Load Balancing in the Presence of Random Node Failure and Recovery

Sagar Dhakal[1], Majeed M. Hayat[1], Jorge E. Pezoa[1], Chaouki T. Abdallah[1],
J. Doug Birdwell[2], and John Chiasson[2]

[1]University of New Mexico
Dept. of Electrical and Computer Engineering
Albuquerque, NM 87131-0001 USA
{dhakal, hayat, jpezoa, chaouki}@eece.unm.edu

[2]University of Tennessee
Dept. of Electrical and Computer Engineering
Knoxville, TN 37996-2100 USA
{birdwell, chiasson}@utk.edu

Abstract

In many distributed computing systems that are prone to either induced or spontaneous node failures, the number of available computing resources is dynamically changing in a random fashion. A load-balancing (LB) policy for such systems should therefore be robust, in terms of workload re-allocation and effectiveness in task completion, with respect to the random absence and re-emergence of nodes as well as random delays in the transfer of workloads among nodes. In this paper two LB policies for such computing environments are presented: The first policy takes an initial LB action to preemptively counteract the consequences of random failure and recovery of nodes. The second policy compensates for the occurrence of node failure dynamically by transferring loads only at the actual failure instants. A probabilistic model, based on the concept of regenerative processes, is presented to assess the overall performance of the system under these policies. Optimal performance of both policies is evaluated using analytical, experimental and simulation-based results. The interplay between node-failure/recovery rates and the mean load-transfer delay are highlighted.

1. Introduction

In a distributed computing system, large workloads are divided among independent computational elements (CEs), or nodes, in an attempt to minimize the average service time per task of the entire system. Such load allocation is referred in the literature as load balancing (LB). In a heterogeneous computing environment, where different nodes (links) may have different processing speeds (delays), an effective LB policy must consider factors like inhomogeneity in node's processing speeds, variability and inhomogeneity in delays in inter-node communications, the number of available nodes in the system, etc.

Additionally, a distributed computing system may utilize dynamic sets of CEs, where nodes may join and leave the system in a random fashion. An example of such systems is "SETI at Home" [1]. Such systems typically use dedicated workstations as well as dynamic resources comprising a network of non-dedicated nodes, such as a collection of desk-tops or portable computing devices that are online, which are used remotely, upon availability, to participate in the distributed computing. However, these nodes can go off-line anytime, regardless of the portion of the load assigned to them. Furthermore, the participation of any node may be interrupted by either the local usage of the node by its owner or due to the occurrence of physical failure or damage to the node. (The latter effect applies even to the set of dedicated nodes.) Such scenarios induce an uncertainty in the availability of the number of functional nodes, whereby any node (including the dedicated nodes) may randomly fluctuate between a "failure" (or "down") and "working" (or "up") states.

Clearly, uncertainty in the number of working nodes is expected to degrade the performance of any LB policy which does not account for the above-described node-failure and recovery mechanism. More precisely, the distribution of task completion time (or service time) is dependent on the statistics of node failure and recovery. Available literature on distributed computing in such uncertain environments, primarily considers LB policies where any node failure is addressed only after its occurrence. Checkpoint-resume or terminate-

1-4244-0054-6/06/$20.00

restart mechanisms are used to detect failures and recover unprocessed tasks at the failed nodes [2, 3]. The node failure can also be tackled by keeping multiple copies of workload on different nodes [4], while the work of Choi *et al.* [5] addresses performance of the system based only on the unreliable resources, without addressing the cooperation between dedicated and non-dedicated nodes. Additionally, most of the existing literature [6, 7] that offers analytical formulation of distributed-computing systems assumes a homogeneous setup and that network delays are deterministic. However, in distributed systems that operate over a wireless LAN, these assumptions are violated since (1) the communication delays are strongly random, (2) the statistics of these delays are inhomogeneous, and (3) the nodes are not homogeneous. We have reported in our earlier work that conventional LB policies that do not consider the presence of random delays perform poorly in a delay infested systems [8, 9].

To the best of our knowledge, there are no LB policies which take preemptive actions to compensate for node failure and recovery in heterogeneous distributed-computing systems. By preemptive we mean adjusting the LB action to compensate for the possibility of node failure/recovery before node failure occurs. To appreciate why such preemptive action is useful, consider a scenario where a node fails while having a large sum of unprocessed load, which would be transferred to other nodes upon its failure. However, the transfer of such large load may be accompanied by a large, random delay, which may potentially result in idle times for the other nodes. Note that such action would not be ineffective had the delay been unsubstantial. Another scenario where action-upon-failure LB policy may not be effective is when the failure rate of a node is high. In this case, the frequent load transfer upon-failure would result in an increase in the overall volume of loads that are in transit. Therefore, depending on the statistics of the heterogeneous delay and node failure, we suspect that a preemptive LB policy may provide superior performance compared to an action-upon-failure-based LB policy. There is therefore a need to develop and analyze preemptive LB policies and understand the precise conditions at which they become effective.

In this paper we present LB policies for distributed computing systems where each node may fail randomly in time and subsequently recover in random amount of time. The policies take into account the randomness in processing speed of each node as well as the uncertainty in the load-transfer delay between the nodes. Two different LB policies are presented specialized for two-node systems. In the first LB policy (LBP-1), scheduling is performed preemptively at the beginning of the execution of a workload. In the second policy (LBP-2), an initial scheduling is performed without the prior knowledge of the statistics of failure and recovery, but additional LB action is taken at every occurrence of node failure. The performance of both policies are analytically modeled, using the concept of regenerative processes [12], and optimized over the initial LB gain to minimize the average overall completion time for a fixed initial load distributions. Both policies are applied to real-time experiments over a wireless LAN test-bed and the results are compared to the theory and Monte-Carlo (MC) simulations. The theory presented in this paper can be extended to a multi-node system in a straightforward way.

This paper is organized as follows. Section 2 contains the description of both LBP-1 and LBP-2 followed by regeneration-theory-based analysis of the stochastic dynamics of LBP-1. The architecture of the distributed computing system implemented to test the LB is detailed in Section 3. In Section 4 we present the experimental results obtained using a wireless LAN environment and compare them to theoretical predictions and to MC simulations. Finally, our conclusions are given in Section 5.

