Future Generation Computer Systems 33 (2014) 1–10

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

Maximizing stochastic robustness of static resource allocations in a periodic sensor driven cluster

Jay Smith [a,b,*], Anthony A. Maciejewski [b], Howard Jay Siegel [b,c]

[a] Lagrange Systems, Boulder, CO 80302, USA

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

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

HIGHLIGHTS

- Novel methodology for creating robust resource allocations given a tight deadline.
- Resource allocation heuristics that maximize robustness.
- Local search operator for a Genetic Algorithm including search space analysis.
- Local search based path relinking crossover operator for a Genetic Algorithm.

ARTICLE INFO

Article history:
Received 15 December 2012
Received in revised form
27 September 2013
Accepted 4 October 2013
Available online 24 October 2013

Keywords:
Robustness
Heterogeneous computing
Resource management
Static resource allocation
Distributed computing

ABSTRACT

This research investigates the problem of robust static resource allocation for distributed computing systems operating under imposed Quality of Service (QoS) constraints. Often, such systems are expected to function in an environment where uncertainty in system parameters is common. In such an environment, the amount of processing required to complete a task may fluctuate substantially. Determining a resource allocation that accounts for this uncertainty—in a way that can provide a probability that a given level of QoS is achieved—is an important area of research. We have designed novel techniques for maximizing the probability that a given level of QoS is achieved. These techniques feature a unique application of both path relinking and local search within a Genetic Algorithm. In addition, we define a new methodology for finding resource allocations that are guaranteed to have a non-zero probability of addressing the timing constraints of the system. We demonstrate the use of this methodology within two unique steady-state genetic algorithms designed to maximize the robustness of resource allocations. The performance results for our techniques are presented for a simulated environment that models a heterogeneous cluster-based radar data processing center.

1. Introduction

In parallel and distributed computing, multiple compute nodes are collectively utilized to simultaneously process a set of applications to improve the performance over that of a single processor [1,2]. Often, such computing systems are constructed from a heterogeneous mix of machines that may differ in their capabilities, e.g., available memory, number of floating point units, clock speed, and operating system. This paper investigates robust resource allocation for a large class of heterogeneous computing (HC) systems that operate on *periodically updated* sensor data sets. Sensors (e.g., radar systems, sonar) in this environment produce new data sets at a fixed period Λ (see Fig. 1). Often, the period at which the sensor produces new data sets is fixed by the sensor and cannot be extended to account for long-running applications. Because these sensors typically monitor the physical world, the characteristics of the data sets they provide may vary in a manner that impacts the execution times of the applications that must process them. Suppose that each input data set must be processed by a collection of N independent applications that can be executed concurrently on the available set of M heterogeneous compute nodes. The goal of resource allocation heuristics in this environment is to allocate the N tasks to the M compute nodes such that all of the applications finish each data set in less than Λ time units, i.e., the makespan of a resource allocation must be less than or equal to Λ.

In this environment, the allocation of compute nodes to applications can be considered static, i.e., all of the applications that are to be executed are known in advance and are immediately available for execution upon the arrival of a new data set. Furthermore,

* Corresponding author at: Colorado State University, Department of Electrical and Computer Engineering, Fort Collins, CO 80523–1373, USA. Tel.: +1 7208395378.
E-mail addresses: jay@lagrangesystems.com (J. Smith), aam@colostate.edu (A.A. Maciejewski), hj@colostate.edu (H.J. Siegel).

