J. Parallel Distrib. Comput. 72 (2012) 1186–1194

Contents lists available at SciVerse ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Probabilistic resource allocation in heterogeneous distributed systems with random failures☆

Vladimir Shestak [a,*], Edwin K.P. Chong [b,d], Anthony A. Maciejewski [b], Howard Jay Siegel [b,c]

[a] Ricoh InfoPrint Solutions, 6300 Diagonal Highway, Boulder, CO 80301, United States

[b] Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523–1373, United States

[c] Department of Computer Science, Colorado State University, Fort Collins, CO 80523–1373, United States

[d] Department of Mathematics, Colorado State University, Fort Collins, CO 80523–1373, United States

ARTICLE INFO

Article history:
Received 21 April 2011
Received in revised form
16 December 2011
Accepted 9 March 2012
Available online 11 April 2012

Keywords:
Resource allocation
Distributed systems
Fault tolerant systems
Load balancing

ABSTRACT

The problem of finding efficient workload distribution techniques is becoming increasingly important today for heterogeneous distributed systems where the availability of compute nodes may change spontaneously over time. Resource-allocation policies designed for such systems should maximize the performance and, at the same time, be robust against failure and recovery of compute nodes. Such a policy, based on the concepts of the Derman–Lieberman–Ross theorem, is proposed in this work, and is applied to a simulated model of a dedicated system composed of a set of heterogeneous image processing servers. Assuming that each image results in a "reward" if its processing is completed before a certain deadline, the goal for the resource allocation policy is to maximize the expected cumulative reward. An extensive analysis was done to study the performance of the proposed policy and compare it with the performance of some existing policies adapted to this environment. Our experiments conducted for various types of task-machine heterogeneity illustrate the potential of our method for solving resource allocation problems in a broad spectrum of distributed systems that experience high failure rates.

1. Introduction

Distributed computing systems are widely used today for the execution of large workloads composed of independent tasks that are divided among multiple heterogeneous compute nodes. The assignment of tasks to compute nodes is referred to in the literature as a *resource allocation* or a *mapping*. Effective mapping policies must account for multiple factors, e.g., the set of available compute nodes in the system, characteristics of these nodes, and links between them. Multiple scenarios can be identified for the first factor where there is an uncertainty in the availability of functional compute nodes over time. Due to this uncertainty, any compute node may randomly fluctuate between the "failure" ("down") and "working" ("up") states.

As a first example, consider a distributed system with a dynamic set of compute nodes, i.e., nodes may join and leave the system in an ad-hoc fashion. One use of such a system is SETI@Home, composed of remote non-dedicated workstations that participate in distributed data processing [31]. Typically, compute 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 local usage of the node by its owner.

A phenomenon known as "software aging" is a second example [14,21]. Software aging sources include memory leaks, unreleased file locks, accumulation of unterminated threads, data corruption/round-off accrual, filespace fragmentation, and shared memory pool latching. Performance problems caused by software aging have become commonplace for computing resources including safety critical systems. For example, software aging of the Patriot Missile software was responsible for the loss of lives of American soldiers during the first Gulf War [22]. The solution found for this problem was to restart the Patriot software components every eight hours. Recovery mechanisms for software aging involve different, often proactive fault management techniques for cleaning up system internal states to prevent the future occurrence of more severe failures or system performance degradation.

Another common source of the changing availability of compute nodes in the system is the malfunction of underlying hardware resources, due to harsh operating conditions or external physical impacts on the system. For example, broken cooling fans in a machine room typically cause a temperature increase

☆ This research was supported in part by the InfoPrint Solutions Company, the National Science Foundation under Grants No: CNS-0615170, ECCS-0700559, CNS-0905399, and the Colorado State University George T. Abell Endowment.

* Corresponding author.
E-mail addresses: vladimir.shestak@infoprint.com (V. Shestak), echong@colostate.edu (E.K.P. Chong), aam@colostate.edu (A.A. Maciejewski), hj@colostate.edu (H.J. Siegel).

0743-7315/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2012.03.003

that results in a high occurrence of processor malfunctions. Consequently, this initiates fatal errors or dramatic performance degradation that OS or process monitoring agents often resolve by restarting a process or rebooting an entire system.

Fault tolerance in distributed computing systems has been extensively explored in the last few years [7]. Available literature on distributed computing in such uncertain environments primarily considers reactive techniques, where a node failure is addressed only after its occurrence. Checkpoint-resume or terminate-restart mechanisms are often used to recover unprocessed tasks at the failed nodes [20,8]. Tolerance against node failure can also be addressed by keeping multiple copies of the workload on different nodes [35]. These approaches are coupled in practice with redundancy schemes that duplicate system hardware resources entirely or partially. Depending on the implementation, duplicated resources are either always active or become active dynamically. Additionally, most of the existing literature that offers an analytical formulation of distributed-computing systems assumes a homogeneity among compute nodes and known system parameters [15,32].

Clearly, the uncertainty in the availability of compute nodes is expected to degrade the performance of any resource-allocation policy that does not account for node failure and recovery. In this study, we model a dedicated image processing system composed of a limited set of heterogeneous servers. Companies specialized on providing GIS services, e.g., ESRI, Digital Globe, GeoEye, widely use such systems for processing streams of raw image data. Furthermore, HPC systems with similar architectures are quite commonly employed to handle high volumes of computationally intensive scientific applications.

The availability of servers is described with exponential distributions with failure rates that are relatively high to simulate harsh operating conditions. Each image is assumed to have a hard deadline, limiting the total time available to process it, and an associated reward. The goal is to design a resource-allocation policy that maximizes the expected cumulative reward received from the processed images that finish before their deadlines, with the uncertainty of compute node failures.

In analyzing systems where compute resources fail in a random fashion, the operational intervals of the compute resources are uncertain. This uncertainty can impact the system by causing tasks to fail during execution and finally miss their deadlines. With current computer systems approaching exascale levels, the concern over robust resource allocation gains much more importance. Often in practice, when the number of failed tasks reaches a certain level a service provider becomes subjected to profit losses and penalties. According to [4], any claim of robustness for a given system must answer three questions: (a) what behavior of the system makes it robust? (b) what uncertainties is the system robust against? (c) quantitatively, exactly how robust is the system? In this research, we say the system is robust if its expected profit is greater than the cost to operate the system. The uncertainty, that the system must be robust against, is which resources fail and when. The degree of robustness is measured as the difference between the actual (or expected) profit from the completed tasks and the lowest level that justifies the operation of the system.

The major contribution of this paper is the design of a mathematical model and resource-allocation policy for the aforementioned distributed image processing environment based on the concepts of the Derman–Lieberman–Ross theorem [10]. Our simulation experiments explore the pros and cons of the proposed solution by comparing it with other policies considered in this study. They demonstrate that the good performance of this solution is sustainable in environments with different types of heterogeneity among tasks and compute nodes confirming the potential of this resource-allocation mechanism in maximizing the performance of a distributed heterogeneous system experiencing temporal compute node failures.

