Dynamic load balancing for robust distributed computing in the presence of topological impairments

Majeed M. Hayat

Jorge E. Pezoa

David Dietz

Sagar Dhakal

Keywords

Distributed Computing; Load Balancing; Network Robustness; WMD Attack; Reliability; Queuing Theory; Renewal Theory

Abstract

The purpose of any distributed computing system (DCS) is to offer a flexible, reliable and powerful computing platform. With the advances in mobile computing, wireless communications and sensor networks, DCSs have emerged in new applications such as wireless sensor networks (WSNs), military battlefield awareness, surveillance and threat detection, to name a few. These new application areas introduce new challenges to DCSs when operated or deployed in harsh or threat-prone environments. For instance, in WSNs deployed in a military battlefield, the computing elements (CEs) of a DSC join and leave the DCS at any time in a stochastic fashion. More generally, factors such as limited or intermittent communication resources, CEs' power constraints or long-term physical damage of the CEs, can result in random topological changes in the DCS, which, in turn, can severely degrade their performance and reliability. Many of these factors can be attributable to physical attacks on our information infrastructure, of which weapons of mass destruction (WMD) is an important example. This observation has triggered government agencies, such as the Defense Threat Reduction Agency, to launch research initiatives in network science to understand the extent of damage that can be inflicted upon networks in the event of attacks and also to develop strategies to increase the robustness of networks when threat is present. In this article, we review modern dynamic load balancing (DLB) techniques and their mathematical stochastic models that can be exploited by DCS developers to increase the DCS's robustness to random topological changes, and at the same time, to efficiently use the available computing resources of the system in the presence of communication uncertainty and CE dysfunction. Two scenarios are considered: one where CEs can fail and recover at random instants and another where CEs can fail permanently. Under the first scenario we look for minimizing the average response time of a given application. In the second scenario the goal is to maximize the probability of successfully running an entire application. DLB policies are tested using a small-scale DCS environment and compared to

theoretical predictions as well as results from Monte-Carlo simulations. The mathematical probabilistic model presented here for network performance is general and can be applied to a broad class of networks and applications.

A new class of applications, based on sensor networks (SNs), has emerged in recent years. Examples of these applications are habitat monitoring, intrusion detection, defense, military battlefield awareness, structural health monitoring, and scientific exploration. These applications share a particular feature, namely the necessity of collaboration among the sensors. Consequently, the idea of performing distributed computing (DC) naturally appears in a SN environment. In fact, DC is being performed in wireless sensor networks (WSNs) in order to reduce the energy consumption of the set of sensors, make efficient use of network bandwidth, achieve desired quality of service, and reduce the response time of the entire system. These new applications have generated challenging scenarios, and the resource allocation solutions developed for traditional distributed computing systems (DCSs) are not appropriate under these new scenarios [1]. For instance, when DC is performed in a WSN, classical assumptions on node and communication-link reliability are no longer valid due to: (i) WSNs are typically deployed in harsh environments where sensors are prone to fail catastrophically; and (ii) in order to save energy, sensors are allowed to turn themselves off at any time. In addition, assuming that the cost of transmitting data among the sensors is negligible or deterministic is not valid either. Also, large-scale donation-based distributed infrastructures, such as peer-to-peer networks and donation grids, exhibit a similar kind of behavior; the topology of the DCS changes over time as computing elements (CEs) enter to or depart from the DCS in a random fashion. Furthermore, any short term or long term damage that can be inflicted to the CEs in these infrastructures adds to this dynamic behavior. Clearly, the new scenarios for DC demand for solutions that can adapt to both fluctuations in the workload and changes in the number of CEs available in the DCS.

We review here modern resource reallocation techniques that are effective in modern dynamic DCSs [2],[3],[4],[5]. In particular, we describe dynamic load balancing (DLB) policies that can be used to improve the performance of a DCS in the presence of random topological changes. These policies simultaneously attempt to improve the robustness of the DCS and to efficiently use the available CEs in the system. In this article we have considered two different performance metrics: the average response time of a software application and the probability of successfully executing an application. These metrics are analytically modeled using the novel concept of stochastic regeneration. Our model takes into account the heterogeneity in the computing capabilities of the CEs, the random failure and recovery times of the CEs and the random transfer times associated to communication network. Based upon this model, we devise DLB policies that optimize these two performance metrics. First, we discuss DLB policies suitable for scenarios where CEs can fail and recover at random instants of time, as in the case of DC in WSNs or donation-based DCSs. Then, we analyze policies for scenarios where CEs can fail permanently, which model long term physical

damages like those inflicted by weapons of mass destruction (WMD). Our theory is supported by Monte-Carlo (MC) simulations and experimental results collected from a small-scale testbed DCS.

The load-balancing problem in distributed computing

Large, time-consuming applications can be processed by a DCS in a parallel fashion. To this end, applications have to be divided into smaller units, called modules or tasks, which can be processed independently at any CE of the system. Tasks have to be intelligently allocated onto the nodes in order to efficiently use the computing resources available in the DCS. This task allocation is referred to in the literature as load balancing (LB). LB strategies can be divided into static and dynamic, centralized or decentralized, and sender-initiated or receiver-initiated [1]. Static LB is performed before the actual execution of the application in the system. At compiling-time and based upon statistics of the DCS, the compiler accordingly divides the application into several tasks, which are allocated onto the CEs upon the execution of the application. In dynamic load balancing (DLB), both applications and LB algorithms are being executed in the DCS. The DLB algorithm continuously monitors the queue-length of the CEs, and upon the detection of an imbalance, it triggers the reallocation of tasks among the CEs. When the CEs are prone to fail, the LB algorithm must monitor also the working or failed state of the CEs. In this article we focus on decentralized DLB policies because, in general, DLB policies outperform static LB policies in DCSs where both the system workload and the number of functioning CEs dynamically change with time [1].