0167-739X/$ – see front matter
http://dx.doi.org/10.1016/j.future.2013.10.001

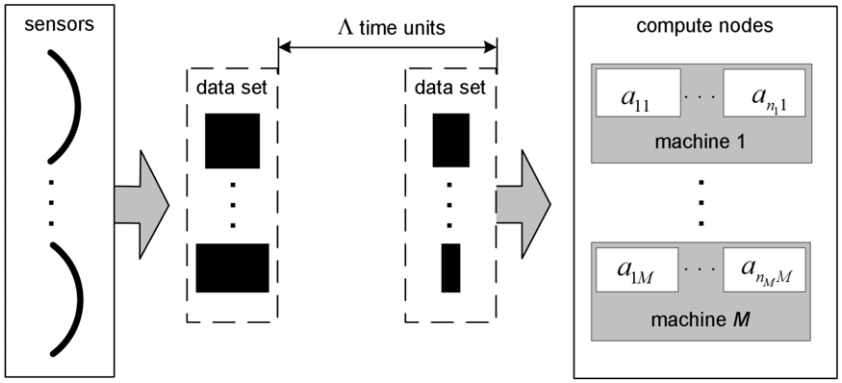

Fig. 1. Major functional units and data flow for a class of systems that must periodically process data sets from a collection of sensors, where a_{ij} corresponds to the ith application assigned to machine j.

the resource allocation remains fixed over a sequence of data sets. We assume that historical application execution time data for each application on each compute node in the HC system is available.

Because this is an HC system, the execution times for each of the N independent applications differ across the M compute nodes. Resource allocation in such computing environments has been shown in general to be an NP-hard problem (e.g., [3,4]). Thus, designing techniques for resource management in such environments is an active area of research (e.g., [5,6,1,7–10]).

We consider an environment where the required time period Λ is fixed by the sensor platform. Because a new data set arrives every Λ time units, the completion time of all applications must be less than or equal to Λ. Thus, ensuring that the system is ready to begin processing the next data set upon arrival without delay. However, unpredictable differences in the characteristics of the input data sets may result in a significant change in the execution times of the applications that must process the data. These differences may prevent some applications from completing because it would cause the makespan of the resource allocation to exceed Λ. *Robust* design for such systems involves determining a resource allocation that can account for uncertainty in application execution times in a way that maximizes the probability that all applications will complete within Λ time units. This probability is used to quantify robustness in this environment.

We first demonstrate several techniques for maximizing the robustness of resource allocations using both a greedy heuristic and more complex approaches, such as Genetic Algorithms. We show that the complex heuristics perform well given a "loose" Λ completion time constraint. However, given a tight Λ constraint all of the initial approaches routinely fail to produce resource allocations with a non-zero probability of completing by Λ. Our results show that by first minimizing the expected makespan of our resource allocations, we can easily find resource allocations with very small but provably non-zero robustness values. Although these resource allocations are poor solutions to our original problem of finding robust resource allocations, they can be used as seeds for our Genetic Algorithm techniques for maximizing robustness. The results of our simulation study demonstrate the utility of this approach for maximizing the robustness of resource allocations given both loose and tight Λ completion time constraints.

The major contributions of this paper include (1) a novel methodology for generating robust resource allocations given tight completion time constraints, (2) the design of resource allocation heuristics that can leverage this methodology to maximize the robustness of resource allocations subject to a constraint on the maximum allowable completion time Λ, (3) design of a local search technique for this problem domain and the associated search space analysis, and (4) the design of a path relinking crossover operator that utilizes local search within a Genetic Algorithm.

The remainder of this work is organized in the following manner. Section 2 describes the model of stochastic compute node completion times used in this work. A brief introduction to the stochastic robustness framework is presented in Section 3 along with an introduction to using the stochastic robustness metric within a heuristic. Four search based algorithms designed for this environment and one simple greedy heuristic are described in Section 4. The parameters of the simulation study used to evaluate a direct maximization of robustness are discussed in Section 5. Section 6 defines our methodology for generating resource allocations that are guaranteed to have non-zero robustness even for tight Λ values. This section presents the details of a steady-state genetic algorithm that successfully applies this approach for defining an initial population that can be used in each of our complex heuristics. The results of this combined approach are provided in Section 6.4. A sampling of the relevant related work is presented in Section 7. Section 8 concludes the paper.

2. System model environment

In this environment, we are concerned with allocating a set of N independent applications to a collection of M dedicated heterogeneous compute nodes. Each of the N applications must process data that is produced by the system's sensors. In a static resource allocation, all of the applications to be executed are known in advance of performing a resource allocation. In this environment, although the data sets to be processed vary every Λ time units, the applications executed are the same and their assignment to compute nodes remains fixed.