The remainder of this work is organized in the following manner. Section 2 describes the general mathematical model used in this study. It starts with a simplified resource allocation problem in Section 2.1, then considers tasks with exponentially distributed execution times and different types of heterogeneity in Section 2.2, a distribution of processor availability in Section 2.3, and, finally, tasks with deadlines in Section 2.4. Section 3 presents the Derman–Lieberman–Ross theorem and a corollary. The parameters of the simulation setup are discussed in Section 4 along with the resource-allocation policies compared in this study. Section 5 discusses the simulation results. A review of related work is presented in Section 6. Section 7 concludes the paper.

2. Mathematical model

2.1. Simplified resource allocation

Consider the following problem statement as a simplified resource allocation problem in a distributed system. Suppose there is a batch of N tasks. Each task i, $1 \leq i \leq N$, is characterized by: (a) the number of instructions it contains, denoted n_i, and (b) reward r_i for completing this task. Suppose that N processors become available sequentially in time. Whenever a new processor becomes available, we need to assign one of the remaining (unassigned) tasks to this processor. Suppose that the jth processor to arrive is characterized by a random variable T_j that specifies the time that processor takes to execute each instruction (assuming that any instruction in any task takes the same amount of time to execute on the given processor). Moreover, the processor might randomly fail during the execution of a task. Its failure can be modeled with an exponential distribution defined by its failure rate λ_j. Specifically, the probability of failure during the execution of a task is given by λ_j multiplied by the task's execution time (we discuss this failure model further below).

Once task i is assigned to a processor, its execution begins, and reward r_i will be obtained if the task is completed successfully. If a processor fails while executing the task, no reward is earned and the task is removed from the batch, i.e., the task will never be executed again. Let $\mathbb{P}[i, m(i)]$ denote the probability that task i is successfully completed when assigned to processor $m(i)$. Then, the goal is to maximize the total expected reward accumulated through N sequential assignments:

$$\text{maximize } E\left[\sum_{i=1}^{N} r_i \mathbb{P}[i, m(i)]\right]. \tag{1}$$

Using the failure model above, $\mathbb{P}[i, m(i)] = 1 - \lambda_{m(i)} n_i T_{m(i)}$. Substituting this into Eq. (1) and simplifying, the optimization problem becomes

$$\text{maximize } E\left[\sum_{i=1}^{N} \left([r_i n_i] \times \left[-\lambda_{m(i)} T_{m(i)}\right]\right)\right]. \tag{2}$$

The failure model above can be viewed as an approximation to the standard model where the time it takes for failure is exponentially distributed with parameter λ_j. In this case, the probability that processor $m(i)$ does not fail while executing task i is given by $e^{-\lambda_{m(i)} n_i T_{m(i)}}$. Using the Taylor series expansion for the exponential function,

$$e^{-\lambda_{m(i)} n_i T_{m(i)}} = 1 - \lambda_{m(i)} n_i T_{m(i)} + \frac{(-\lambda_{m(i)} n_i T_{m(i)})^2}{2!} + \frac{(-\lambda_{m(i)} n_i T_{m(i)})^3}{3!} + \cdots.$$

Ignoring second order and higher terms, we arrive at the failure model used in the derivation above.

2.2. Tasks with exponentially distributed execution times

In the simplified resource-allocation problem described above, the time to compute task i on processor j, denoted t_{ij}, is the product of n_i and T_j. However, in practice, it is practically difficult or even impossible to represent t_{ij} with such a product. This is mainly because the exact number of executed instructions is usually unknown in advance due to conditional branching, uncertain input data, etc. Furthermore, instruction execution rates of participating processors may depend on the mix of the types of executed instructions [16]. Therefore, an *E*stimated *T*ime to *C*ompute (*ETC*) matrix based on statistical data typically provides a more accurate means of capturing the relationship among tasks and processors in distributed systems. Each entry in an ETC matrix is an *estimate* of time t_{ij}. In the literature on distributed systems, ETC matrices are often assumed to be given [19,17,36].

The rest of this paper will focus on tasks with execution times exponentially distributed for each task-processor pair. This assumption implies that an ETC matrix contains expected execution times that are assumed to be known (expected values, i.e., means, are sufficient to fully describe exponential distributions [11]). Eq. (2) can be restructured as:

$$\text{maximize } E\left[\sum_{i=1}^{N}\left(r_i \times \left[-\lambda_{m(i)} t_{im(i)}\right]\right)\right]. \tag{3}$$

The relationships among elements in an ETC matrix for a given distributed system depends on what class of heterogeneity this system belongs to. In this study, we distinguish between four major classes of heterogeneity in distributed systems: *task-processor-consistent*, *task-consistent*, *processor-consistent*, and *inconsistent*. The first class of heterogeneity describes systems where two conditions are satisfied: (1) if task A requires more time to execute than task B on one processor then the same is true for any other processor in the system; (2) if processor A requires more time to execute one task than processor B then it is true for any other task in the batch. The second type of heterogeneity implies condition (1) only. Similarly, the third type of heterogeneity implies condition (2) only. The fourth class describes systems where neither of these two conditions are met.

2.3. Distribution of processors

In Section 2.1, our mathematical model was restricted such that if a processor fails while executing the task, no reward is earned and the task is removed from the batch. This restriction is removed in the next subsection where multiple reassignments of a task are allowed as long as the waiting and processing times for a task do not exceed the task's deadline. As a preliminary step required to adjust the model to cover such cases, this subsection derives a statistical distribution of processor availability in a dedicated distributed system.

Distributions in dedicated distributed systems can be derived based on characteristics of the tasks batched and the parameters of the limited set of available processors. Fig. 1 illustrates a probability mass function (pmf) for a dedicated system composed of M processors. The values $-\lambda_j t_j^{\text{av}}$, $1 \le j \le M$, where t_j^{av} is an average execution time for processor j across execution times of all tasks stacked in the batch, arranged in ascending order are used to characterize processors.

Let p_j, $1 \le j \le M$, denote the probability (to become available for assignment) associated with each processor in the pmf shown in Fig. 1. To evaluate the p_j values, consider the following models describing the availability of processors in a dedicated distributed system. The first two models are of little practical value; they are presented here solely to provide a transition to the third model actually used in this study.

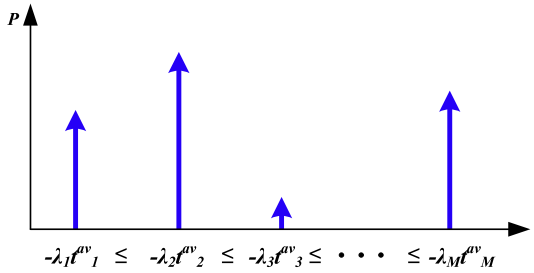


Fig. 1. The general pmf for a dedicated system with M processors.

Identical failure rates, processor becomes available just after failure: Assume that each processor has the same failure rate λ and becomes available for the next assignment immediately after a failure. Assume also that task completions do not make a processor available. Clearly in this situation, each processor fails, on average, the same number of times, i.e., the probabilistic components p_j are the same in the pmf shown in Fig. 1, given by $p_j = 1/M$.

Individual failure rates, processor becomes available just after failure: When each processor has its own failure rate λ_j, it fails over time, on average, every λ_j^{-1} time units. Therefore, the probability components p_j can be calculated as $p_j = \lambda_j / \sum_{k=1}^{M} \lambda_j$.