In order to optimize a given performance metric, any DLB policy has to answer the following fundamental questions at each CE: (i) *When the jth CE has to trigger a LB action?* (ii) *How many tasks have to be processed at the jth CE?,* and (iii) *How many tasks have to be reallocated from the jth CE to the other CEs?* The first question is answered by comparing the load at the *j*th CE with the average load in the system. To this end, information about the number of tasks queued at each CE must be collected by the *j*th CE. We denote by $\hat{Q}_{k,j}(t)$ the number of tasks queued at the *k*th CE as perceived by the *j*th CE at time *t*. Based upon this information, the *j*th CE computes its excess load at time *t*, denoted as $L_j^{ex}(t)$, using the formula:

$$L_j^{ex}(t) = Q_j(t) - \frac{\Lambda_j}{\sum_{l=1}^{n} \Lambda_l} \sum_{l=1}^{n} \hat{Q}_{l,j}(t) . \qquad (1)$$

A positive value for $L_j^{ex}(t)$ means that the *j*th CE is overloaded compared to the average load in the system, and as a consequence, a LB action must be triggered at time *t*. Note that the Λ_j's parameters in (1) can be defined in several

ways in order to establish different balancing criteria. If the Λ_j's are associated with the processing speed of the CEs then the balancing criterion is determined by the relative computing power of the nodes. Alternatively, if the Λ_j's are associated to the failure rates of the nodes then the balancing criteria is determined by the reliability of the CEs.

The second question is answered by employing (1) again. When the excess load at the *j*th CE is positive, the amount of tasks to be processed locally by the *j*th CE is given by the difference between its current and its excess load. The remaining tasks should be reallocated to other CEs. Otherwise, if $L_j^{ex}(t) \leq 0$ then the *j*th CE has to process all its tasks.

Finally, the third question is answered by computing which CEs are underloaded compared to the average load in the system, as is perceived by the *j*th CE. These underloaded processors are CEs candidates to receive tasks. The number of tasks that each candidate CE receives is denoted by $p_{jk} L_j^{ex}(t)$, where p_{jk} is the partition of the excess load at the *j*th CE assigned to the *k*th candidate receiving CE. Partitions can be defined in several ways: evenly, according to the relative excess load of each candidate CE, etc. In general, these partitions may not be effective and must be adjusted in order to compensate for the effects of the random communication times. Thus, the load to be transferred from the *j*th to the *k*th CE must be adjusted according to what is called the LB gain, which is denoted as K_{jk}. Finally, the number of tasks to be transferred from the *j*th to the *k*th CE is given by: $L_{jk}(t) = \lfloor K_{jk} p_{jk} L_j^{ex}(t) \rfloor$, where $\lfloor x \rfloor$ is the greatest integer less than or equal to *x*.

The LB gains are parameters that ultimately define the number of tasks to reallocate to each CE. We optimally select the value of such gains so that we can minimize the average response time of an application or maximize the probability of successfully serving an application. In order to properly optimize these metrics, first we need a precise model for the response time of an application. The response time is a complex random variable, which combines the randomness of the task processing times at the CEs, the failure and recovery times of the CEs and the task transfer times in the communication network.

Stochastic modeling of the response time

The idea of stochastic regeneration in DCSs is thoroughly reviewed; to do so, we freely draw from our earlier published works [2],[3],[4]. Our novel approach based upon stochastic regeneration allows us to obtain recurrence equations that characterize both the average response time of an application and the probability of successfully serving an application [2],[3],[4]. The key idea is to introduce a special random variable, called the regeneration time, τ, which is defined as the minimum of the following six random variables: the time to the *first* task service by any CE, the time to the *first* occurrence of failure at any CE, the time to the *first* occurrence of recovery at any CE, the time to the *first* arrival of a queue-length information at any CE, the time to the *first* detection of a failure at any CE,

or the time to the *first* arrival of a group of tasks at any CE. The key property of the regeneration time is that upon the occurrence of the regeneration event {τ=*s*}, a *fresh* copy of the original stochastic process (from which the random response time is defined) will emerge at time *s,* with a *new* initial system configuration that transpires from the regeneration event.

In order to appreciate how the idea of regeneration works, let us consider the following simplified example. For a two-node DCS let the following random variables W_j, X_j, Y_j, and Z_{jk} denote the service time of a task at the *j*th CE, the failure time of the *j*th CE, the recovery time of the *j*th CE, and the task transfer time from the *j*th to the *k*th CE, respectively, with $j,k \in \{1,2\}, j \neq k$. Thus, the regeneration time is precisely defined as, $\tau = \min(W_1, W_2 X_1, X_2, Y_1, Y_2, Z_{12}, Z_{21})$. Let us denote by $T^{\mathbf{I},\mathbf{F}}_{m_1,m_2}(t_b;\mathbf{C})$ the random response time of an application that comprises m_1+m_2 tasks, where t_b denotes the balancing instant and ***I***, ***F***, and ***C*** are vectors that monitor the number of tasks in the CEs, the working state of the CEs and the number of tasks being transferred in the network, respectively. By exploiting the random variable τ, we can compute the average response time of the application as

$$\mathrm{E}\left[\mathrm{E}\left[T^{\mathbf{I},\mathbf{F}}_{m_1,m_2}(t_b;\mathbf{C})\middle|\tau\right]\right] = \mathrm{E}\Bigg[\sum_{j=1}^{2}\mathrm{E}\left[T^{\mathbf{I},\mathbf{F}}_{m_1,m_2}(t_b;\mathbf{C})\middle|\tau=W_j\right]P(\tau=W_j)+\sum_{j=1}^{2}\mathrm{E}\left[T^{\mathbf{I},\mathbf{F}}_{m_1,m_2}(t_b;\mathbf{C})\middle|\tau=X_j\right] \times P(\tau=X_j)+\sum_{j=1}^{2}\mathrm{E}\left[T^{\mathbf{I},\mathbf{F}}_{m_1,m_2}(t_b;\mathbf{C})\middle|\tau=Y_j\right]P(\tau=Y_j)+\sum_{j=1}^{2}\sum_{k=1,j\neq k}^{2}\mathrm{E}\left[T^{\mathbf{I},\mathbf{F}}_{m_1,m_2}(t_b;\mathbf{C})\middle|\tau=Z_{jk}\right]P(\tau=Z_{jk})\Bigg]. \quad (2)$$