Application execution times are known to be data dependent. The data sets produced by the system sensors vary with changes in the real world, causing application execution times to vary in unpredictable ways. For this reason, we model the execution time of each application i ($1 \leq i \leq N$) on each compute node j ($1 \leq j \leq M$) as a random variable [11], denoted by η_{ij}. We assume that, based on historical data or experiments, probability mass functions (pmfs) describing the possible execution times for each application on each compute node exist and are available for each η_{ij}. (See [12] for some techniques for estimating these pmfs.)

Using the execution time pmfs, we can produce a completion time distribution for each compute node for a given resource allocation. The finishing time of each compute node is calculated as the sum of the execution time random variables for each application assigned to that compute node [13]. Let n_j be the number of applications assigned to compute node j. The pmf for the finishing time of compute node j, referred to as a local performance characteristic ψ_j, can be expressed as follows:

$$\psi_j = \sum_{i=1}^{n_j} \eta_{ij}. \tag{1}$$

Thus, the system makespan, denoted by ψ, can be expressed in terms of the local performance characteristics as follows:

$$\psi = \max\{\psi_1, \ldots, \psi_M\}. \quad (2)$$

If the execution times η_{ij} for the applications assigned to compute node j are mutually independent, then the summation of Eq. (1) can be computed using an $(n_j - 1)$-fold convolution of the corresponding pmfs [14,13].

3. Problem statement

Resource allocation decisions are often based on estimated values of task and system parameters, whose actual values are uncertain and may differ from available estimates. A resource allocation can be considered "robust" if it can mitigate the impact of uncertainties in system parameters on a given performance objective [15]. Any claim of robustness for a given system must answer these three questions [16]: (a) What behavior makes the system robust? (b) What are the uncertainties that the system is robust against? (c) Quantitatively, exactly how robust is the system? A robustness measure for this environment was first derived in [13] using these three questions and is summarized below.

For this system, the performance feature of interest is makespan, ψ. A resource allocation can be considered *robust* if the actual finishing time of each compute node is less than or equal to the periodicity of the data set arrivals Λ, i.e., $\psi \leq \Lambda$. We refer to this as the robustness requirement of the system.

Uncertainty in the system arises because the exact execution time for each application is not known in advance of its execution. That is, each of the execution time random variables η_{ij} is a source of uncertainty in a resource allocation. Because of its functional dependence on the execution time random variables, the system makespan is itself a random variable. That is, the uncertainty in application execution times can have a direct impact on the performance metric of the system.

To determine exactly how robust the system is under a specific resource allocation, we conduct an analysis of the impact of uncertainty in system parameters on our chosen performance metric. For a given resource allocation, the stochastic robustness measure provides the probability that the generated system performance will satisfy our robustness requirement. Clearly, unity is the most desirable value of the stochastic robustness measure, i.e., there is a 100% chance that the system will meet the established robustness requirement. For each compute node j, its robustness value, $\mathbb{P}[\psi_j \leq \Lambda]$, is found as the sum over the impulses in the pmf describing ψ_j whose impulse values are less than or equal to Λ.

The stochastic robustness measure, denoted by θ, is defined as the probability that the performance feature of the system is less than or equal to Λ, i.e., $\theta = \mathbb{P}[\psi \leq \Lambda]$. Because there are no inter-task data transfers among the applications to be assigned, the random variables for the local performance characteristics $(\psi_1, \psi_2, \ldots, \psi_M)$ are mutually independent. As such, the stochastic robustness metric for a resource allocation can be found as the product of the probability that each local performance characteristic is less than or equal to Λ. Mathematically, this is given as:

$$\theta = \prod_{\forall 1 \leq j \leq M} \left(\mathbb{P}\left[\psi_j \leq \Lambda\right] \right). \quad (3)$$

Intuitively, the stochastic robustness of a resource allocation defines the probability that all of the applications will complete within the allotted time period Λ. In this research, we are provided with a constraint Λ on the maximum time period required to process all applications and attempt to derive a resource allocation that maximizes robustness, i.e., the probability that all applications will complete by this deadline.