Individual failure rate, processor available just after failure and task completion: Suppose that each processor has failure rate λ_j and becomes available immediately after a failure or a task completion. Let w_j denote the total availability rate for processor j. Assuming that times between failures and task completions are exponentially distributed for each processor, w_j can be computed as:

$$w_j = \lambda_j + \frac{e^{-\lambda_j t_j^{\text{av}}}}{t_j^{\text{av}}}. \tag{4}$$

The second term in the sum in Eq. (4) estimates the processor's availability rate from task completions. On average, processor j would complete a task every t_j^{av} if there were no failures. The actual availability rate from task completions is lower than that because the probability of having no failures during t_j^{av} is accounted for in $e^{-\lambda_j t_j^{\text{av}}}$. As before, the probability components p_j can be calculated by normalizing the w_j values, i.e., $p_j = w_j / \sum_{j=1}^{M} w_j$.

Based on our simulation studies, the following iterative scheme to compute the pmf was found to be the most efficient. According to this scheme, the distribution, described with a weighted sum of two pmfs, is recalculated at each step of the mapping process before the assignment of the next task takes place. The first pmf is a normalized histogram constructed from the actual number of times that each processor has become available since the execution of the batch started. For the second pmf, the processor availability rates w_j are recomputed because the number of tasks left in the batch decreases with time. Let $N(t_0)$ denote the initial number of tasks in the batch and let $N(t)$ denote the number of tasks left in the batch at time t. Then, $1 - N(t)N(t_0)^{-1}$ and $N(t)N(t_0)^{-1}$ are the weighting factors for the first and second pmfs in the sum, respectively. As such, these weighting factors are adjusted at each iteration proportionally to the progress made.

Note that in this study, we use exponential distributions to model processor failures as well as task completions. It was shown in the literature that stochastic processes of hardware failures can be well approximated with exponential distributions [26,13]. Our choice of exponential distribution for task completions was justified by extensive in-house testing conducted by Ricoh InfoPrint for its IP500 production printer controller series [18].

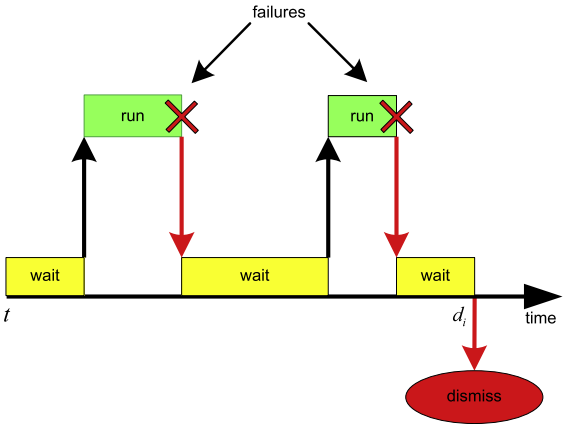


Fig. 2. Any task in the batch waits in the queue before it is assigned and returns back to the queue if a processor fails during its execution. If d_i expires the task is dismissed from the queue.

Working on optimization of the load balancing algorithm in that cluster-based image processing system, we found that exponential distributions provide a good fit into the spread of actual task completion times.

2.4. Tasks with deadlines

Suppose that in addition to the exponentially distributed execution time, each task has a deadline d_i, i.e., the reward r_i will be earned only if task i is successfully completed by time d_i. This means that if a processor fails before d_i while executing task i then task i returns to the batch, so it can be reassigned again. When d_i expires, the task is dismissed from the batch. Fig. 2 illustrates the possible states for a task and transitions between these states assuming that the considered task eventually fails.

To account for possible multiple reassignments of each task in the resource-allocation policy, let $\mathbb{P}_i(t)$ be an estimate of the probability at time t that task i will be successfully completed, i.e., its reward r_i is collected, before its deadline d_i expires. Then, Eq. (3) can be adapted as follows:

$$\text{maximize } E\left[\sum_{i=1}^{N(t)}[r_i\mathbb{P}_i(t)]\left[-\lambda_{m(i)}t_{im(i)}\right]\right]. \tag{5}$$

The following approach was used in this study to estimate $\mathbb{P}_i(t)$. Let $V_i(t)$ denote the expected number of reassignments for task i from time t until its deadline. As the probability of failure for the assigned task i is

$$\sum_{j=1}^{M} p_j(1-e^{-\lambda_j t_{ij}}), \tag{6}$$

the probability $\mathbb{P}_i(t)$ is

$$\mathbb{P}_i(t) = 1-\left(\sum_{j=1}^{M} p_j(1-e^{-\lambda_j t_{ij}})\right)^{V_i(t)}. \tag{7}$$

Let $T_i^{\text{wait}}(t)$ denote how long, on average, task i spends in a "wait" state (see Fig. 2). Similarly, let T^{run} denote the average time for a task to spend in a "run" state. Once $T_i^{\text{wait}}(t)$ and T^{run} are found, $V_i(t)$ can be computed as:

$$V_i(t) = \frac{d_i - t}{T_i^{\text{wait}}(t) + T^{\text{run}}}. \tag{8}$$

Assuming that any task from the batch can be assigned to any processor, the average expected delay from the time when a task is assigned to a processor to the time when the processor fails can be computed as the mean across λ_j^{-1} values:

$$T^{\text{run}} = \sum_{j=1}^{M} p_j\lambda_j^{-1}. \tag{9}$$

A delay expected between sequential assignments for a given task, i.e., $T_i^{\text{wait}}(t)$, depends on (1) how often processors become available and (2) how many tasks remain, on average, in the batch. To estimate (1), let W denote the overall processor availability rate in the system. As shown in Section 2.3, availability rate w_j for processor j can be estimated with Eq. (4). Similarly, in the system composed of M processors, W can be computed as a sum of w_j rates:

$$W = \sum_{j=1}^{M} w_j. \tag{10}$$

Although task completions and deadline expirations change the number of tasks that remain in the batch, in this study we use a lower bound for this number assuming that only the tasks with expired deadlines are dismissed over the time period $d_i - t$. This results in the following estimate for $T_i^{\text{wait}}(t)$:

$$T_i^{\text{wait}}(t) = \frac{N(t)+N(d_i)}{2\times 2W}. \tag{11}$$

In Eq. (11), $(N(t)+N(d_i))/2$ gives the average length of the queue over the time period $d_i - t$. This length is divided by 2 assuming that, on average, 50% of the remaining tasks will be assigned before task i. Our extensive experiments show that even if the values found for $\mathbb{P}_i(t)$ with the described method might not be quite accurate, they allow for a good relative ranking among the tasks—the only factor that matters in the resource-allocation policies described further.

3. DLR concept

This section describes the basic optimal policy for the sequential stochastic assignment problem introduced by Derman et al. [10]. Although this optimal policy cannot be applied directly to the resource allocation problem considered in this study, some of the policies in the next section are based on its principals.

