(r_1 + L_{12}; 0) \leq t - s\} \frac{\lambda_{t_{21}}}{\lambda} \right] ds,$$

where $\lambda = \lambda_{d_1} + \lambda_{t_{21}}$. We can solve for $\mathrm{P}\{T_2(r_2; L_{21}) \leq t\}$ similarly.

## APPENDIX C
## DETAILED ALGORITHM FOR DYNAMIC LOAD BALANCING

For an $n$-node distributed system, we specify the "sync" periods for each node by $\delta_j$, $j = 1, \ldots, n$. These are the periods, for each node, at which each node broadcasts its queue length and processing speed to other nodes. (In our experiments, we used a common sync period of 1 s.)

Algorithm:
$\forall t \geq 0$, at every node $j$, the DLB algorithm is:
**if** $\mathrm{mod}(t, \delta_j) = 0$ **then**
Broadcast current queue size and current processing rate
**end if**
**if** "sync" is received **then**
Update queue size and processing rate of the sender node
**end if**
**if** external-load is received, say at time $t = t_a$ **then**
Calculate local excess load from (2), partitions from (3) or (5), and optimal $K_{ij}$ from (14)
Perform LB *only* by node $j$ in accordance to (6)
Update $\theta^k_{ij}$ using (15) after each load transmission numbered by $k$
**end if**

## ACKNOWLEDGMENTS

This work was supported by the US National Science Foundation (NSF) under Award ANI-0312611 and in part by the US Air Force Research Laboratory, NSF Grants CAREER CCF-0611589, ACI-00-93039, NSF DBI-0420513, ITR ACI-00-81404, ITR EIA-01-21377, Biocomplexity DEB-01-20709, ITR EF/BIO 03-31654, and Defense Advanced Research Projects Agency Contract NBCH30390004.

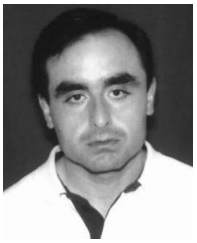
## REFERENCES

[1] http://www.planetlab.org, 2004.
[2] Z. Lan, V.E. Taylor, and G. Bryan, "Dynamic Load Balancing for Adaptive Mesh Refinement Application," *Proc. Int'l Conf. Parallel Processing (ICPP)*, 2001.
[3] 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.
[4] G. Cybenko, "Dynamic Load Balancing for Distributed Memory Multiprocessors," *J. Parallel and Distributed Computing*, vol. 7, pp. 279-301, Oct. 1989.
[5] C. Hui and S.T. Chanson, "Hydrodynamic Load Balancing," *IEEE Trans. Parallel and Distributed Systems*, vol. 10, no. 11, pp. 1118-1137, Nov. 1999.
[6] B.W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *The Bell System Technical J.*, vol. 49, pp. 291-307, Feb. 1970.
[7] M.M. Hayat, S. Dhakal, C.T. Abdallah, J.D. Birdwell, and J. Chiasson, "Dynamic Time Delay Models for Load Balancing. Part II: Stochastic Analysis of the Effect of Delay Uncertainty," *Advances in Time Delay Systems*, vol. 38, pp. 355-368, Springer-Verlag, 2004.
[8] S. Dhakal, B.S. Paskaleva, M.M. Hayat, E. Schamiloglu, and C.T. Abdallah, "Dynamical Discrete-Time Load Balancing in Distributed Systems in the Presence of Time Delays," *Proc. IEEE Conf. Decision and Controls (CDC '03)*, pp. 5128-5134, Dec. 2003.
[9] S. Dhakal, M.M. Hayat, M. Elyas, J. Ghanem, and C.T. Abdallah, "Load Balancing in Distributed Computing over Wireless LAN: Effects of Network Delay," *Proc. IEEE Wireless Comm. and Networking Conf. (WCNC '05)*, Mar. 2005.
[10] D.L. Eager, E.D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. Software Eng.*, vol. 12, no. 5, pp. 662-675, May 1986.
[11] J. Liu and V.A. Saletore, "Self-Scheduling on Distributed-Memory Machines," *Proc. ACM Int'l Conf. Supercomputing*, pp. 814-823, Nov. 1993.
[12] J.M. Bahi, C. Vivier, and R. Couturier, "Dynamic Load Balancing and Efficient Load Estimators for Asynchronous Iterative Algorithms," *IEEE Trans. Parallel and Distributed Systems*, vol. 16, no. 4, Apr. 2005.
[13] A. Cortes, A. Ripoll, M. Senar, and E. Luque, "Performance Comparison of Dynamic Load-Balancing Strategies for Distributed Computing," *Proc. 32nd Hawaii Conf. System Sciences*, vol. 8, p. 8041, 1999.
[14] M. Trehel, C. Balayer, and A. Alloui, "Modeling Load Balancing Inside Groups Using Queuing Theory," *Proc. 10th Int'l Conf. Parallel and Distributed Computing System*, Oct. 1997.
[15] C. Knessly and C. Tiery, "Two Tandem Queues with General Renewal Input I: Diffusion Approximation and Integral Representation," *SIAM J. Applied Math.*, vol. 59, pp. 1917-1959, 1999.
[16] F. Bacelli and P. Bremaud, *Elements of Queuing Theory: Palm-Martingale Calculus and Stochastic Recurrence.* Springer-Verlag, 1994.