Algorithm 1 Pseudocode for the Two-Phase Greedy heuristic.

while not all applications are mapped **do**
  for each unmapped application a_i **do**
    find the compute node m_k such that,
    $k \leftarrow \text{argmin}^*_{1 \leq j \leq M} \left\{ \sum_i^{n_j} \mathbb{E}\left[\eta_{ij}\right] \right\}$
    resolve ties arbitrarily
  end for
  from all (a_i, m_k) pairs found above select pair (a_x, m_y) with the smallest completion time
  resolve ties arbitrarily
  map application a_x to compute node m_y
end while

*Argmin is an operation that returns the value of the argument for which the given expression attains its minimum value.

4. Heuristics

4.1. Two-phase greedy

The Two-Phase Greedy heuristic is based on the principles of the Min–Min algorithm (first presented in [4], and shown to perform well in many environments, e.g., [1,17]). The heuristic requires N iterations to complete, resolving a single application to compute node assignment during each iteration.

In the first phase of each iteration, the heuristic determines the application to compute node assignment that minimizes the individual expected completion time for each of the applications left unmapped. In the second phase, the heuristic selects the application to compute node assignment pair from the first phase that has the smallest expected completion time. Pseudo-code for the Two-Phase Greedy heuristic is provided in Fig. 1.

In this heuristic, we chose to minimize the expected completion time of applications as opposed to maximizing robustness because early in building an allocation, many of the possible application/compute-node assignments will have identical robustness values, i.e., unity. This happens because many of the applications have yet to be assigned, therefore, the subsets that have already been assigned are all guaranteed to finish before Λ (i.e., $\theta = 1$). Thus, an application of the Min–Min algorithm that directly maximizes robustness in this environment is ineffective.

4.2. Genetic algorithm

Our genetic algorithm (GA) was motivated by the Genitor evolutionary heuristic first introduced in [18]. A complete resource allocation is modeled as a sequence of numbers, referred to as a chromosome, where the ith entry in the sequence corresponds to the compute node assignment for the ith application, denoted by a_i. The ordering of applications in a chromosome is not significant for this environment and can be considered arbitrary. The fitness of each chromosome is determined according to the robustness of the resource allocation it represents. That is, a higher robustness value θ indicates a more fit chromosome.

In our GA implementation, we maintain a sorted list of the 100 most fit chromosomes encountered, referred to as the population, where the chromosomes are listed in decreasing value of their robustness. The appropriate size of the population was selected through experimentation.

The initial members of the population were generated by applying the simple greedy heuristic from [13] and applying a local search operator to the result. The simple greedy heuristic randomly selects an application and assigns it to the compute node that provides the smallest application completion time. After a chromosome is produced by this simple greedy heuristic, we

apply the local search operator defined in Section 4.3 to produce a new chromosome that is a local maximum, i.e., the robustness of the chromosome cannot be further improved upon using our local search procedure. Before the generated chromosome can be inserted into the population, a check is performed to ensure that the new chromosome is unique. In this way, we ensure that no member of the initial population is over-represented. This chromosome generation process is repeated, each time with a new random application selection order, until the population size reaches its limit, e.g., 100 chromosomes.

Our GA operates in a *steady state* manner, i.e., for each iteration of the GA only a single pair of chromosomes is selected from the population for crossover [18]. Chromosome selection is performed using a linear bias function [18], based on the fitness of each chromosome. If the new chromosome generated is not already in the population, it is inserted in sorted order according to its fitness value. For each chromosome inserted, the least fit chromosome in the population is removed, so that the size of the population is held fixed.

The crossover operator was implemented using the two-point reduced surrogate procedure [18]. Crossover points are randomly selected such that at least one element of the parent chromosomes differs between the selected crossover points to guarantee offspring that are not clones of their parents. Thus, each execution of the crossover operator will produce two unique offspring.

The final step within each iteration of our GA implementation applies a mutation operator. For each iteration of the GA, the mutation operator selects a small percentage, referred to as the mutation selection rate, of the overall population and randomly alters the chromosome to produce a new chromosome. For the simulated environment, based on experimentation, we chose a mutation selection rate of 0.1. During mutation, each application to compute node assignment within the chromosome to be mutated is individually modified with a probability referred to as the mutation rate. For the simulated environment, the best results were achieved using a mutation rate of 0.02, determined through experimentation. Once an application/compute-node assignment has been selected for mutation, the mutation operator randomly selects a different compute node assignment for the chosen application. After mutation, the new chromosome, if it is unique, is inserted into the population, and the worst chromosome is dropped.