Suppose there are N workers available to perform N jobs. The N jobs arrive in sequential order, i.e., job 1 arrives first, followed by job 2, etc. Associated with the jth ($j = 1, 2, \ldots, N$) job is a real-valued random variable X_j representing its worth. It will be assumed that the X_j are independent and identically distributed random variables with cumulative density function (cdf) $G_X(z)$ with finite mean value. If a "perfect" worker is assigned to the type jth job, a reward X_j is obtained. However, none of the N workers are perfect, and whenever the ith worker is assigned to the jth job, the reward is given by p_iX_j, where $0 \le p_i \le 1$, $i = 1, 2, \ldots, N$ represents the probability of worker i successfully completing any job. Each worker is assigned to one and only one job. The goal is then to assign the N workers to the N jobs so as to maximize the total expected reward. In particular, if $m(i)$ is defined to be the job to which ith worker is assigned, then the total expected reward is given by

$$E\left[\sum_{i=1}^{N} p_iX_{m(i)}\right]. \tag{12}$$

The desired policy is the one that maximizes this expected cumulative reward.

Intuitively, it appears reasonable to match the worth of a job to the probability of a worker of completing this job, e.g., assign an appeared high worth job to a worker with high p_i. Quantifying this match requires $G_X(z)$ and values of p_i. The following Derman–Lieberman–Ross (DLR) theorem embodies this intuition, providing the *optimal* resource-allocation policy using the available stochastic information (see [10] for further details).

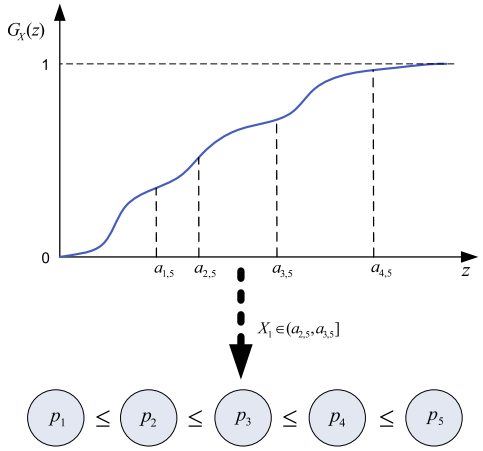


Fig. 3. The distribution described with cdf $G_X(z)$ is divided into five intervals. According to the DLR policy, as the worth X_1 of the arrived job belongs to the third interval, this job will be assigned to the third worker in the list of workers ranked based on probability values p_i.

DLR Theorem. *For each $N \geq 1$, there exist numbers $-\infty = a_{0,N} \leq a_{1,N} \leq a_{2,N} \leq \cdots \leq a_{N,N} = +\infty$, such that whenever there are N assignments to make and probabilities $p_1 \leq p_2 \leq \cdots \leq p_N$ then the optimal choice in the first assignment is to use the worker i such that X_1 is contained in the interval $(a_{i-1,N}, a_{i,N}]$. The $a_{i,N}$ depend on G_X but are independent of the p_i values and calculated recursively for N as follows:*

$$a_{i,N} = \int_{a_{i-1,N-1}}^{a_{i,N-1}} z\, dG_X(z) + a_{i-1,N-1} G_X(a_{i-1,N-1}) + a_{i,N-1}[1 - G_X(a_{i,N-1})],$$

with the convention that $a_{0,N} = -\infty$, $a_{N,N} = +\infty$, $-\infty \times 0 = 0$, and $\infty \times 0 = 0$.

Suppose that $N = 5$, and $G_X(z)$ is as illustrated in Fig. 3. When the DLR policy is applied, the $a_{i,5}$ values are calculated recursively starting from $N = 1$. These values divide the domain of all possible job worths into five intervals. Once the first job has arrived, a resource-management system identifies which of these five intervals this job falls in based on the job's worth X_1. Assume that this happened to be the third interval. Then, the resource-allocation system assigns the arrived job to the third worker in the sorted list of probability values $p_1 \leq p_2 \leq p_3 \leq p_4 \leq p_5$, as shown in Fig. 3. Sequentially as the next job arrives, N is decremented, and the same procedure is repeated by recalculating intervals.

4. Simulation setup and resource-allocation policies

The prototype system studied in this research is a dedicated heterogeneous cluster-based image processing system. In this study, we address a *static* environment assuming that a batch is filled with raw images (i.e., tasks) without considering any new arrivals during processing. New arrivals will form the next batch. We also do not consider concurrent execution of multiple tasks on the single image processing server, i.e., the multitasking mode. This is because of the fact that, typically, image processing algorithms are highly parallelized and their threads utilize almost all available cpu and memory resources. Thus, the tactics of spawning multiple kernel processes and switching between them is not considered here.

Fig. 4 depicts the high level architectural design on the prototyped image processing system. Note that image servers (i.e., processors) are not involved in the computations associated with resource-allocation process as it is all offloaded to the front-end resource manager. The resource manager receives updates reporting processor "health" and task completions through the network of software agents deployed on every processor. As each agent is a relatively light weight process, its concurrent execution does not seriously impact processor's resources and, therefore, is not taken into account in our model. Separate front-end resource managers are quite common in modern cluster architectures, e.g., the hardware-based F5 BIG-IP product family [6] and elastic resource managers offered by Amazon EC2 [5] and RackSpace [25].

To generate ETC matrices to be used in the simulations, we assume that each task in the batch belongs to one of five classes, based on its complexity. Each task class is defined by a set of exponential distributions, where each distribution describes the

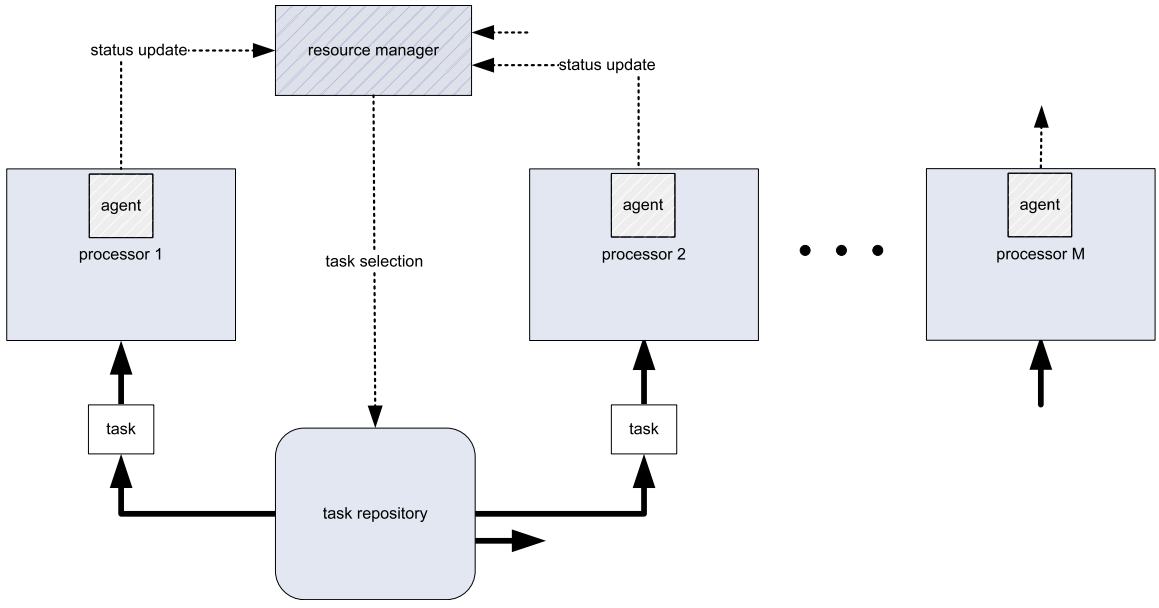


Fig. 4. The prototype of the cluster-based image processing system. The set of back-end processors is not involved in the computations associated with resource-allocation process as it is all offloaded to the front-end resource manager.