If we consider the regeneration event {τ=W_1} (the service of a task at the first CE), one can show that $\mathrm{E}\left[T^{\mathbf{I},\mathbf{F}}_{m_1,m_2}(t_b;\mathbf{C})\middle|\tau=W_1\right] = \tau + \mathrm{E}\left[T^{\mathbf{I},\mathbf{F}}_{m_1-1,m_2}(t_b-\tau;\mathbf{C})\right]$, which means conditional on the service of a task at the first CE the original problem starts afresh with a new initial task distribution (m_1-1 *tasks at the first CE and* m_2 tasks at the second CE) and a new balancing instant (at t_b-τ). Similar "recurrence" relationships can be proven for every possible regeneration event. After considering all possible regeneration events, we can obtain a set of coupled difference-differential equations that completely describe the dynamics of the average response time. Going back to our simplified example, a sample equation from the set is

$$\frac{d}{dt_b}\mathrm{E}\left[T^{\mathbf{I},\mathbf{F}}_{m_1,m_2}(t_b;\mathbf{C})\right] = \sum_{j=1}^{2}\lambda_{d,j}\mathrm{E}\left[T^{\mathbf{I},\mathbf{F}}_{m_1-\delta_{j,1},m_2-\delta_{j,2}}(t_b;\mathbf{C})\right]+\sum_{j=1}^{2}\lambda_{f,j}\mathrm{E}\left[T^{\mathbf{I},\mathbf{F}_j}_{m_{1\mathrm{l}},m_2}(t_b;\mathbf{C})\right] +\sum_{j=1}^{2}\sum_{k=1,k\neq j}^{2}\lambda_{jk}\mathrm{E}\left[T^{\mathbf{I},\mathbf{F}}_{m_1+L_{21}\delta_{j,1},m_2+L_{21}\delta_{j,2}}(t_b;\mathbf{C})\right], \quad (3)$$

where $\lambda_{d,j}$, $\lambda_{f,j}$ and λ_{jk} are the processing rate of the *j*th CE, the failure rate of the *j*th CE and the task transfer rate from the *j*th to the *k*th CE, respectively. The term L_{jk} is the number of tasks transferred from the *j*th to the *k*th CE and $\delta_{j,k}$ is the Kroenecker delta. Given that L_{jk} depends on K_{jk}, we can formulate an optimization problem where t_b and the LB gains are judiciously selected so that

the response time is minimized. Finally, structurally similar equations can also be obtained for the probability of successfully serving an application, as it is shown in [2],[3],[4].

Small-scale implementation of a DCS

We have implemented a small-scale testbed DCS to experimentally test our DLB policies. The hardware architecture consists of the CEs and the communication network. Upon the occurrence of a failure, a CE is switched from the so-called working state to the backup state. If a node is in the backup state then it is not allowed to continue processing tasks, but is allowed to work as a backup system that only receives and transmits tasks, so that no task in the system is missing. The occurrence of failures and recoveries at any CE is simulated by software. The communication network employed in our architecture is the Internet, where the final links connecting the CEs are either wired or wireless. Our testbed also allows us to introduce, if needed, artificial latency by employing traffic shaper applications. The software architecture of our DCS is divided in three layers: application, LB and communication. Layers are implemented in software using POSIX threads. The application layer executes the application selected to illustrate the distributed processing of matrix multiplication. To achieve variability in the processing speed of the CEs, the randomness is introduced in the size of each row of the matrix by independently choosing its arithmetic precision with an exponential distribution. In addition, the application layer determines the failure and recovery instants at each CE by switching its state from working and to backup. The LB layer executes the LB policy defined for each type of experiment conducted. This layer monitors the queue-length of the CEs and triggers the LB action. It also: (i) determines if a CE is overloaded with respect to the other nodes in the system; (ii) computes the amount of task to transmit to other CEs; and (iii) selects which CEs are candidates to receive the excess amount of tasks. Finally, the communication layer of each node handles the transfer of tasks as well as the transfer of queue-length information among the CEs. Each node uses the UDP transport protocol to transfer a queue-length packet to the other CEs, and the TCP transport protocol to transfer the tasks between the CEs. Also, when a CE is in the backup state, this layer receives and transmits tasks, if necessary.

DLB policies for systems where CEs fail and recover

The preemptive DLB policy

The so-called preemptive DLB policy allows a single load transfer between the CEs and no other balancing action is taken afterwards. The DLB action is preemptive in the sense that it will counteract for the combined effects of failures, recoveries and communication times on the application response time. At time zero, CEs are assumed to be functioning and a CE, say the *j*th CE transfers $L_{jk}(t_b)$ tasks to the *k*th CE at time t_b, where $L_{jk}(t_b) = \lfloor K_{jk} p_{jk} L_j^{ex}(t_b) \rfloor$ and $L_j^{ex}(t_b)$ is computed using (1) with the Λ_j's parameters defined to be equal to the

processing rate of the CEs. The LB gains are optimally selected by solving recurrence equations (4) in [3].