2. LB policies and Theoretical Modeling

Two different LB policies, namely LBP-1 and LBP-2, are presented in this section. Both policies allow the nodes to jointly execute scheduling at time $t = 0$. LBP-1 takes a proactive initial scheduling action by considering the node failure and recovery, while in LBP-2, the initial scheduling does not account for the node failure but a new balancing action is allowed at every failure instant to compensate for the node failure. In the rest of the paper, a task is the smallest indivisible unit of workload, and a load, or workload, refers to a collection of tasks. For simplicity, we present these two policies in the context of a two-node system; however, the same rationale and analysis applies to systems with multiple nodes.

Consider a 2-node distributed computer system, where the ith node has $m_i \geq 0$ tasks in its queue at time $t = 0$ and each node has the initial queue-size information of the other node. Suppose that the service time (execution time per task), failure time and recovery time of ith node follow exponential distribution with parameters (inverse of the mean), $\lambda_{d_i}, \lambda_{f_i}$ and λ_{r_i}, respectively. The load-transfer delay between the nodes, say delay in transferring L_{ji} tasks from node i to node j, is also assumed to follow an exponential distribution with rate λ_{ji}, which depends on the size of the load L_{ji}. These approximations are supported

in Section 4 by experimental results.

2.1. Definition and analysis of LBP-1

The policy LBP-1 allows a one-time and a one-way load transfer between the nodes and no other balancing action is taken afterwards. At time $t = 0$, both nodes are assumed to be functional and *only* one of the nodes, say node i transfers L_{ji} tasks to node j, where

$$L_{ji} = \lfloor K m_i \rfloor, \tag{1}$$

$K \in [0,1]$, $i, j \in \{1,2\}$ and $i \neq j$, where the user-specified control parameter K is known as LB *gain*. No other load transfer occurs afterwards and each node will process its remaining initial tasks as well as the tasks transferred to it. The optimal policy is to choose K, the sender node i and the receiver node j that will minimize the expected value of the overall completion time of the workload in the system. Our next goal is to characterize the expected value of the overall completion time for a given initial workload and optimize it over gain K. For the rest of this section, without loss of generality, we will suppose that node 1 is sender i.e., at time $t = 0$, node 1 sends $L := L_{21}$ tasks to node 2 as given by (1).

To do so it is necessary that we introduce the notion of *work states* of the distributed system and show its implication in the dynamics of the system. A node may either be in a working mode, indicated by "1," or dead or recovery mode, indicated by "0." Therefore, at any given instant, a 2-node system has 2^2 work states that can be represented as (k_1, k_2), where $(k_1, k_2) \in \{0,1\}^2$. For example, work state $(1,1)$ corresponds to the case when both nodes are working while work state $(1,0)$ means node 1 is working while node 2 is dead. For the ith node, let X_i and Y_i be the random variables representing the time to failure (if the node is initially in a working state) and the time to recovery (if the node is initially in a dead state), respectively. We set $Y_1 = \infty$ if the initial work state of the nodes is $(1,k)$, $Y_2 = \infty$ if the initial work state of the nodes is $(k,1)$, $X_1 = \infty$ if the initial work state of the nodes is $(0,k)$, and $X_2 = \infty$ if the initial work state of the nodes is $(k,0)$. Thus, the random variable $\sigma \triangleq \min(X_1, X_2, Y_1, Y_2)$ represents the time when the first node recovery or node failure occurs.

Next, we use the concept of regenerative processes [12] to obtain a set of difference equations that characterize the expected value of the overall completion time. Let W_i and Z be the random variables representing the time of the first task completion at the ith node and the time of arrival of load L sent from node 1 to node 2, respectively. Define the random variable $\tau \triangleq \min(\sigma, W_1, W_2, Z)$, which represents the time of the first occurrence of any of the above events. The event $\{\tau = s\}$ is termed a *regeneration event* since its occurrence will regenerate queues at time s that have similar statistical properties and dynamics as their predecessors, but with a different initial work state or different workload distribution. For example, with the initial work state $(0,0)$, the occurrence of the event $\{\tau = s\}$ implies that new versions of the queues will emerge at time s with similar dynamics, except that the initial work state has now changed to either $(0,1)$ or $(1,0)$, depending on the occurrence of the events $\{\tau = Y_2 = s\}$ or $\{\tau = Y_1 = s\}$, respectively. Note that the occurrence of the event $\{\tau = W_2 = s\}$ implies that the set of queues that emerge at time $t = s$ would statistically be the same as those at time $t = 0$ if node 2 had one task less than what node 2 had for its initial workload. In summary, upon the occurrence of the event $\{\tau = s\}$ the queueing system re-emerges, or it is regenerated, with a different initial load or working state.

2.1.1 Expected completion time

Let $T^{k_1,k_2}_{M_1,M_2}$ denote the overall completion time, where node 1 has $M_1 := m_1 - L$ tasks and node 2 has $M_2 := m_2$ tasks at time $t = 0$, with L tasks in transit (from node 1 to node 2) given that the system's work state is (k_1, k_2) at time $t = 0$. We can use iterated conditional expectations to write:

$$\begin{aligned} &\mathsf{E}[T^{k_1,k_2}_{M_1,M_2}] = \mathsf{E}\big[\mathsf{E}[T^{k_1,k_2}_{M_1,M_2}|\tau]\big] \\ &= \int_0^\infty \mathsf{E}[T^{k_1,k_2}_{M_1,M_2}|\tau = s] f_\tau(s) ds, \end{aligned} \tag{2}$$

where $f_\tau(t)$ is the probability density function of τ. For $(k_1, k_2) = (1,1)$, we can write

$$\begin{aligned} &\mathsf{E}[T^{1,1}_{M_1,M_2}|\tau = s] = \\ &\sum_{i=1}^{2} \mathsf{E}[T^{1,1}_{M_1,M_2}|\tau = s = W_i]\mathsf{P}\{\tau = W_i\} \\ + &\sum_{i=1}^{2} \mathsf{E}[T^{1,1}_{M_1,M_2}|\tau = s = X_i]\mathsf{P}\{\tau = X_i\} \\ + &\mathsf{E}[T^{1,1}_{M_1,M_2}|\tau = s = Z]\mathsf{P}\{\tau = Z\}. \end{aligned}$$