probability of all execution times for that class on a given processor within the heterogeneous suite. To specify each distribution, the mean execution time is generated randomly in the range of [0.5, 4] s for each task-class-processor pair. As a result of this random approach, an unsorted $5 \times M$ ETC matrix models the inconsistent heterogeneity described in Section 2.2. If the elements of each row in this matrix are independently sorted in ascending order, the matrix models processor-consistent heterogeneity. If, in addition, the elements of each column in the matrix are independently sorted in ascending order, that matrix models task-processor-consistent heterogeneity. Note that as a new ETC matrix is created after each sorting procedure it represents a completely different distributed system.

A batch for each simulation trial consisted of 200 tasks. Each task was randomly associated with one of the five classes. Each task must be completed within a deadline d_i. If a task cannot be completed by its deadline then the request is considered timed out and will be discarded from the batch. The deadline for each task was set at a certain level $\underline{D}$ based on the longest mean execution time for that task class determined across all the processors. The performance of resource-allocation policies was explored with respect to two proportionality factors D: 300% and 600%.

Integer task rewards r_i were generated in two different ways. The *execution time independent rewards* method computed the reward r_i for each task i by sampling a uniform distribution in the range of (0, 100] regardless of task execution times t_{ij}. In contrast, the *execution time dependent rewards* method generated reward r_i based on a task class. Specifically, the interval (0, 100] was divided evenly into five subintervals, and r_i was determined by sampling a uniform distribution on a corresponding subinterval. For example, if task i belonged to the third class its reward value r_i was generated from subinterval (40, 60]. The second method appears to be plausible in many practical applications as it implies that tasks with longer execution times are more valuable in task-processor consistent systems.

Typically, hardware failures are rare when a distributed system operates under normal operating conditions [37]. However because the goal of our research is to maximize system performance under harsh operating conditions, the simulated mean time between failures was set for each processor relatively low, i.e., within the range of [0.6, 1]. As mentioned before, a processor in the simulation model becomes available either just after a hardware failure or just after a task completion. Once a task is assigned to a processor it is removed from the batch; if it fails it is returned to the batch. A task is discarded when its deadline expires. Six processors were used in our simulations.

The performance of the following five different resource-allocation policies was explored in the given environment.

(1) In the *deadline* policy, a task with the closest deadline d_i is assigned to the next available processor.
(2) In the *reward* policy, a task with the highest reward r_i is assigned to the next available processor.
(3) In the *reward_P* policy, a task with the highest value $r_i\mathbb{P}_i(t)$ is assigned to the next available processor, where $\mathbb{P}_i(t)$ is computed as described in Section 2.4.
(4) In the *match-making* policy, once a processor becomes available, a resource-allocation system follows the path of the DLR framework:
   (a) It recomputes the distribution of processor availability, as explained in Section 2.3. This step is required as a prerequisite to obtain a pmf used in DLR's computation of $a_{i,N(t)}$ values.
   (b) It calculates $a_{i,N(t)}$, or "bin boundaries", for $N(t)$ intervals as explained in the DLR theorem, Section 3.
   (c) Based on processor characteristics, $-\lambda_j t_j^{\text{av}}$, it determines index $k \in [1, N(t)]$ of the "bin" that the arrived processor corresponds to.
   (d) It sorts the remaining $N(t)$ tasks in ascending order of $r_i,\ 1 \le i \le N(t)$ values.
   (e) In this sorted list of tasks, it selects a task with index k and assigns it to the arrived processor.
(5) In the *match-making_P* policy, the steps are the same as in the *match-making* policy, but the task list is sorted based on $r_i\mathbb{P}_i(t),\ 1 \le i \le N(t)$ values.

5. Experimental results

The resource-allocation policies described above were compared in the simulated environment based on their ability to maximize the cumulative reward, averaged over 100 simulation trials. For each of these trials performed for the same set of parameters (i.e., distributions, number of tasks, etc.), a random number generator was seeded differently. This allowed the performance of the resource-allocation policies to be explored over a broad range of samples pulled from the corresponding distributions. In each simulation trial, the ETC matrix was generated in a random fashion to model inconsistent heterogeneity. The same matrix was sorted in the row direction and, then, in the column direction for the processor-consistent and task-processor-consistent scenarios, respectively.

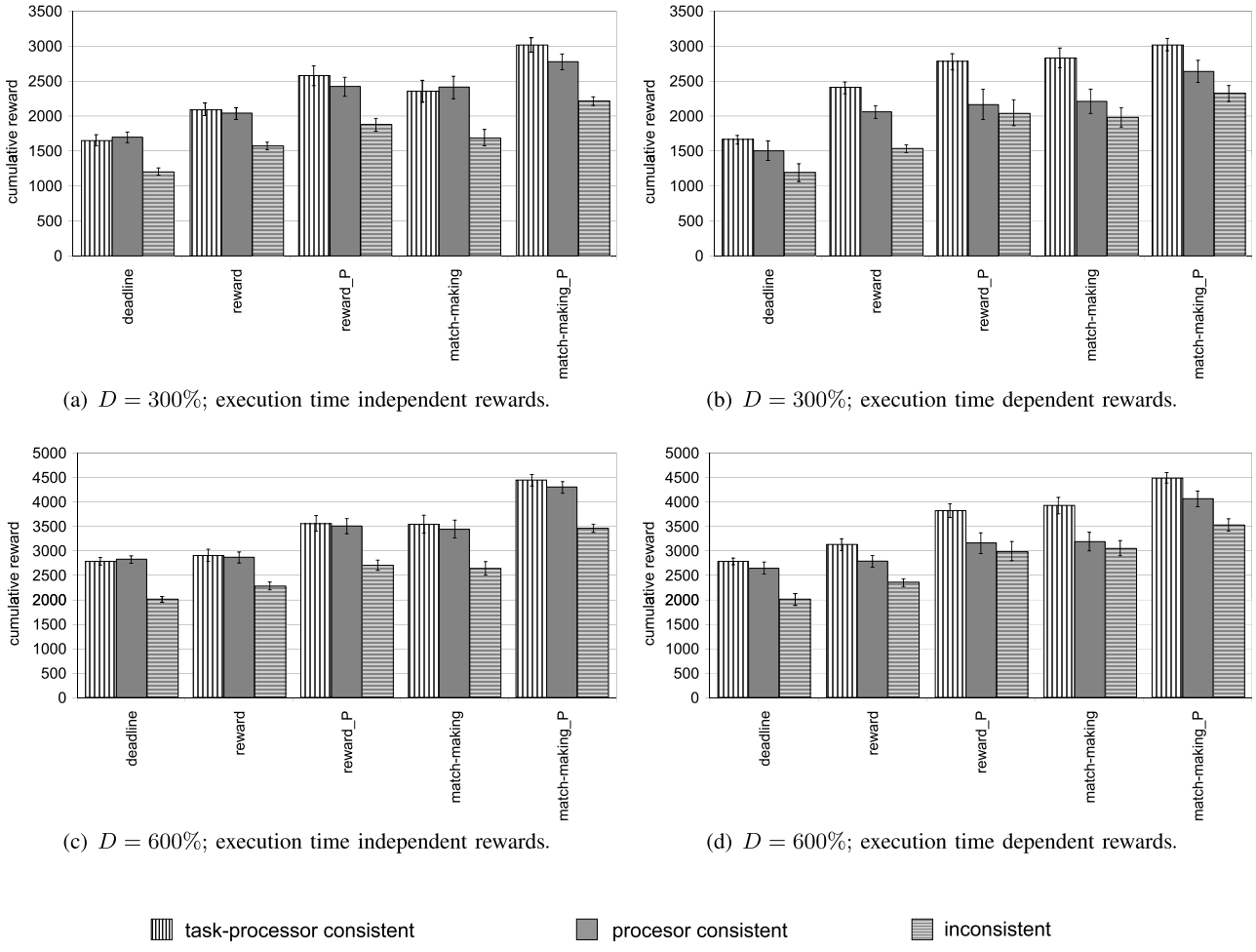
The average performance across 100 simulation trials along with the corresponding 95% confidence intervals are presented in Fig. 5 for all five resource-allocation policies. The *match-making_P* policy consistently delivers the best results for all three types of heterogeneity explored in this study. The superior performance of this policy is based on two factors incorporated into the decision making process: (1) the stochastic information describing the availability of processor types in the system, and (2) the probability $\mathbb{P}_i(t)$ estimated for task i to be successfully completed through multiple reassignments.