Figure 1 (extracted from Fig. 3 in [3]) depicts the average response time of the matrix multiplication application as a function of the LB gain. Notably, the theoretical, the MC-simulated, and experimental results show a fairly good agreement. In addition, for comparison purposes, the results for the no failure case are also shown. From the theoretical curves we can observe that a proper task allocation can effectively reduce the average response time of an application. Note that the optimal number of tasks to reallocate is smaller in the scenario when CEs randomly fail and recover, as compared to the no failure case. Intuitively, we can state that the optimal task reallocation in case of CE failure will always be less than the optimal task reallocation for the no failure case.

The responsive DLB policy

In this policy, each CE initially executes a DLB action at time t_b assuming that the CEs are totally reliable. In other words, the policy does not care about future CE failures and subsequent recoveries during the execution of the application. Therefore, the initial LB action is taken to achieve an "approximately" uniform division of the total workload of the DCS among all the nodes assuming that all the CEs will remain functional. Once again, equation (1) is used to detect an imbalance and the Λ_j's parameters are defined as the processing rate of the CEs. The first difference with the preemptive DLB policy is that the LB gains are optimally selected solving a set of equations simpler that the one used by the preemptive DLB policy, these equations are stated in equation (9) in [2]. The second difference is that upon the occurrence of failure at any CE, the backup system of the failed CE executes an extra LB action in order to compensate for the time that will be wasted during the recovery process of the CE. Upon failure, the backup system of the failed CE reallocates tasks according to the following rule:

$$L_{jk}(t_f) = \left\lfloor \left(\frac{\lambda_{d,j}}{\lambda_{r,j}} \right) \left(\frac{\lambda_{d,k}}{\sum_{l=1}^{n} \lambda_{d,l}} \right) \left(\frac{\lambda_{r,j}}{\lambda_{r,j} + \lambda_{f,k}} \right) \right\rfloor, \qquad (4)$$

where the $\lambda_{r,j}$ is the recovery rate of the *j*th CE. Note that the first term at the right hand side of (4) is the average number of tasks that have not been served during the recovery time of the *j*th CE, the second term is the fair share of tasks of the receiving CE, and the third term is the steady-state probability of any CE to be functioning during the recovery time of the *j*th CE. Note that this upon failure task reallocation is fixed and determined by the parameters of the CEs.

Table 1 (extracted from Table II in [3]) lists the average response time achieved by the responsive DLB policy for some representative initial workload allocations. For comparison we also list also the average response time achieved by the preemptive DLB policy as well as the no-failure case. We can see that the

responsive DLB policy outperforms the preemptive policy in all cases. In general, this behavior is observed for communication links where the transfer times per task are small (about 1 second per task). However, the preemptive DLB policy outperforms the responsive policy in scenarios where the communications times per tasks are larger than 1 second (more details shown in Table III in [3]). Finally, in order to compare the dynamics of the DCS under each policy, we show in Figure 2 (extracted from Fig.4 in [3]) the actual queues at each CE during one realization of the experiments performed for the preemptive and responsive DLB policies. We can observe that the long flat portions of the queues correspond to the recovery times of the CEs. The downward (upward) jumps in the queues correspond to the action of transferring (receiving) tasks after every failure instant.

DLB policies for systems where CEs fail permanently

In harsh environments where nodes fail catastrophically, or during a time much larger than the average application response time, an appropriate metric of reliability is the probability of successfully serving the application. Theorems 1 and 2 in [5] present the set of recurrence equations governing the dynamics of a DCS in the presence of this type of long term failures. The DLB policies developed for this scenario employ (1) to determine the imbalances in the system. Unlike in the failure and recovery case, the Λ_j's parameters are defined also as the average failure time of the CEs, or alternatively, they can be defined as $\Lambda_j = \lambda_{d,j}\left(1-\lambda_{f,j}/\sum_{l=1}^{n}\lambda_{f,l}\right)$. For this last definition, the Λ_j's parameters can be thought of as an *effective processing rate* of a CE, which penalizes the processing rate of a CE by its relative unreliability.

Figure 3 (extracted from Fig. 2b in [5]) shows the probability of successfully serving an application, which we have denoted as $R^{\mathbf{I},\mathbf{F}}_{m_1,m_2}(t_b;\mathbf{C})$, as a function of the LB gains. We can observe again a remarkable agreement between theoretical, MC and experimental results. We can see that the probability of successful completion seems to be convex as a function of the number of tasks to be reallocated. From our experience on the LB problem, we can say that the probability of successful completion (correspondingly, the average response time) has such concave (correspondingly, convex) shape when the communication network of the DCS exhibits notorious random transfer times.

Table 2 (extracted from Table 3 in [5]) lists the probability of success for different DLB policies. The policies are differ in the balancing criterion used, i.e., by the definition of the Λ_j's parameters. The Maximal-Service policy defines the Λ_j's in terms of the average failure time of the CEs, the Processing-Speed Policy defines the parameters in terms of the processing rate of the CEs, and the Complete policy define the parameters in terms of the *effective processing rate* of the CEs. In addition, we have considered an example where the fastest CEs are at the same time the less reliable ones. From the Table 2 it can be concluded that a similar performance level can be achieved independently of the balancing

criterion employed, thereby showing the strength of our approach. Finally, as it is shown in Figure 4 (extracted from Fig. 2c in [5]), caution must be exercised in the selection of the balancing instant and the LB gains, otherwise the probability of successfully serving an application can be notoriously degraded, as in the LB policy where K_{12}=0.1 and K_{21}=0.6.