Now we can utilize the concept of regeneration that was explained earlier in this section to obtain

$$\mathsf{E}[T^{1,1}_{M_1,M_2}|\tau = W_2 = s] = s + \mathsf{E}[T^{1,1}_{M_1,M_2-1}].$$

Therefore,

$$\begin{aligned} &\int_0^\infty \mathsf{E}[T^{1,1}_{M_1,M_2}|\tau = s = W_2] f_\tau(s) ds = \\ &\qquad \mathsf{E}[\tau] + \mathsf{E}[T^{1,1}_{M_1,M_2-1}]. \end{aligned} \tag{3}$$

Similarly, if $\{\tau = X_2 = s\}$ occurs, we obtain

$$\mathsf{E}[T^{1,1}_{M_1,M_2}|\tau = X_2 = s] = s + \mathsf{E}[T^{1,0}_{M_1,M_2}].$$

Interestingly, the occurrence of the event $\{\tau = Z = s\}$ will regenerate the new queues, independently of the original queues, with similar statistical properties with the exception that $Z = \infty\}$. This prompts us to define a new, simpler random variable $\hat{T}^{1,1}_{r_1,r_2}$, which denotes the overall completion time when node i has $r_i \geq 0$ tasks and the system work state is $(1,1)$ at time $t = 0$ while there is no tasks in transit. With this, we can write

$$\mathsf{E}[T^{1,1}_{M_1,M_2}|\tau = Z = s] = s + \mathsf{E}[\hat{T}^{1,1}_{M_1,M_2+L}].$$

Let $\mu^{1,1}_{M_1,M_2} := \mathsf{E}[T^{1,1}_{M_1,M_2}]$. From basic probability, the definition of τ implies that τ is an exponential random variable with rate $\lambda = \sum_{i=1}^{2}\lambda_{d_i} + \sum_{i=1}^{2}\lambda_{f_i} + \lambda_{21}$. Also, $\mathsf{P}\{\tau = W_i\} = \lambda_{d_i}/\lambda$, $\mathsf{P}\{\tau = X_i\} = \lambda_{f_i}/\lambda$ and $\mathsf{P}\{\tau = Z\} = \lambda_{21}/\lambda$. Using (2) and the decomposition based on regeneration principle, we obtain

$$\begin{aligned}\mu^{1,1}_{M_1,M_2} &= \frac{1}{\lambda} + \frac{\lambda_{d_1}}{\lambda}\mu^{1,1}_{M_1-1,M_2} + \frac{\lambda_{d_2}}{\lambda}\mu^{1,1}_{M_1,M_2-1} \\ &+ \frac{\lambda_{f_1}}{\lambda}\mu^{0,1}_{M_1,M_2} + \frac{\lambda_{f_2}}{\lambda}\mu^{1,0}_{M_1,M_2} + \frac{\lambda_{21}}{\lambda}\hat{\mu}^{1,1}_{M_1,M_2+L}.\end{aligned}$$

Similarly, with the initial work state $(k_1,k_2) = (0,1)$, we can characterize $\mathsf{E}[T^{0,1}_{M_1,M_2}]$ by the equation

$$\begin{aligned}\mu^{0,1}_{M_1,M_2} &= \frac{1}{\lambda} + \frac{\lambda_{d_2}}{\lambda}\mu^{0,1}_{M_1,M_2-1} \\ &+ \frac{\lambda_{r_1}}{\lambda}\mu^{1,1}_{M_1,M_2} + \frac{\lambda_{f_2}}{\lambda}\mu^{0,0}_{M_1,M_2} + \frac{\lambda_{21}}{\lambda}\hat{\mu}^{0,1}_{M_1,M_2+L},\end{aligned}$$

where, $\lambda = \lambda_{d_2} + \lambda_{r_1} + \lambda_{f_2} + \lambda_{21}$. In this fashion, we obtain a set of difference equations satisfying

$$\mu = \mathbf{A^{-1}b}, \tag{4}$$

with

$$A = \begin{pmatrix} 1 & -\lambda_{r_2}/\lambda_A & -\lambda_{r_1}/\lambda_A & 0 \\ -\lambda_{f_2}/\lambda_B & 1 & 0 & -\lambda_{r_1}/\lambda_B \\ -\lambda_{f_1}/\lambda_C & 0 & 1 & -\lambda_{r_2}/\lambda_C \\ 0 & -\lambda_{f_1}/\lambda_D & -\lambda_{f_2}/\lambda_D & 1 \end{pmatrix}$$

$$\mathbf{b} = \begin{bmatrix} \frac{1}{\lambda_A} + \frac{\lambda_{21}}{\lambda_A}\hat{\mu}^{0,0}_{M_1,M_2+L} \\ \frac{1}{\lambda_B} + \frac{\lambda_{d_2}}{\lambda_B}\mu^{0,1}_{M_1,M_2-1} + \frac{\lambda_{21}}{\lambda_B}\hat{\mu}^{0,1}_{M_1,M_2+L} \\ \frac{1}{\lambda_C} + \frac{\lambda_{d_1}}{\lambda_C}\mu^{1,0}_{M_1-1,M_2} + \frac{\lambda_{21}}{\lambda_C}\hat{\mu}^{1,0}_{M_1,M_2+L} \\ \frac{1}{\lambda_D} + \frac{\lambda_{d_1}}{\lambda_D}\mu^{1,1}_{M_1-1,M_2} + \frac{\lambda_{d_2}}{\lambda_D}\mu^{1,1}_{M_1,M_2-1} + \frac{\lambda_{21}}{\lambda_D}\hat{\mu}^{1,1}_{M_1,M_2+L} \end{bmatrix}$$

where,

$$\begin{aligned}\lambda_A &= \lambda_{r_1} + \lambda_{r_2} + \lambda_{21}, \\ \lambda_B &= \lambda_{d_2} + \lambda_{r_1} + \lambda_{f_2} + \lambda_{21}, \\ \lambda_C &= \lambda_{d_1} + \lambda_{f_1} + \lambda_{r_2} + \lambda_{21}, \\ \lambda_D &= \lambda_{d_1} + \lambda_{d_2} + \lambda_{f_1} + \lambda_{f_2} + \lambda_{21} \\ \mu &= \left[\mu^{0,0}_{M_1,M_2}, \mu^{0,1}_{M_1,M_2}, \mu^{1,0}_{M_1,M_2}, \mu^{1,1}_{M_1,M_2}\right]^T.\end{aligned}$$