The second factor plays a very important role; once integrated into the *reward_P* policy, it improves its performance by 22% on average, as compared to *reward* policy, making its performance comparable to that of the *match-making* policy.

Fig. 6 demonstrates the progress of each policy over time in one of the simulation trials for processor-consistent heterogeneity with execution time independent rewards. This trial was selected because it reflects the average performance results plotted in Fig. 5(a). As expected, the *reward* and *reward_P* policies quickly accumulate the total reward in the beginning as they select the most profitable tasks first. When the number of such tasks becomes smaller, the total reward accumulation slows down as a result of more frequent task failures. Furthermore, closer to the end, the *reward* policy experiences a significant loss of tasks due to expired task deadlines.

Tables 1 and 2 show the total number of successfully completed tasks averaged over 100 simulation trials. Despite the fact that the *deadline* policy completes the highest number of tasks, its performance remains the worst because many of the tasks have low reward values. In contrast, the *match-making* and *match-making_P* policies have rather moderate numbers of completed tasks, but the demonstrated performance results highlight their ability of selecting tasks on a "more intelligent" basis to maximize the cumulative reward.

It is easy to observe that the average results shown in Fig. 5 demonstrate a significant difference in performance between the inconsistent and the processor-consistent types of heterogeneity. This can be explained by the fact that t_j^{av} values dominate in the $-\lambda_j t_j^{\text{av}}$ expressions, which are used to rank processor types as described in Section 2.2. Thus, the majority of rows in an ETC matrix



(a) $D = 300\%$; execution time independent rewards.

(b) $D = 300\%$; execution time dependent rewards.

(c) $D = 600\%$; execution time independent rewards.

(d) $D = 600\%$; execution time dependent rewards.

Fig. 5. The performance of five resource-allocation policies explored over the two methods of assigning rewards r_i and two setups for D.

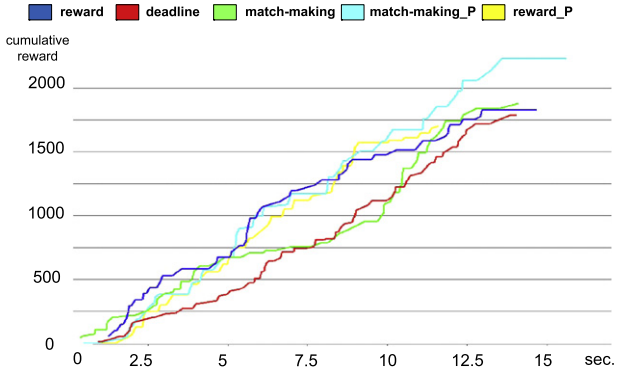


Fig. 6. A progress over time of the five resource-allocation policies with respect to the cumulative reward captured in one simulation trial for processor-consistent heterogeneity.

Table 1
Average numbers of successfully completed tasks computed across 100 simulation trials. Execution time independent rewards.

Policy	D (%)	Heterogeneity		
		Task-processor consistent	Processor consistent	Inconsistent
Deadline	300	48	47	44
Reward	300	33	35	29
Reward_P	300	37	31	27
Match-making	300	41	37	32
Match-making_P	300	43	38	33
Deadline	600	75	73	61
Reward	600	64	57	53
Reward_P	600	63	58	50
Match-making	600	66	69	51
Match-making_P	600	67	67	53

for the processor-consistent type of heterogeneity will be ranked in the same manner as the processor types in the DLR framework.

A task-processor consistent heterogeneity in the *match-making* policy does not result in an additional performance improvement with *execution time independent* rewards because only r_i values are used to rank tasks. The slight improvement for this reward generation method is observed in the *match-making_P* policy due to the fact that t_{ij} values are involved in the method of computing $\mathbb{P}_i(t)$, and, consequently, $r_i\mathbb{P}_i(t)$ are used to characterize tasks in that policy. However, when the reward generation method is changed to *execution time dependent*, a significant performance improvement is obtained in the task-processor consistent case (see Fig. 5). This can be explained by the fact that all the columns in the ETC matrix were ranked exactly as required in the DLR framework.

Both *match-making* and *match-making_P* general policies demonstrate a better performance than simple *deadline* and *reward* policies, but their computational complexity might be prohibitively expensive for some practical implementations, e.g., embedded systems. Although this problem is partially alleviated by the fact that policy computation is offloaded from processors to a separate front-end resource manager, the recursive computation of bin boundaries, i.e., $a_{i,N(t)}$ could take a rather long time especially for large N.

Table 2
Average numbers of successfully completed tasks computed across 100 simulation trials. Execution time dependent rewards.

Policy	D (%)	Heterogeneity		
		Task-processor consistent	Processor consistent	Inconsistent
Deadline	300	48	47	28
Reward	300	42	37	26
Reward_P	300	39	35	24
Match-making	300	43	43	26
Match-making_P	300	43	41	25
Deadline	600	77	73	61
Reward	600	64	50	43
Reward_P	600	63	60	51
Match-making	600	66	65	54
Match-making_P	600	68	58	50

6. Related work

The original work of Derman et al. [10] inspired research in various areas. Example applications of the DLR theorem include selling houses and job-search strategy [1]. In 1972, Albright and Derman determined the limiting behavior of the $a_{i,N}$'s as N becomes large [3]. The derived closed form solution can be used to avoid lengthy computation of $a_{i,N}$'s in such cases. Later, these authors addressed cases where the arrival process is a non-homogeneous Poisson process with a general discount function [2]. Nakai permits the distribution of resources to change according to a partially observable Markov process and allows the number of resources to be random [23]. Sakaguchi allows a fixed time horizon [29] and permits resource values to be dependent [30]. The results of these studies were applied to investment strategies [28], firing torpedoes at randomly arriving targets [30], allocating organs for transplants [9], and manufacturing and telecommunications [27]. Similar to our work, all the aforementioned studies address a sequential assignment problem in different problem domains, i.e., at each time when the resource-allocation system observes a realization of random variable, it must select the best action, one at a time in sequential order.