We limited the total number of chromosome evaluation function calls in our GA implementation to 4,000,000. For the simulation trials tested, this number of chromosome evaluation function calls enabled the GA to find a resource allocation that provided a near unity robustness value. The GA procedure is summarized in Fig. 2.

4.3. Genetic algorithm with local search (GALS)

The Genetic Algorithm with local search (GALS) heuristic is identical to our earlier GA procedure with the exception that a local search procedure is applied to every chromosome before it is inserted into the population. The local search procedure is conceptually analogous to the steepest descent technique [19]. In this way, the GALS heuristic attempts to confine the population of the GA to those solutions that are local maxima.

The local search implemented in GALS is similar to the fine refinement presented as part of the GIM heuristic in [20]. The local search relies on a simple four-step procedure to maximize θ relative to a fixed Λ value. First, for a given resource allocation, the compute node with the lowest individual probability to meet Λ is identified. From the applications assigned to this compute node, local search identifies the application that, if moved to a different compute node, would increase θ the most. This requires re-evaluating θ every time an application-compute node re-assignment is considered. However, because the machine

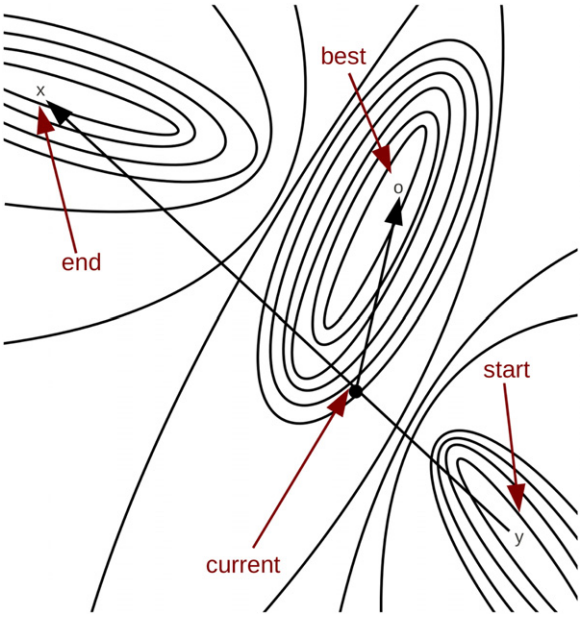

Fig. 2. A conceptual depiction of the path relinking procedure. Each level set or contour line in this depiction corresponds to a single robustness value. The search begins at chromosome *y*, labeled start in the figure, and generates a path to chromosome *x*, labeled end, using the path relinking procedure. Local search is applied to each current chromosome produced along the path from *y* to *x*. In this example, the path connecting *y* and *x* passes through an area that leads to the improved solution *o*.

Algorithm 2 Pseudo-code for the GA heuristic.

```
generate initial population using simple greedy heuristic
rank population in descending order based on θ values
while eval count < 4, 000, 000 do
  select two chromosomes from the population for crossover
  select crossover points
  exchange compute node assignments between crossover points
  if not already present in the population then
    insert resulting offspring into population
    if population size > limit then
      remove worst chromosome
    end if
  end if
  for each chromosome in the population do
    generate a random number x in the range [0,1]
    if x < mutation selection rate then
      for each application in the selected chromosome do
        generate a random number y in the range [0,1]
        if y < mutation rate then
          arbitrarily change compute node assignment of the
          selected application
        end if
      end for
    end if
  end for
  if not already present in the population then
    insert resulting offspring into population
    if population size > limit then
      remove worst chromosome
    end if
  end if
end while
output the best solution encountered
```

completion time distributions can be calculated independently, producing the new θ value only requires that we produce new completion time distributions for the two machines that participated in the reassignment, i.e., the complexity of calculating the new θ is greatly reduced. Once an application-compute node pair has been identified, the chosen application is moved to its chosen compute node.