Research directions

More fundamental research on modeling complex computing infrastructures is needed, in order to predict and understand the response of such infrastructures in the presence of network and/or CE failures. Models must include the possibility of failures at different layers. For instance, models must consider long-term physical attacks such as those inflicted by WMDs, communication-layer attacks like network flooding, and application-layer attacks like those produced by malicious software. New models will significantly enhance the capabilities of homeland security to: (i) optimize computing networks' performance and robustness in the presence of failures; (ii) warn users and operators about a system dysfunction and provide quantitative description of its magnitude; (ii) design modern computing networks that have superior resilience and robustness in the presence of failures.

There is also the need for developing models to characterize, from the point of view of a computing network, the environment and the extent of damage produced by severe attacks on a DCS. The random nature of these attacks demands for a statistical framework, where spatial and temporal correlations of the attacks must be considered. In addition, early detection mechanisms of infrastructure network attacks are also required. These detection mechanisms must be informed also about the spatio-temporal correlations associated to the attacks, so that we can develop smart detection systems capable of warning the users potentially affected by a certain attack.

Acknowledgement

This work was supported by the Defense Threat Reduction Agency (Combating WMD Basic Research Program) and in part by the National Science Foundation (award ANI-0312611).

References

[1] Shirazi, BA, Kavi, KM, and Hurson, AR. Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE Computer Society Press 1995.

[2] Dhakal, S, Hayat, MM, Pezoa, JE, Yang, C, and Bader, DA. Dynamic load balancing in distributed systems in the presence of delays: A regeneration-theory approach. IEEE Trans. Parallel and Dist. Systems 2007 Vol. 18: 485–497.

[3] Dhakal, S, Hayat, MM, Pezoa, JE, Abdallah, CT, Birdwell, JD, and Chiasson, J. Load balancing in the presence of random node failure and recovery. Proc. IEEE IPDPS 2006.

[4] Dhakal, S, Pezoa, JE, and Hayat, MM. A regeneration-based approach for resource allocation in cooperative distributed systems. Proc. ICASSP 2007, III-1261–III-1264.

[5] Dhakal, S, Pezoa, JE, and Hayat, MM. Maximizing Service Reliability in Distributed Computing Systems with Random Failures: Theory and Implementation. Submitted to IEEE Trans. Parallel and Dist. Systems 2008. Paper available at http://www.ece.unm.edu/lb.

Further reading list

Dhakal, S, Hayat, MM, Elyas, M, Ghanem, J, and Abdallah, CT. Load balancing in distributed computing over wireless LAN: Effects of network delay. Proc. IEEE WCNC 2005.

Hayat, MM, Dhakal, S, Abdallah, CT, Birdwell, JD, and Chiasson, J. Dynamic time delay models for load balancing. Part II: Stochastic analysis of the effect of delay uncertainty. Adv. in Time Delay Systems, LNCS 2004 Vol. 38: 355–368.

Dhakal, S, Paskaleva, BS, Hayat, MM, Schamiloglu, E, and Abdallah, CT. Dynamical discrete-time load balancing in distributed systems in the presence of time delays. Proc. IEEE CDC 2003, 5128–5134.

The website of the Load Balancing Group at The University of New Mexico: http://www.ece.unm.edu/lb.

Sonnek, J, Chandra, A, and Weissman, J. Adaptive Reputation-Based Scheduling on Unreliable Distributed Infrastructures. IEEE Transactions on Parallel and Distributed Systems 2007 Vol. 18: 1551–1564.

Daley, DJ and Vere-Jones, D. An introduction to the theory of point processes. Springer-Verlag 1988.

Cross-references

Glossary terms

Suggested reviewers

Figure legends

Figure 1

The average response time of the matrix multiplication application as a function of the LB gain *K* for the preemptive LB policy. Extracted from Fig. 3 in [3].

Figure 2

A DCS where CEs can randomly fail and recover: a sample realization of the queues dynamics. Extracted from Fig. 4 in [3].

Figure 3

The probability of successfully serving an application as a function of the LB gains. The example considers a DCS where CEs can catastrophically fail at any random time. Extracted from Fig. 2b in [5].

Figure 4

The probability of successfully serving an application as a function of the balancing instant. The example considers a DCS where CEs can catastrophically fail at any random time. Extracted from Fig. 2c in [5].