As mentioned before, the fault tolerant aspect of modern distributed computing systems has been extensively explored. However, the available literature on distributed computing in such uncertain environments primarily considers reactive techniques, where a node failure is addressed only after its occurrence [7]. One of the few exceptions is the paper of Dhakal et al. [12] that presents two preemptive load-balancing policies for a heterogeneous distributed computing system with wireless links between nodes. Preemptiveness in this case implies adjusting actions to compensate for the possibility of node failure/recovery. The main goal for these policies is to avoid a scenario where a node fails while having a large amount of unprocessed load. The data transfer of such a load to other nodes may result in a large random delay over wireless channels with following idle times on other nodes. A probabilistic model, based on the concept of regenerative processes, is presented to assess the overall performance of the system under these policies. The experiments show that preemptively utilizing the statistical information about the failure and recovery processes, to adjust the load-balancing gain to an optimal value, allows one to minimize the mean of the overall completion time of the total workload. Although the problem domain differs from ours, [12] exemplifies an effective integration of the available node failure/recovery statistics into the resource-allocation process.

7. Conclusion

This paper presents a method for robust static resource allocation in distributed systems under high failure rate. Given a batch of tasks, a resource-allocation system assigns tasks to compute resources that become available just after recovery from failures or task completions. The major contribution is the design of a resource-allocation policy for the above environment based on the concepts of the Derman–Lieberman–Ross theorem. The derived policy maximizes the expected cumulative reward received from the tasks that finish before their deadlines, given that the compute resources fail in a random fashion.

First, the resource-allocation policy was derived for the case where tasks and processors are categorized by the number of instructions and the time required to execute one instruction, respectively. Further, this policy was adapted to accommodate ETC matrices that provide a more accurate means of capturing the relationship among tasks and processors in distributed systems. As this study focused on dedicated distributed systems with a *limited* set of compute resources, the distribution describing the relative random availability of each processor was derived in Section 2.3 based on the characteristics of the tasks batched and the parameters of the processors available in the system.

The superior performance demonstrated by the *match-making_P* resource-allocation policy reveals its great potential for a variety of applications in a broad spectrum of distributed systems. For example, the problem of maximizing the performance when the system experiences temporal failures of compute resources is an important issue for embedded systems (e.g., [34,24]), sensor networks (e.g., [38]), or special purpose cluster-based systems (e.g., [33]). Similar to the environment considered in this work, such systems sometimes are employed under harsh conditions but must deliver a certain level of performance to remain in operation. The application domains include surveillance for homeland security, monitoring vital signs of medical patients, and automatic target recognition systems. Thus, the proposed DLR-based resource-allocation scheme can be adapted in those systems.

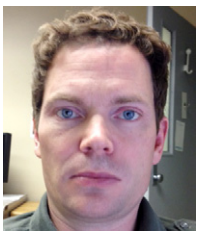
Acknowledgments

The authors thank Bhavesh Khemka for his valuable comments.

References

[1] S.C. Albright, Optimal sequential assignments with random arrival times, Management Science 21 (1) (1974) 60–67.
[2] S.C. Albright, A Bayesian approach to a generalized house selling problem, Management Science 24 (4) (1977) 432–440.
[3] S.C. Albright, C. Derman, Asymptotic optimal policies for the stochastic sequential assignment problem, Management Science 19 (1) (1972) 46–51.
[4] S. Ali, A.A. Maciejewski, H.J. Siegel, Perspectives on robust resource allocation for heterogeneous parallel systems, in: S. Rajasekaran, J. Reif (Eds.), Handbook of Parallel Computing: Models, Algorithms, and Applications, Chapman & Hall/CRC Press, Boca Raton, FL, 2008, pp. 41.1–41.30.
[5] Amazon elastic compute cloud. Accessed Dec. 01, 2011. [Online]. Available: http://aws.amazon.com/ec2/.
[6] Big ip product family. Accessed Dec. 01, 2011. [Online]. Available: http://www.f5.com/products/big-ip/.
[7] K. Birman, Reliable Distributed Systems, Springer-Verlag, New York, NY, 2005.
[8] CONDOR: High throughput computing. Accessed December 6, 2011. [Online]. Available: http://research.cs.wisc.edu/condor/.
[9] I. David, U. Yechiali, A time-dependent stopping problem with application to live organ transplants, Operations Research 33 (3) (1985) 491–504.
[10] C. Derman, G.J. Lieberman, S.M. Ross, A sequential stochastic assignment problem, Management Science 18 (7) (1972) 349–355.
[11] J.L. Devore, Probability and Statistics for Engineering and Sciences, fifth ed., Duxbury Press, Los Angeles, CA, 1999.
[12] S. Dhakal, M.M. Hayat, J.E. Pezoa, C.T. Abdallah, J.D. Birdwell, J. Chiasson, Load balancing in the presence of random node failure and recovery, in: The 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, 2006, pp. 25–29.
[13] C. Ebeling, Introduction to Reliability and Maintainability Engineering, in: Probability and Statistics, Waveland Pr. Inc., Long Grove, IL, 2009.
[14] S. Garg, A. Moorsel, K. Vaidyanathan, K.S. Trivedi, A methodology for detection and estimation of software ageing, in: The Ninth International Symposium on Software Reliability Engineering, November 1998, pp. 283–292.