Observe that (4) holds for $M_1, M_2 \geq 1$; thus, it is required that we compute the solution to the initial conditions $\mu^{k_1,k_2}_{0,0}$, $\mu^{k_1,k_2}_{0,1}$ and $\mu^{k_1,k_2}_{1,0}$, as well as $\hat{\mu}^{k_1,k_2}_{M_1,M_2+L}$. The first three quantities can be computed in a similar fashion but with a fewer possible events. For example, $\mu^{k_1,k_2}_{0,0}$ can be computed by setting $W_1 = \infty$ and $W_2 = \infty$, since there are no tasks to be executed. Finally, using arguments based on regeneration, it turns out that $\hat{\mu}^{k_1,k_2}_{M_1,M_2+L}$ can also be written as $\hat{\mu} = \mathbf{\hat{A}^{-1}\hat{b}}$ (we omit details), where $\mathbf{\hat{A}}$ is similar to $\mathbf{A}$ with $\lambda_{21} = 0$, and $\mathbf{\hat{b}}$ is similar to $\mathbf{b}$ with $\lambda_{21} = 0$. In this case, the initial condition $\hat{\mu}^{k_1,k_2}_{0,0} = 0$.

We make the final observation that swapping the roles of the sender and receiver nodes does not change the analysis detailed above. Therefore, our earlier assumption that node 1 was the sending node can now be relaxed. With the above complete solution of the mean overall completion time, the optimal gain value K and the sender/receiver pair (i.e., which node should be sending tasks to the other) that will minimize $\mu^{1,1}_{M_1,M_2}$ can be found. This allows the determination of the optimal implementation of LBP-1. Numerical and experimental results are discussed in Section 4.

2.1.2 Probability distribution function of the completion time

Let $p^{k_1,k_2}_{M_1,M_2}(t) := \mathsf{P}\{T^{k_1,k_2}_{M_1,M_2} \leq t\} = \mathsf{E}[1_{\{T^{k_1,k_2}_{M_1,M_2} \leq t\}}]$, where, 1_A is an indicator function for the event A. By using the smoothing property of expectations and exploiting the concept of regeneration, it is straight forward to show that

$$\dot{\mathbf{p}} = \mathbf{A_1 p} + \mathbf{B_1 u}, \tag{5}$$

where $\mathbf{A_1} = \begin{pmatrix} -\lambda_D & \lambda_{f_1} & \lambda_{f_2} & 0 \\ \lambda_{r_1} & -\lambda_B & 0 & \lambda_{f_2} \\ \lambda_{r_2} & 0 & \lambda_C & \lambda_{f_1} \\ 0 & \lambda_{r_2} & \lambda_{r_1} & \lambda_A \end{pmatrix}$, $\mathbf{B_1} = \mathbf{I_4}$, the 4×4 identity matrix,

$$\mathbf{u} = \begin{bmatrix} \lambda_{d_1} p^{1,1}_{M_1-1,M_2}(t) + \lambda_{d_2} p^{1,1}_{M_1,M_2-1}(t) + \lambda_{21}\hat{p}^{1,1}_{M_1,M_2+L}(t) \\ \lambda_{d_2} p^{0,1}_{M_1,M_2-1}(t) + \lambda_{21}\hat{p}^{0,1}_{M_1,M_2+L}(t) \\ \lambda_{d_1} p^{1,0}_{M_1-1,M_2}(t) + \lambda_{21}\hat{p}^{1,0}_{M_1,M_2+L}(t) \\ \lambda_{21}\hat{p}^{0,0}_{M_1,M_2+L}(t) \end{bmatrix}$$

$\mathbf{p} = \left[p^{1,1}_{M_1,M_2}(t), p^{0,1}_{M_1,M_2}(t), p^{1,0}_{M_1,M_2}(t), p^{0,0}_{M_1,M_2}(t)\right]^T$, and $\hat{p}^{k_1,k_2}_{M_1,M_2}(t) = E[1_{\{\hat{T}^{k_1,k_2}_{M_1,M_2} \leq t\}}]$, which results in a similar set of equations as in (5) but with $\lambda_{21} = 0$. Clearly, solving (5) requires the explicit and *a priori* solution of $\hat{p}^{k_1,k_2}_{M_1,M_2}(t)$. Indeed, by using the initial condition $\hat{p}^{k_1,k_2}_{0,0}(t) = 1$, we can iteratively solve for $\hat{p}^{k_1,k_2}_{M_1,M_2}(t)$ and use it in (5).

2.2. Policy LBP-2

In this policy, each node initially executes load balancing at time $t = 0$ without considering the future possibility of node failures and subsequent recoveries of any node in the distributed system. Each node is assumed to be equipped with a backup system that can only send or receive tasks, as explained in more detail in Section 3. Subsequently, upon the occurrence of every node failure, the backup system of the failing node executes another balancing action to compensate for the time that will be wasted till the node recovers. As before we will assume that at time $t = 0$ each node in the system has knowledge of the initial workload of all other nodes.