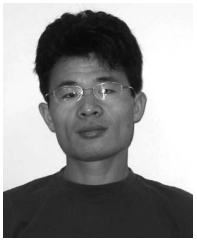
[17] D.J. Daley and D. Vere-Jones, *An Introduction to the Theory of Point Processes.* Springer-Verlag, 1988.
[18] V. Jacobson, "Congestion Avoidance and Control," *Proc. ACM SIGCOMM,* Aug. 1988.
[19] G. Petrie, G. Fann, E. Jurrus, B. Moon, K. Perrine, C. Dippold, and D. Jones, "A Distributed Computing Approach for Remote Sensing Data," *Proc. 34th Symp. Interface,* pp. 477-489, 2002.
[20] D.L. Snyder and M.I. Miller, *Random Point Processes in Time and Space.* 1991.
[21] S. Shenker and A. Weinrib, "The Optimal Control of Heterogeneous Queuing Systems: A Paradigm for Load Sharing and Routing," *IEEE Trans. Computers,* vol. 38, no. 12, pp. 1724-1735, Dec. 1989.
[22] K. Kabalan, W. Smari, and J. Hakimian, "Adaptive Load Sharing in Heterogeneous Systems: Policies, Modifications, and Simulation," *Int'l J. Simulation Systems Science and Technology,* vol. 3, nos. 1-2, pp. 89-100, June 2002.

**Sagar Dhakal** received the bachelor of engineering degree in electrical and electronics engineering in May 2001 from Birla Institute of Technology, India. He received the MS and PhD degrees in electrical engineering, respectively, in December 2003 and December 2006, from the University of New Mexico. From August 2001 to July 2002, he served as an instructor in the Electrical and Electronics Engineering Department at Kathmandu University, Nepal. He is currently working at NORTEL Networks, Richardson, Texas. His research interests include queuing theoretic modeling and stochastic optimization of distributed systems and wireless communication systems.

**Majeed M. Hayat** (S'89-M'92-SM'00) received the BS degree (summa cum laude) in 1985 in electrical engineering from the University of the Pacific, Stockton, California. He received the MS and PhD degrees in electrical and computer engineering, respectively, in 1988 and 1992, from the University of Wisconsin-Madison. From 1993 to 1996, he worked at the University of Wisconsin-Madison as a research associate and co-principal investigator of a project on statistical minefield modeling and detection, which was funded by the US Office of Naval Research. In 1996, he joined the faculty of the Electro-Optics Graduate Program and the Department of Electrical and Computer Engineering at the University of Dayton. He is currently an associate professor in the Department of Electrical and Computer Engineering at the University of New Mexico. His research contributions cover a broad range of topics in statistical communication theory, and signal/image processing, as well as applied probability theory and stochastic processes. Some of his research areas include queuing theory for networks, noise in avalanche photodiodes, equalization in optical receivers, spatial-noise-reduction strategies for focal-pane arrays, and spectral imaging. He is a recipient of a 1998 US National Science Foundation Early Faculty Career Award. He is a senior member of the IEEE and a member of SPIE and OSA. Dr. Hayat is an associate editor of *Optics Express* and an associate editor member of the conference editiorial board of the IEEE Control Systems Society.



**Jorge E. Pezoa** received the bachelor of engineering degree in electronics and the MSc degree in electrical engineering with honors in 1999 and 2003, respectively, from the University of Concepción, Chile. From 2003-2004, he served as an instructor in the Electrical Engineering Department at the University of Concepción. Currently, he is working toward the PhD degree in the areas of communications and signal processing.



**Cundong Yang** is a graduate student in the Electrical and Computer Engineering Department at the University of New Mexico, and works as a software engineer at Teledex LLC, San Jose, California. His areas of interest are wireless networks, VoIP, and optimization of parallel algorithms. From 2002-2004, Cundong worked as a software engineer in Huawei Technologies, Shenzhen, China on the R&D of radio resource management algorithms for WCDMA communication system.

**David A. Bader** received the PhD degree in 1996 from the University of Maryland and was awarded a US National Science Foundation (NSF) Postdoctoral Research Associateship in Experimental Computer Science. From 1998-2005, He served on the faculty at the University of New Mexico. He is an associate professor in computational science and engineering, a division within the College of Computing, at the Georgia Institute of Technology. He is an NSF CAREER Award recipient, an investigator on several NSF awards, a distinguished speaker in the IEEE Computer Society Distinguished Visitors Program, and a member of the IBM PERCS team for the DARPA High Productivity Computing Systems program. Dr. Bader serves on the steering committees of the IPDPS and HiPC conferences and was the general cochair for IPDPS (2004-2005) and vice general chair for HiPC (2002-2004). He has chaired several major conference program committees: program chair for HiPC 2005, program vice-chair for IPDPS 2006, and program vice-chair for ICPP 2006. He has served on numerous conference program committees related to parallel processing and computational science and engineering and is an associate editor for several high-impact publications, including the *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, the *ACM Journal of Experimental Algorithmics (JEA)*, *IEEE DS Online*, and *Parallel Computing*. He is a senior member of the IEEE and the IEEE Computer Society and a member of the ACM. Dr. Bader has been a pioneer in the field of high-performance computing for problems in bioinformatics and computational genomics. He has cochaired a series of meetings, the IEEE International Workshop on High-Performance Computational Biology (HiCOMB), written several book chapters, and coedited special issues of the *Journal of Parallel and Distributed Computing (JPDC)* and *IEEE TPDS* on high-performance computational biology. He has coauthored more than 75 articles in peer-reviewed journals and conferences, and his main areas of research are in parallel algorithms, combinatorial optimization, and computational biology and genomics.