If no application can be moved to a new compute node to obtain a strict improvement in θ, then we consider swapping two applications. In a swap, one of the two applications considered for swapping must currently be assigned to the compute node under consideration, while the second application is selected from the set of applications assigned to a different compute node. The pair of applications selected corresponds to the two that would improve θ the most. As in the reassignment case, the complexity of calculating the new θ value is limited to recomputing the completion time distributions for the two machines involved.

The procedure repeats from the first step until there are no application moves or swaps from the lowest probability compute node that would strictly improve θ. The procedure is then repeated for each compute node in increasing order of the probability that the local performance characteristic is less than or equal to Λ until no application can be moved to improve θ. For this procedure, it is assumed that $\theta < 1$; otherwise, no improvements can be made through local search.

4.4. Genetic algorithm with path relinking (GAPR)

Path relinking can be used as a means of combining two "parents" to form a new type of crossover operator [21]. The path relinking implementation of crossover begins with a pair of chromosomes, and draws a path from one parent to the other by exchanging one application/compute-node assignment at a time. Using path relinking, we form a new GA heuristic based on the GALS heuristic presented in the previous subsection by replacing the two-point reduced surrogate crossover operator with path relinking.

Our implementation of path relinking begins with a pair of chromosomes to be used as parents in the operation and selected as in the GA procedure. One is labeled the start chromosome and the other is labeled the end chromosome. Next, we initialize the "best chromosome encountered" to the parent chromosome with the highest robustness value and initialize the "current chromosome" to the start chromosome.

Starting with i initialized to 1, in the current chromosome, we change the compute node assignment of a_i to the assignment of a_i in the end chromosome. We copy the current chromosome into a temporary location and apply the local search operator from GALS to this temporary chromosome. Finally, if the robustness of the chromosome produced by the local search is greater than that of the best chromosome encountered, then we replace the best with this result. We then increment i and the procedure continues in this manner until the current chromosome is identical to the end chromosome. When this occurs, the procedure outputs the best chromosome encountered during path relinking as the offspring of the operation and exits. Note that if the machine assignment for a_i is the same in both the start and end chromosomes, we skip the local search and increment i. To generate a second offspring from this operator, we proceed as before, but we exchange the start and end chromosomes to follow a second path.

An example of the path relinking procedure is depicted in Fig. 2. In the figure, two chromosomes y and x are connected by transforming the start chromosome y into the end chromosome x. For each intermediate point along the path from y to x, we apply the local search procedure to find the resource allocation whose robustness defines a local optimum. Because all of the members of the GAPR population are local optima, the chromosomes y and x are themselves local optima. By transforming solution y into solution x one compute node assignment at a time, path relinking may encounter an intermediate chromosome which leads to the local optimum chromosome o. In our implementation, we produce only a single offspring (the "best" chromosome encountered) from each direction of the path relinking operator. Pseudo-code for our implementation of path relinking is provided in Fig. 3.

Algorithm 3 Pseudo-code of one direction of the path-relinking crossover procedure.

```
start ← first chromosome selected for crossover
end ← second chromosome selected for crossover
if θ(start) > θ(end) then
  best ← start
else
  best ← end
end if
current ← start
for i = 1 to N do
  if a_i in start ≠ a_i in end then
    a_i in current ← a_i in end
    result ← apply local search to current
    if θ(result) > θ(best) then
      best ← result
    end if
  end if
end for
offspring ← best
output offspring
```

4.5. Simulated annealing

The Simulated Annealing (SA) algorithm – also known in the literature as Monte Carlo annealing or probabilistic hill-climbing [17] – is based on an analogy taken from thermodynamics. In SA, a single solution, is generated by the Two-Phase Greedy heuristic, structured as a chromosome, and then iteratively modified and refined.