[15] E. Gelenbe, D. Finkel, S.K. Tripathi, On the availability of a distributed computer system with failing components, ACM SIGMETRICS Performance Evaluation Review 13 (2) (2000) 6–13.
[16] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, third ed., Morgan Kaufmann, San Francisco, CA, 2003, Chapter 8.
[17] O.H. Ibarra, C.E. Kim, Heuristic algorithms for scheduling independent tasks on non-identical processors, Journal of the ACM 24 (2) (1977) 280–289.
[18] Infoprint. Accessed Dec. 02, 2011. [Online]. Available: http://www.infoprint.com/internet/ipww.nsf/vwWebPublished/print_infoprint-5000_en.
[19] J.-K. Kim, S. Shivle, H.J. Siegel, A.A. Maciejewski, T. Braun, M. Schneider, S. Tideman, R. Chitta, R.B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, S.S. Yellampalli, Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment, Journal of Parallel and Distributed Computing 67 (2) (2007) 154–169.
[20] H.M. Lee, S.H. Chin, J.H. Lee, D.W. Lee, K.S. Chung, S.Y. Jung, H.C. Yu, A resource manager for optimal resource selection and fault tolerance service in grids, in: 10th IEEE International Symposium on Cluster Computing and the Grid, 2004.
[21] L. Li, K. Vaidyanathan, K. Trivedi, An approach for estimation of software ageing in a web server, in: The 2002 International Symposium on Empirical Software Engineering, ISESE'02, 2002, pp. 91–100.
[22] E. Marshall, Fatal error: how patriot overlooked a scud, Science (1992) 13–19.
[23] T. Nakai, A sequential stochastic assignment problem in a partially observable Markov chain, Mathematics of Operations Research 11 (2) (1986) 230–240.
[24] S.I. Park, V. Raghunathan, M.B. Srivastava, Energy efficiency and fairness tradeoffs in multi-resource, multi-tasking embedded systems, in: The 2003 International Symposium on Low Power Electronics and Design, ISLPED'03, August 2003, pp. 2–13.
[25] Rackspace cloud. Accessed Dec. 01, 2011. [Online]. Available: http://www.rackspace.com/.
[26] M. Rausand, A. Hoyland, System Reliability Theory: Models, Statistical Methods, and Applications, second ed., in: Probability and Statistics, Wiley-Interscience, New York, NY, 2003.
[27] R. Righter, Stochastically maximizing the number of successes in a sequential assignment problem, Journal of Applied Probability 27 (2) (1990) 351–364.
[28] V. Saario, Limiting properties of the discounted house-selling problem, European Journal of Operational Research 20 (2) (1985) 206–210.
[29] K. Sakaguchi, A sequential stochastic assignment problem associated with a non-homogeneous Markov process, Japanese Journal of Mathematics 29 (1984) 13–22.
[30] K. Sakaguchi, Best choice problems for randomly arriving offers during a random lifetime, Japanese Journal of Mathematics 31 (1986) 107–117.
[31] Seti at home. Accessed March 31, 2008. [Online]. Available: http://setiathome.berkeley.edu.
[32] R. Sheahan, L. Lipsky, P. Fiorini, The effect of different failure recovery procedures on the distribution of task completion times, in: IEEE Workshop on Dependable Parallel, Distributed and Network–Centric Systems (DPDNS05), April 2005.
[33] V. Shestak, J. Smith, A.A. Maciejewski, H.J. Siegel, Stochastic robustness metric and its use for static resource allocations, Journal of Parallel and Distributed Computing 68 (8) (2008) 1157–1173.
[34] I. Shin, I. Lee, Periodic resource model for compositional real-time guarantees, in: 24th IEEE Real-Time System Symposium, RTSS 2003, December 2003, pp. 469–474.
[35] V. Subramani, R. Kettimuthu, S. Srinivasan, P. Sadayappan, Distributed job scheduling on computational grids using multiple simultaneous requests, in: 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), July. 2002, pp. 359–368.
[36] L. Wang, H.J. Siegel, V.P. Roychowdhury, A.A. Maciejewski, Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach, Journal of Parallel and Distributed Computing 47 (1) (1997) 8–22.
[37] A. Wood, Predicting client/server availability, Computer 28 (4) (1995) 41–48.
[38] T. Yuan, J. Boangoat, E. Ekici, F. Ozguner, Real-time task mapping and scheduling for collaborative in-network processing in dvs-enabled wireless sensor networks, in: 20st International Parallel and Distributed Processing Symposium, IPDPS 2006, April 2006, pp. 21–27.



Vladimir Shestak received a Ph.D. degree from the Department of Electrical and Computer Engineering at Colorado State University. He is currently a Senior Software Engineer at InfoPrint Solution Company, Boulder CO. He received his M.S. degree in Computer Engineering from New Jersey Institute of Technology in May 2003. His research interests include resource management within distributed computing systems, algorithm parallelization, and computer network design and optimization.

Edwin K.P. Chong received the B.E. (Hons.) degree with First Class Honors from the University of Adelaide, South Australia, in 1987; and the M.A. and Ph.D. degrees in 1989 and 1991, respectively, both from Princeton University, where he held an IBM Fellowship. He joined the School of Electrical and Computer Engineering at Purdue University in 1991, where he was named a University Faculty Scholar in 1999. Since August 2001, he has been a Professor of Electrical and Computer Engineering, and Professor of Mathematics, at Colorado State University. His current research interests span the areas of stochastic modeling and control, optimization methods, and communication and sensor networks. He coauthored the best-selling book, *An Introduction to Optimization, 3rd Edition*, Wiley-Interscience, 2008. He received the NSF CAREER Award in 1995 and the ASEE Frederick Emmons Terman Award in 1998. He was a co-recipient of the 2004 Best Paper Award for a paper in the journal *Computer Networks*. In 2010, he received the *IEEE Control Systems Society Distinguished Member Award*. Professor Chong is a Fellow of the IEEE. He was founding chairman of the IEEE Control Systems Society Technical Committee on Discrete Event Systems, and served as an IEEE Control Systems Society Distinguished Lecturer. He is currently a Senior Editor of the *IEEE Transactions on Automatic Control*, and also serves on the editorial boards of *Computer Networks*, the *International Journal of Control Science and Engineering*, and *IEEE Expert Now*. He is a member of the IEEE Control Systems Society Board of Governors (2006–2008, 2010–present). He has also served on the organizing committees of several international conferences. He has been on the program committees for the IEEE Conference on Decision and Control, the American Control Conference, the IEEE International Symposium on Intelligent Control, IEEE Symposium on Computers and Communications, and the IEEE Global Telecommunications Conference. He has also served in the executive committees for the IEEE Conference on Decision and Control, the American Control Conference, the IEEE Annual Computer Communications Workshop, the International Conference on Industrial Electronics, Technology & Automation, and the IEEE International Conference on Communications. He was the Conference (General) Chair for the *Conference on Modeling and Design of Wireless Networks*, part of SPIE ITCom 2001. He was the General Chair for the 2011 *Joint 50th IEEE Conference on Decision and Control and European Control Conference*.

Anthony A. Maciejewski received the BSEE, MS, and Ph.D. degrees from Ohio State University in 1982, 1984, and 1987. From 1988 to 2001 he was a professor of Electrical and Computer Engineering at Purdue University, West Lafayette. He is currently a Professor and Department Head of Electrical and Computer Engineering at Colorado State University. He is a Fellow of the IEEE, with research interests that include robotics and high performance computing. A complete vita is available at: http://www.engr.colostate.edu/~aam.

Howard Jay Siegel was appointed the Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU) in 2001, where he is also a Professor of Computer Science. He is the Director of the CSU Information Science and Technology Center (ISTeC), a university-wide organization for promoting, facilitating, and enhancing CSU's research, education, and outreach activities pertaining to the design and innovative application of computer, communication, and information systems. From 1976 to 2001, he was a professor at Purdue University. Prof. Siegel is a Fellow of the IEEE and a Fellow of the ACM. He received a B.S. degree in electrical engineering and a B.S. degree in management from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He has co-authored over 400 technical papers. His research interests include robust computing systems, resource allocation in computing systems, heterogeneous parallel and distributed computing and communications, parallel algorithms, and parallel machine interconnection networks. He was a Coeditor-in-Chief of the *Journal of Parallel and Distributed Computing*, and was on the Editorial Boards of both the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. For more information, please see www.engr.colostate.edu/~hj.