Based on the processing rates of the nodes, the initial LB action is taken to achieve an "approximately" uniform division of the total workload of the whole system among all the nodes assuming that all nodes will remain functional. It has been shown in our previous works that in a delay infested system, the division of the load based on the nodes' processing speeds alone does not yield a minimal average overall completion time [8, 10]. Specifically, at time $t = 0$, every node computes its excess load by comparing its load to the average over all load of the system. For example, the excess load at node j is given by

$$L_j^{\text{excess}} = \left(m_j - \frac{\lambda_{d_j}}{\sum_{k=1}^{n} \lambda_{d_k}} \sum_{l=1}^{n} m_l \right)^+,$$

where $(x)^+ \triangleq \max(x, 0)$. This is a more plausible way to calculate the excess load of a node in a heterogeneous computing environment as compared to our earlier method that did not consider the processing speeds of the nodes [8–11]. Clearly, with the inclusion of the processing speeds of the nodes, a slower (faster) node has a larger (smaller) excess load than what it would have had under our earlier definition. The excess load is then partitioned among the $n-1$ nodes by assigning a larger portion to a node with smaller relative load. Such a partition is obtained by using fraction p_{ij} given as

$$p_{ij} = \begin{cases} \frac{1}{n-2}\left(1 - \frac{\lambda_{d_i}^{-1} m_i}{\sum_{l \neq j} \lambda_{d_l}^{-1} m_l}\right), & n \geq 3 \\ 1, & n = 2 \end{cases} \quad (6)$$

with $p_{jj} = 0$. Indeed, it is easy to check that $\sum_{l=1}^{n} p_{lj} = 1$. Therefore, these fractions form a partition based not only on the amount of loads at the receiver nodes, as in [8–11], but also use the processing speeds of the receiver nodes. Consequently, $p_{ij} L_j^{\text{excess}}$ uniformly divides the excess load of node j among all other nodes. But, due to the inherent delay in load transfer between nodes, such a uniform partition (which, in comparison to LBP-1, does not involve a LB gain, K) will not guarantee the minimum possible overall completion time [8]. To remedy this, the user-defined gain $K \in [0, 1]$ will be used to calculate the actual portion of the load to be dispatched to node i from node j. The load to be transferred from node j to node i therefore becomes

$$L_{ij} = \lfloor K p_{ij} L_j^{\text{excess}} \rfloor. \quad (7)$$

In LBP-2, we choose the LB gain K that will minimize the average overall completion time under the hypothesis that the nodes will never fail. This optimization problem has already been solved using the concept of regeneration [10, 11] and it has been observed that due to the presence of random delay the LB gain is not unity.

Now suppose node j fails at time $t > 0$, which means that in average node j will be out of work for $\lambda_{r_j}^{-1}$ amount of time. Therefore, knowing that the distributed system has already been balanced to optimize the average overall completion time, the failure of node j results in an accumulation, on average, of $\lambda_{d_j}/\lambda_{r_j}$ of unattended tasks (which is the average recovery time multiplied by the processing speed) during its recovery period. Hence, node j should be allowed another balancing opportunity to transfer a load $\left(\frac{\lambda_{d_i}}{\sum_{k=1}^{n} \lambda_{d_k}}\right)\left(\frac{\lambda_{d_j}}{\lambda_{r_j}}\right)$ tasks to node i. But, the steady-state probability of any node i to be working is given by $\left(\frac{\lambda_{r_i}}{\lambda_{f_i}+\lambda_{r_i}}\right)$. Therefore, at every failure instant of node j, the node should send L_{ij}^F number of tasks to node i $(i \neq j)$, where

$$L_{ij}^F = \lfloor \left(\frac{\lambda_{r_i}}{\lambda_{f_i} + \lambda_{r_i}}\right)\left(\frac{\lambda_{d_i}}{\sum_{k=1}^{n} \lambda_{d_k}}\right)\left(\frac{\lambda_{d_j}}{\lambda_{r_j}}\right) \rfloor. \quad (8)$$

3. Implementation of LB Policies

We have developed a distributed computing system architecture to validate our stochastic model and to determine the performance of the system in case of recoverable failures. The system is formed by a certain number of CEs that are processing jobs in a cooperative environment. Each CE contains a back-up system that is saving the context of the application running on the node; therefore, if a node randomly fails (or "goes down") and recovers (or "goes up") after a certain random time, then it can resume its work by simply reloading the data saved by the back-up process. The software architecture of the distributed system is divided in three layers: application, communication and load-balancing/failure.

The application layer software developed consists of the application that is being processed in parallel in the nodes forming the system. The application used to illustrate the load-balancing process is matrix multiplication, where one task is defined as the multiplication of one row by a static matrix duplicated on all nodes. To produce a certain randomness in the processing time of the tasks, we defined the arithmetic precision of each element in a row to be picked randomly from an exponential distribution. This also randomizes the sizes of the tasks. In addition, the application layer manages the shared data that is being transferred between the nodes and produces the state information of each node, which is transferred to the remaining nodes of the system.

The communication layer handles all the data and state information transfers made by the system. This layer receives the shared data from the application layer and transfers it to its peer layer on the receiving nodes. The UDP transport protocol is employed to perform all the state information exchanges among the nodes. The state information consists of the information about the current queue size, the node computational power, and other local information that may be relevant to the load-balancing policy in use. The sizes of the information state packets are between 20 and 34 bytes, depending on the policy employed. On the other hand, the TCP transport protocol is used to transfer the application data between the CEs. In this case, the size of the data packets depend on the number of tasks to be transferred and on the particular realization of each randomly generated task. In addition, the communication layer receives the data from the application layer and formats the data before transferring it to the receiving nodes. Therefore, after receiving data from the network, this layer performs inverse operation to the received packets before delivering the data to the application layer.

The load-balancing/failure layer implements the load-balancing policies and the failure/recovery actions presented in this paper. The load-balancing layer is implemented in software using a multi-threaded process, where the POSIX-threads programming standard was used. One of the threads is in charge of the load-balancing policy and it is initiated, using an scheduler, at a predefined or at a calculated amount of time depending on the policy used. Therefore, the thread determines the portion of the tasks to be transferred to every node in the system, if applicable, and accesses the shared data to define the tasks to be sent by the communication layer. A different thread was coded for the back-up node, in order to implement the failure policy. This thread computes the amount of tasks to be sent in case of a non-catastrophic failure, and also accesses the shared data to define which tasks must be transferred. The mentioned software is running identical copies on each node of the system; so, the system's load-balancing action (i.e., the instants of balance and the number of tasks to transmit from one node to the others) is distributed because of the local decision that each local node takes, based on the state information of the system that was exchanged during the synchronization events.

Finally, the software platform was coded in ANSI-C over UNIX-based systems, and it has been successfully tested on SPARC processor-based machines running Solaris operating system and on IA-32 processor-based computers running both Linux and Microsoft Windows operating systems.