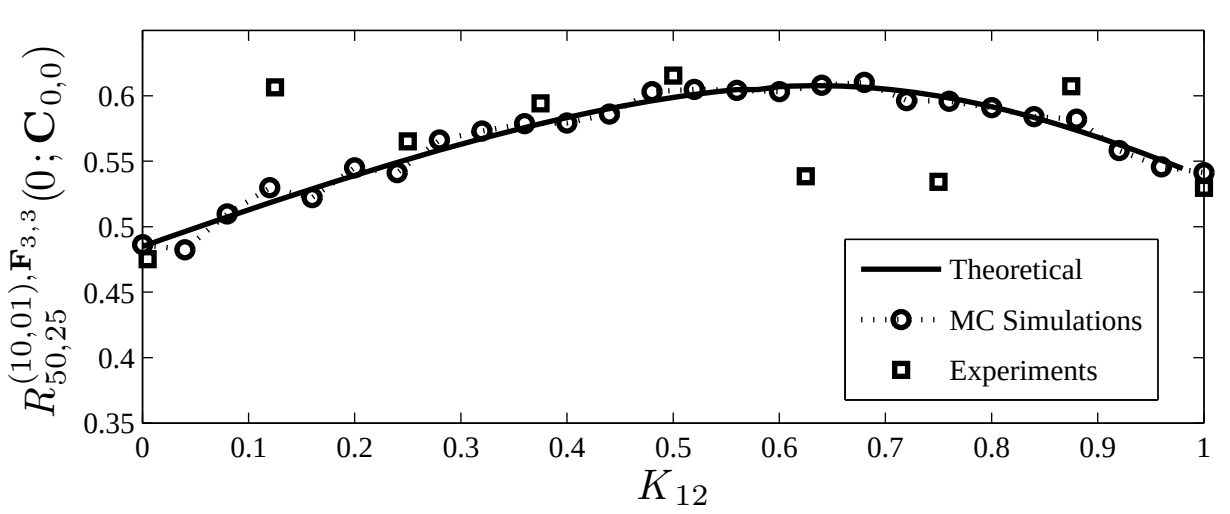
Table legends

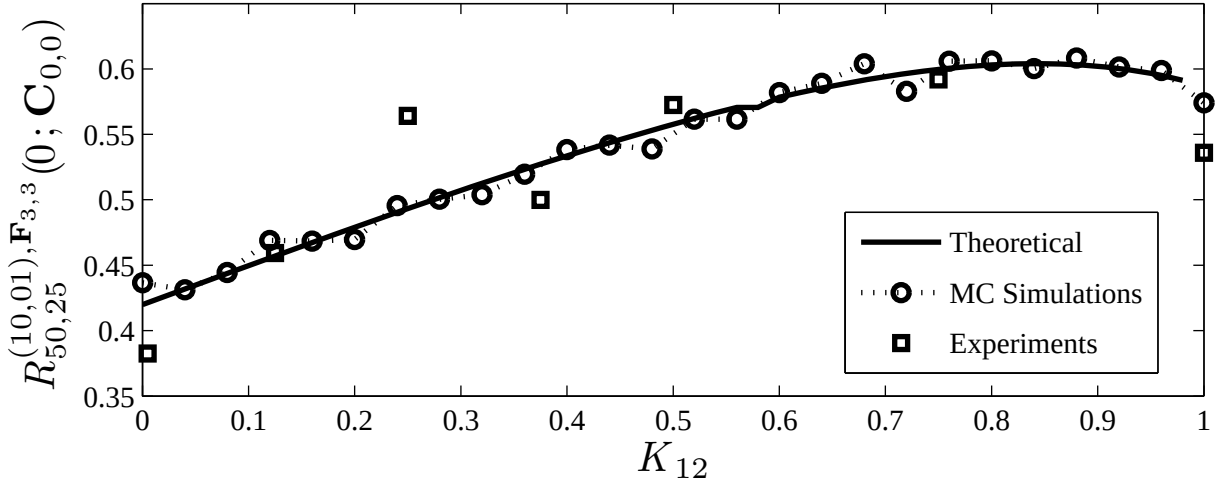
Table 1

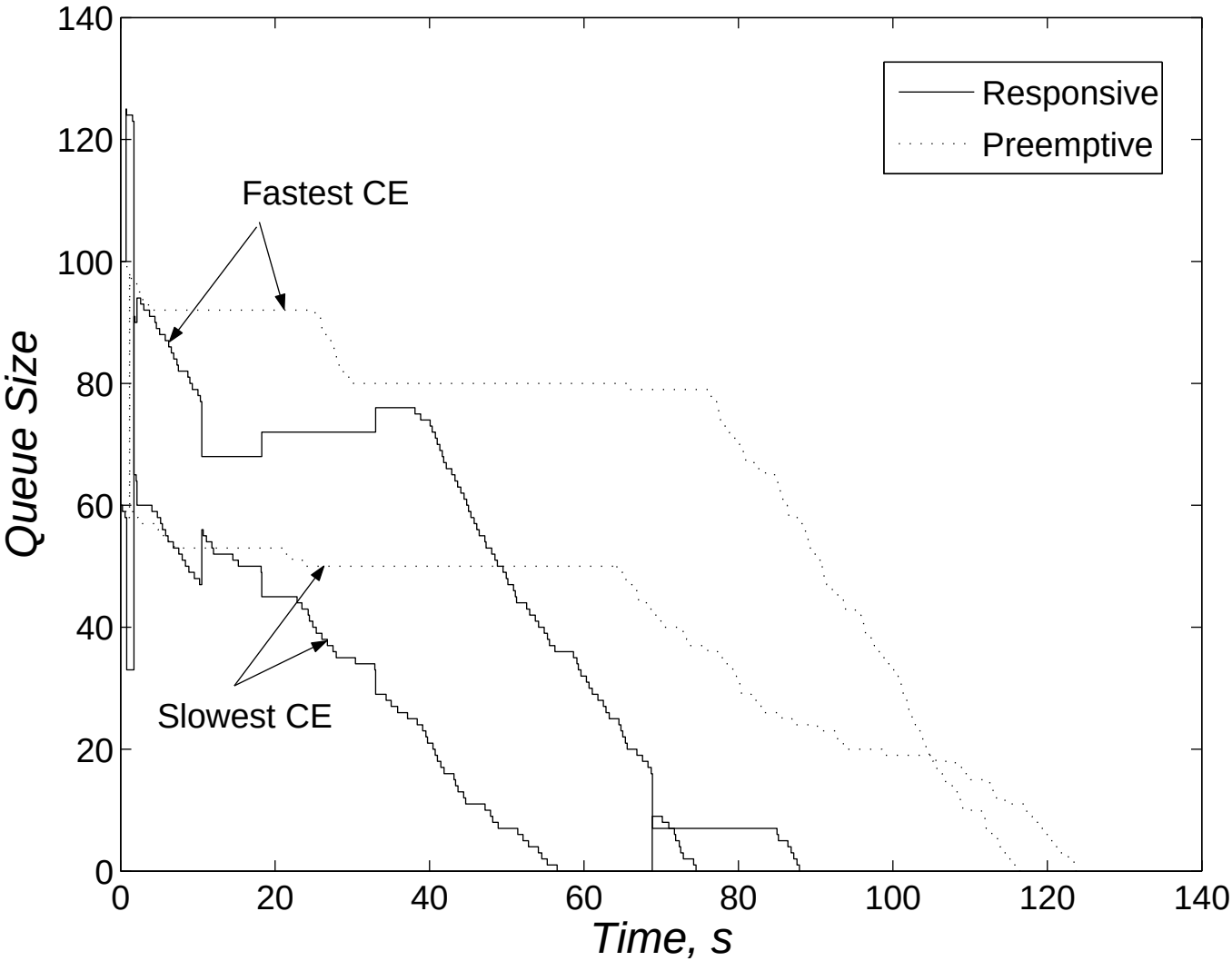
Experimental and simulation results for the responsive DLB policy applied in scenarios where nodes can fail and recover. For comparison, results for the no failure case and the preemptive DLB policy are also listed. Extracted from Table II in [3].