4. Results

To test the performance of the proposed LB policies, a comparative set of experiments was performed between these two policies, and the results were also compared to the theoretical predictions and MC simulations. The experiments were conducted over the EECE infrastructure-based IEEE 802.11b/g network at the University of New Mexico in normal work-days of operation. In our experiments we employed a 1 GHz Transmeta Crusoe processor-based computer (node 1) and a 2.66 GHz Intel P4 processor-based computer (node 2).

Firstly, experiments were conducted to estimate the processing speed of the nodes as well as to calculate the average delay in the network in the context of our application, i.e., to estimate processing time per task, and the delay incurred in transferring tasks. To recall, one task is an array of numbers and the processing time of a task is the time to multiply it with a static matrix of fixed size. As we indicated earlier in Section 3, the task sizes are generated randomly and independently, according to a common distribution, which will, in turn, result in independent and identically-distributed execution times. The empirically calculated probability density functions (pdf's) of the processing time per task of each node is shown in Fig. 1. Clearly, each empirical pdf can be approximated with an exponential pdf, where the processing rates of node 1 and node 2 are 1.08 tasks per second and 1.86 tasks per second, respectively. Also, from Fig. 2, we see that the average transfer delay depends on the load size (in terms of number of tasks), and grows linearly with the size. Further, the transfer delay per task can be approximated by an exponentially distributed random variable with mean 0.02s. Although there is a slight shift observed in the

pdf of the delay, in our approximation we maintained the exponential form of the pdf and compensated for the shift through the choice of the exponential parameter. This approximation simplifies the analysis (as the analysis in Section 2 assumes an exponential pdf) while capturing the random, load-dependent nature of the delay.

In the experiments, the nodes are assumed to fail and recover independently and randomly following an exponential pdf. In order to achieve this in our experiments, we have coded a process that dynamically generates failure instants and sends signals, at all such failure instants, to the application layer ordering it to stop executing tasks. Also, at every failure instant, the same process generates a recovery time and waits for that amount of time before sending a new signal to the application layer ordering it to resume the execution of tasks. In this paper, the average failure time for both nodes is 20s, while the average recovery times of nodes 1 and 2 are 10s and 20s, respectively. Clearly, node 1 is expected to be available for more time than node 2.

Initially, experiments were performed to assess the performance of LBP-1. Node 1 was assigned 100 tasks, while node 2 was assigned 60 tasks. The load-balancing was performed at time $t = 0$ according to (1), and the load transfer was made from node 1 to node 2 using different values for K. The average overall completion time was computed for each case and the theoretical, MC-simulated, and experimental results are shown in Fig. (3). For comparison, the results for the no-failure case (when the failure rate is set to zero) are also shown. From the theoretical curves, it can be seen that the minimum average overall completion time occurs at $K = 0.35$, while the minimum occurs at $K = 0.45$ for the no-failure case. Note that in the former case, node 1 transfers 35 tasks to node 2, while in the latter case it transfers 45 tasks to node 2. In both cases, the average overall completion time is minimized when node 1 transfers tasks to node 2, which has a higher processing speed. But, in presence of node failure, the amount of transfer is reduced because node 2 is now less reliable. Intuitively, we can state that the optimal K in case of node failure will always be less than the optimal K for the no failure case.

Next, we conducted experiments for LBP-2. The initial workload distribution was 100 and 60 tasks at nodes 1 and 2, respectively. Note that from (8) the amount of load to be transferred at every failure instant happens to be a constant, depending on parameters that are fixed in this paper. The optimal gain K for the initial LB (which does not account for node failure) was found to be 1 [11]. Using this optimal gain, the average overall completion time was calculated using 60 realizations of the experiments and was found to be 109.17s. We also performed the MC simulation under the same initial set-up for the LBP-2, and the average overall completion time turned out to be 112.43s using 500 realizations. Note that in the case of LBP-1 (see Fig. (3)), the minimum average overall completion time is 117s, which is greater than the value obtained for the LBP-2. This is expected since LBP-1 takes a preemptive action in the beginning by predicting the failure instants, while LBP-2 avoids the prediction by taking an action of transferring tasks only when failures occur. In order to compare the dynamics of each policy, we show in Fig. 4 the actual queues of each node, under one realization of the experiments performed for LBP-1 and LBP-2. We can observe that the longer flat portions of the queues corresponds to the recovery times of the nodes. Note also the downward (upward) jumps in the queues under LBP-2, which correspond to the action of transferring (receiving) tasks after every failure instant.

In order to compare the performance of the two policies in a small delay environment, we conducted experiments with different initial workloads under similar channel conditions. To achieve this, we first estimated the average delay per task using channel probing experiments, and the estimate was found to be 0.02s. The theoretical model was then used to calculate the optimal gains as well as the sender/receiver pairs for LBP-1 for each initial workload listed in Table 1. It was found that if the initial load of node 1 is smaller than the initial load of node 2, then the load transfer has to be made from node 2 to node 1; otherwise, node 1 has to be the sender node. Using the respective optimal gains and the sender/receiver pairs, the actual experiments were conducted and the average overall completion time was estimated using 20 realizations for each initial workload. In Table 1 we also list the theoretically calculated average overall completion time under the no-failure case. To conduct the experiments under LBP-2, we first calculated the optimal gains K using our previously reported theoretical model [11] (which does not consider node failure and recovery). In Table 2 we have listed the results obtained from our MC simulations and the real-time experiments. We can see from both Tables that LBP-2 outperforms LBP-1 in all cases.

We also studied the performance of LBP-1 and LBP-2 under different amount of delays in the channel. The results are shown in Table 3, and it can be seen that when the average delay per task is bigger than 1s, LBP-1 results in a smaller average overall completion time than LBP-2. This is attributable to the amount of time needed in making load transfers at every failure

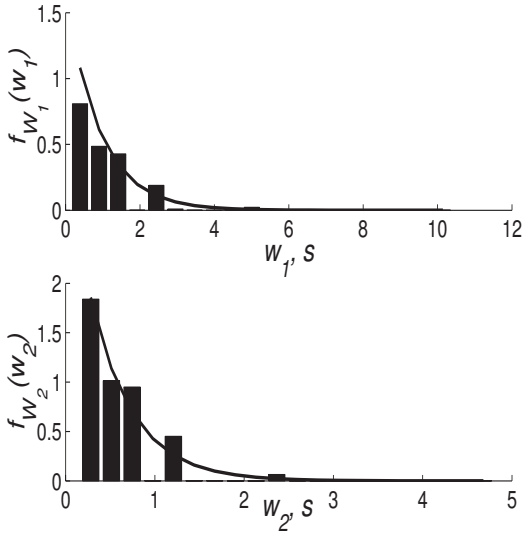

Figure 1. Empirically estimated pdfs of the processing time per task for node 1 (top) and node 2 (bottom) as well as their exponential approximations (solid curves).