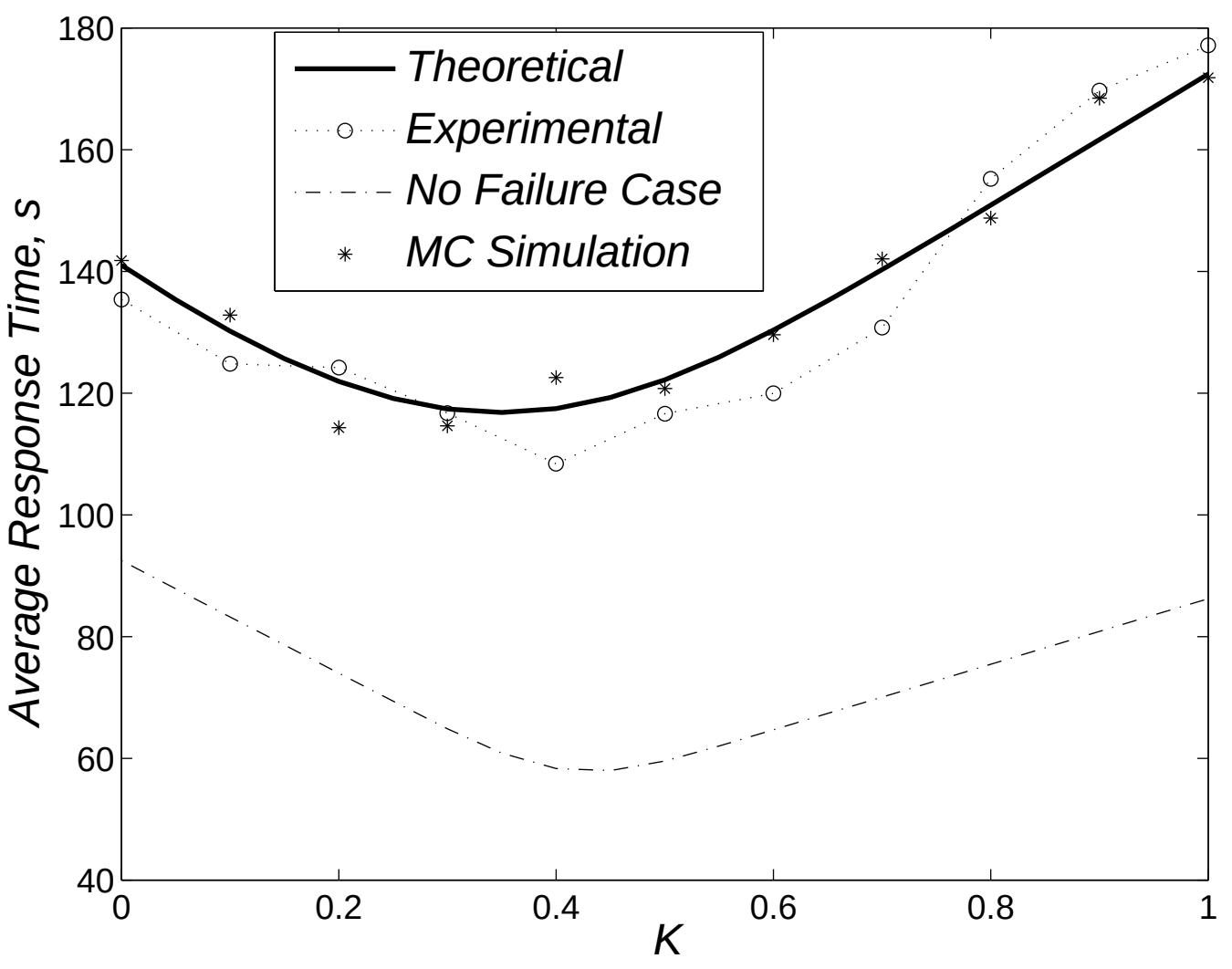
Table 2

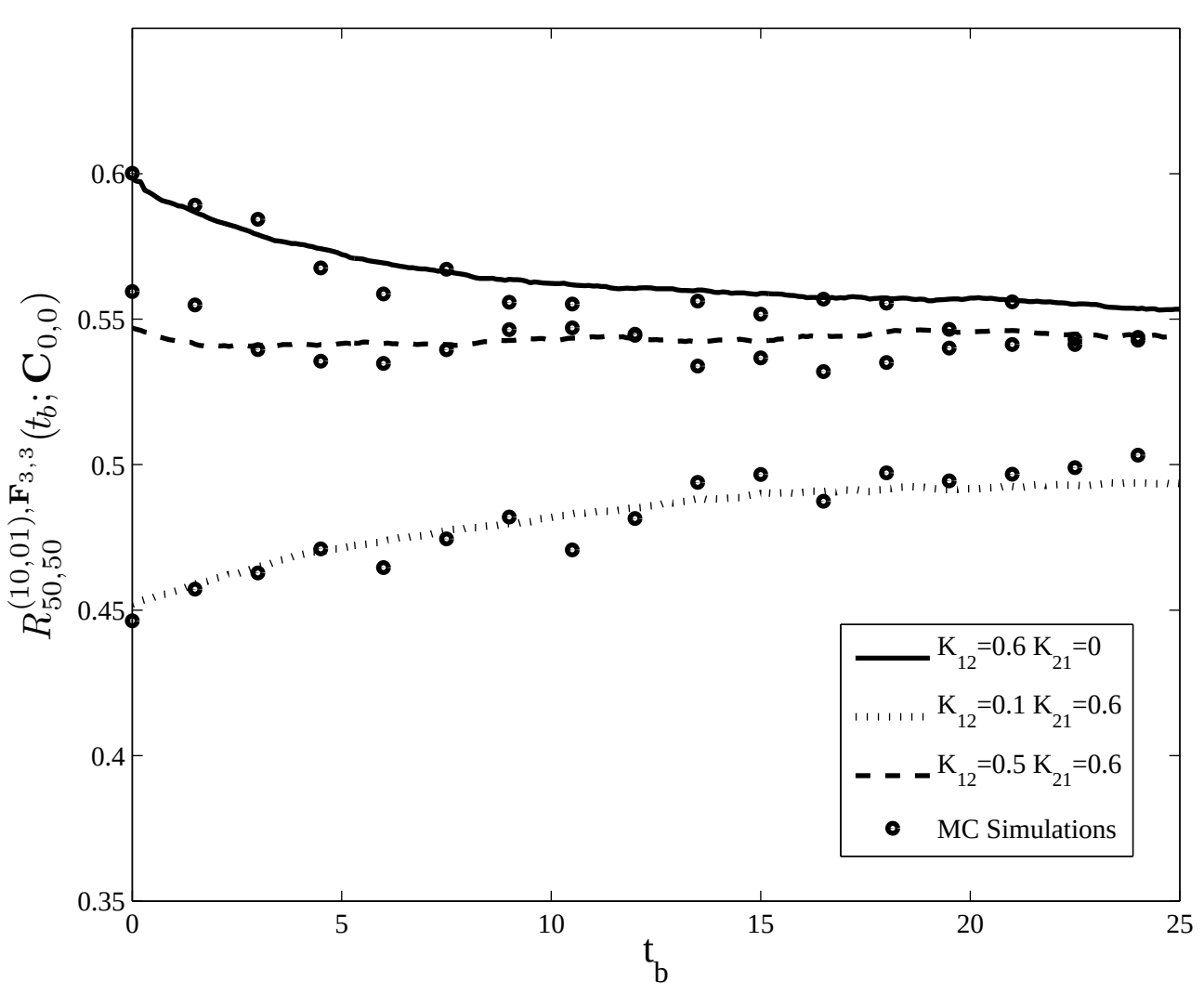
The probability of successfully serving an application for different initial workload distribution. Different balancing criteria have been employed to balance the DCS. Extracted from Table 3 in [5].


$R_{50,25}^{(10,01),\mathbf{F}_{3,3}}(0;\mathbf{C}_{0,0})$
0.6
0.55
0.5
0.45
0.4
0.35
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
K_{12}
Theoretical
MC Simulations
Experiments


$R_{50,25}^{(10,01),\mathbf{F}_{3,3}}(0;\mathbf{C}_{0,0})$
0.6
0.55
0.5
0.45
0.4
0.35
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
K_{12}
Theoretical
MC Simulations
Experiments


Responsive
Preemptive
Fastest CE
Slowest CE
Queue Size
Time, s
0
20
40
60
80
100
120
140


Theoretical
Experimental
No Failure Case
MC Simulation
Average Response Time, s
K
180
160
140
120
100
80
60
40
0
0.2
0.4
0.6
0.8
1


$R^{(10,01),\mathbf{F}_{3,3}}_{50,50}(t_b;\mathbf{C}_{0,0})$