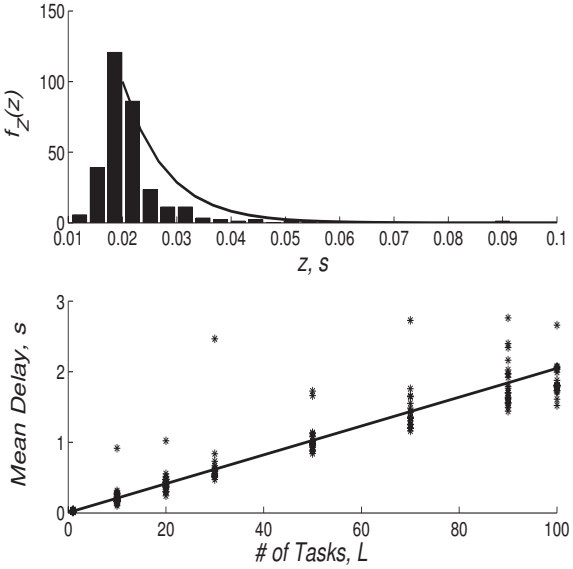

Figure 2. Top: Empirical pdf, calculated from 30 realizations of the transfer delay per task. Their exponential approximations (solid curves) are also shown. Bottom: A linear approximation of the empirically calculated mean delay as a function of the number of tasks transferred between nodes, calculated from 30 realizations per task. The dots represent the actual delays found in the 30 realizations for each number of tasks.

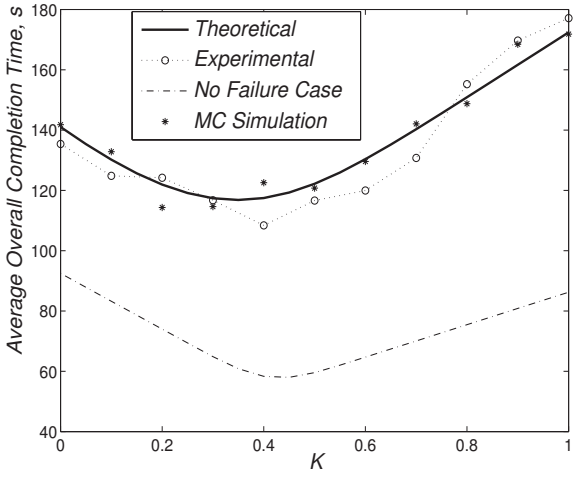

Figure 3. The average overall completion time as a function of the LB gain K for the LBP-1.

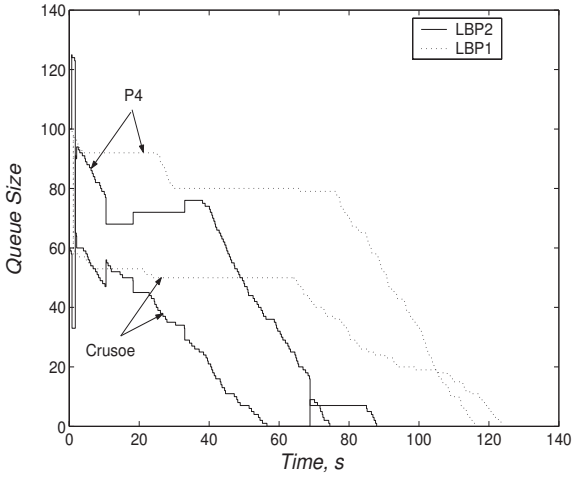

Figure 4. A realization of the queues obtained from the experiments conducted for LBP-1 and LBP-2.

Table 1. Experimental results for LBP-1 using the theoretically determined optimal gain.

Initial Workload (m_1,m_2)	Optimal Gain K_{opt}	Average Overall Completion Time (s)		
		Node Failure		Without Node Failure
		Theo. Pred.	Exp. Result	
(200,200)	0.15	274.95	264.72	141.94
(200,100)	0.35	210.13	207.32	106.93
(100,200)	0.15	210.13	229.19	106.93
(200,50)	0.5	177.09	172.56	89.32
(50,200)	0.25	177.09	215.66	89.32

Table 2. Experimental and simulation results for the LBP-2.

Initial Workload (m_1,m_2)	Initial LB Gain K	Average Overall Completion Time (s)	
		MC Simulation	Exp. Result
(200,200)	1.00	277.9	263.4
(200,100)	1.00	202.4	188.8
(100,200)	0.80	203.07	212.9
(200,50)	1.00	170.81	171.42
(50,200)	0.95	189.72	177.6

instant in case of the LBP-2, which may result in idle times at the receiving node while it waits for the load to arrive. On the other hand, the LBP-1 only makes a one-time transfer at the beginning of workload execution. Therefore, when the time needed to transfer the tasks from one node to the other is in the order of the mean recovery time of the sender node, the LBP-1 performs better than the LBP-2.

Table 3. Performance of the LBP-1 and the LBP-2 under different network delays.

Average Delay Per Task (s)	Calculated Average Overall Completion Time (s)	
	LBP-1	LBP-2
.01	116.82	112.43
0.5	117.76	115.94
1	120.99	122.25
2	127.62	133.02
3	131.64	142.86

Finally, we used (5) to compute $p^{1,1}_{M_1,M_2}(t)$ under LBP-1 by using K that minimizes the average overall completion time. The average transfer delay per task was assumed to be 0.02s. As an example, in Fig. 5 we present the cumulative distribution function for the overall completion time for the initial workloads: $(50, 0)$ and $(25, 50)$.