0.6
0.55
0.5
0.45
0.4
0.35
0
5
10
15
20
25
t_b
K12=0.6 K21=0
K12=0.1 K21=0.6
K12=0.5 K21=0.6
MC Simulations

Initial Workload	Responsive DLB Policy		Preemptive DLB Policy (Theoretical)	No Failure
	MC Simulations	Experimental Results		
(200,200)	277.9	263.4	275.0	141.9
(200,100)	202.4	188.8	210.1	106.9
(100,200)	203.1	212.9	210.1	106.9
(200,50)	170.8	171.4	177.1	89.3
(50,200)	170.8	177.6	177.1	89.3

Initial load (m_1,m_2,m_3,m_4,m_5)	Maximal-Service	Processing-Speed	Complete	Optimum
(150,0,0,0,0)	0.2338	0.1845	0.2223	0.2632
(0,150,0,0,0)	0.2853	0.2908	0.2653	0.2908
(0,0,150,0,0)	0.2868	0.2678	0.2760	0.2867
(0,0,0,150,0)	0.2915	0.2965	0.2978	0.2978
(0,0,0,0,150)	0.2545	0.2940	0.3125	0.3125
(30,30,30,30,30)	0.2953	0.3105	0.3045	0.3170
(59,2,4,34,51)	0.2579	0.2583	0.2868	0.3183
(18,55,29,27,21)	0.2943	0.3098	0.3053	0.3148
(26,30,28,38,28)	0.3185	0.2978	0.3040	0.3185
(40,15,40,35,20)	0.2860	0.2873	0.2845	0.2963