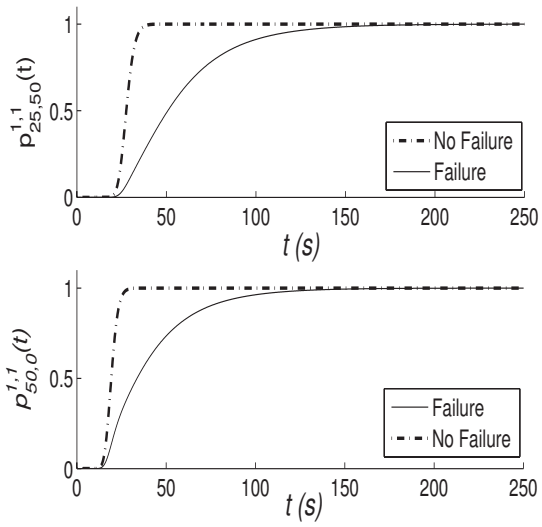

Figure 5. The cumulative distribution function of the overall completion time in LBP-1. The upper figure shows the case of an initial workload of $(50, 0)$, while the lower figure is for an initial workload of $(25, 50)$.

5. Conclusion

We have presented two load-balancing policies for a distributed computing system in presence of node failure and recovery. The first policy, LBP-1, preemptively utilizes the statistical information about the failure and recovery processes to adjust the load-balancing gain K to an optimal value, thereby minimizing the mean of the overall completion time of the total workload. The probabilistic analysis of this policy was carried out using a regeneration-theory-based analytical approach. The performance metrics considered through out the study included the mean overall completion time and to a lesser extent its probability distribution function. We have observed that under LBP-1, as the failure rates of nodes increase (while holding other parameters fixed), the minimum achievable average overall completion time is obtained by reducing the strength of balancing. We also observed that the presence of node failure and recovery warrants the use of a reduced load-

balancing gain K compared to the case of no-failure case. This conclusion is not dissimilar to our earlier results for the optimal gain in presence of stochastic delay in load transfer [9–11]. In both the case, the presence of uncertainty (viz., node faulre/recovery or random delay) calls for an attenuation in the level of load-balancing action.

On the other hand, the second policy, LBP-2, does not predict the node failure but instead it takes due action at every failure instant in order to distribute its uncompleted workload during its recovery time. Comparative study of the two policies for different initial workloads and different average delays indicated that when the network delays are small compared to the average recovery times, LBP-2 outperforms LBP-1. In contrast, when the network delays are large compared to the average recovery times, the time wasted in transferring tasks at every failure instant adversely affects the average overall completion time. Therefore, it is advantageous to use the LBP-1 instead of the LBP-2 in such situations.

We make the final remark that if new external workloads arrive regularly to the distributed system at random instants, one can continue to utilize the rationale of analogues to LBP-1 and LBP-2 to develop dynamic versions of them. One simplified approach is to execute load-balancing episodes (either LBP-1 or LBP-2) at every external arrival of new workloads.

Acknowledgment

This work is supported by the National Science Foundation under Information Technology Research (ITR) grants No. ANI-0312611 and ANI-0312182.

References

[1] *http://setiathome.ss.berkeley.edu/*

[2] H. M. Lee, S. H. Chin, J. H. Lee, D. W. Lee, K. S. Chung, S. Y. Jung and H. C. Yu, "A Resource Manager for Optimal Resource Selection and Fault Tolerance Service in Grids," in the *Proc. 4th IEEE International Symposium on Cluster Computing and the Grid*, Chicago, Illinois, USA 2004.

[3] M. Litzkow, M. Livny and M. Mutka, "Condor - A hunter of idle Workstations," in the *Proc. 8th International Conference of Distributed Computing Systems*, pp. 104–111, June 1988.

[4] V. Subramani, R. Kettimuthu, S. Srinivasan and P. Sadayappan, "Distributed Job Schedulingon Computational Grids Using Multiple simultaneous Requests," *Proc. 11th IEEE International Sumposium on High Performance Distributed Computing HPDC-11*, 2002 (HPDC'02), Edinburgh, Scotland, July 24–26, pp. 359–368, 2002.

[5] S. Choi, M. Balik, and C. S. Hwang "Volunteer Availability based Fault Tolerant Scheduling Mechanism in Desktop Grid Computing Environment," *Proc. 3rd IEEE International Symposium on Network Computing and Applications*, Boston, Massachusetts, August 30th - September 1st, pp. 366–371, 2004.

[6] E. Gelenbe, D. Finkel, and S. K. Tripathi, "On the availability of a distributed computer system with failing components," *ACM SIGMETRICS Performance Evaluation Review*, vol. 13, Issue 2, pp. 6–13, 1985.

[7] R. Sheahan, L. Lipsky, and P. Fiorini, "The Effect of Different Failure Recovery Procedures on the Distribution of Task Completion Times," *IEEE DPDNS05*, Denver CO, April 2005.

[8] M. M. Hayat, S. Dhakal, C. T. Abdallah, J. Chiasson, and J. D. Birdwell, *Dynamic time delay models for load balancing. Part II: Stochastic analysis of the effect of delay uncertainty.* In Advances in Time Delay Systems, Springer Series on Lecture Notes in Computational Science and Engineering, (Keqin Gu and Silviu-Iulian Niculescu, Editors), vol. 38, pp. 371–385, Springer: Berlin, 2004.

[9] 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 Conference on Decision and Controls (CDC 2003)*, Maui, Hawaii, pp. 5128–5134, Dec 2003.

[10] S. Dhakal, M. M. Hayat, J. Ghanem, C. T. Abdallah, H. Jerez, J. Chiasson, and J. D. Birdwell, *On the optimization of load balancing in distributed networks in the presence of delay.* In Advances in Communication Control Networks, Springer series Lecture Notes in Control an Information Sciences (LCNCIS), (S. Tarbouriech, C. T. Abdallah, and J. Chiasson, Editors) LNCSE vol. 308, pp. 223–244, Springer-Verlag, 2004.

[11] S. Dhakal, M. M. Hayat, J. Ghanem, and C. T. Abdallah "Load Balancing in Distributed Computing Over Wireless LAN: Effects of Network Delay," *Proceedings of the IEEE Wireless Communication & Networking Conference (WCNC-2005)*, New Orleans, LA, vol. 3, pp. 1755–1760, March 13–17, 2005.

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