To deviate from the current solution in an attempt to find a better one, SA repeatedly applies the mutation operation of our steady-state GA, applying the local search operator to the mutated result prior to its evaluation. The mutation rate was determined through experimentation with our simulation trials and set to 10%. Once a new solution, denoted by S_{new}, is produced, a decision regarding the replacement of the previous solution, denoted by S_{old}, with S_{new} has to be made. If the fitness of the new solution, denoted by $\theta(S_{new})$, found after evaluation, is higher than the old solution, the new solution replaces the old one. Otherwise, SA will probabilistically allow poorer solutions to be accepted during the search process, which makes this algorithm different from other strict hill-climbing algorithms [17]. The probability of replacement is based on a system temperature, denoted by $\mathbb{T}$, that decreases with each iteration. As the system temperature "cools down", it becomes more difficult for poorer solutions to be accepted. Specifically, the SA algorithm selects a random number from the range [0, 1) according to a uniform distribution. If

$$random[0, 1) > \frac{1}{1 + \exp\left(\frac{\theta(S_{new}) - \theta(S_{old})}{\mathbb{T}}\right)}, \tag{4}$$

then the new poorer resource allocation is accepted; otherwise, the old solution is kept. As can easily be seen in Eq. (4), the probability for a new solution of similar quality to be accepted is close to 50%. In contrast, the probability that a much poorer solution is rejected is rather high, especially when the system temperature becomes relatively small.

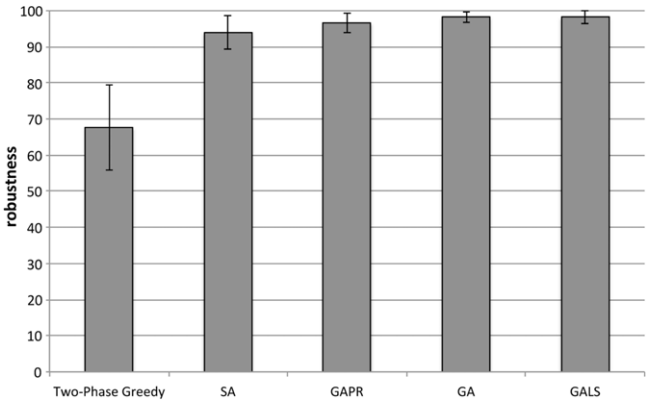

Fig. 3. A comparison of the average robustness values over 50 simulation trials for the Two-Phase Greedy, Simulated Annealing (SA), Genetic Algorithm with Path Relinking (GAPR), Genetic Algorithm (GA), and Genetic Algorithm with Local Search (GALS) heuristics.

After each mutation (described in Section 4.3) and subsequent local search, the system temperature $\mathbb{T}$ is reduced to 99% of its current value. This percentage, defined as the cooling rate, was determined experimentally by varying the cooling rate in the range [0.9, 1). The fitness of each chromosome, θ, is inherently bound to the interval [0, 1]. Consequently, only small differences between $\theta(S_{new})$ and $\theta(S_{old})$ are possible, causing Eq. (4) to remain very near 0.5 for large values of $\mathbb{T}$. Based on our experimentation, we set the initial system temperature in Eq. (4) to 0.1.

For the simulation trials tested, our implementation of SA was terminated after 4,000,000 chromosome evaluation function calls to enable a fair comparison with the results of all of our steady-state GA implementations. The SA procedure is summarized in Algorithm 4.

Algorithm 4 Pseudo-code for the Simulated Annealing heuristic.

```
S_old ← initial Two-phase greedy solution
𝕋 ← 0.1
while eval count < 4,000,000 do
  S_new ←mutate S_old
  apply local search to S_new
  if θ(S_new) > θ(S_old) then
    S_old ← S_new
  else if Eq. (4) holds then
    S_old ← S_new
  end if
  𝕋 ← 0.99 × 𝕋
end while
return S_old
```

5. Simulation study for loose Λ

5.1. Setup

In these simulations, the periodicity Λ of data set arrivals was assumed fixed at 540 time units. The value for the constraint Λ was selected to present a challenging resource allocation problem for our chosen heuristics (i.e., the resulting θ was neither 1 nor 0) based on the number of applications, the number of compute nodes, and the execution time pmfs used for η_{ij}. The goal of the heuristics is to maximize robustness.

To evaluate the performance of the heuristics described in Section 4, the following approach was used to simulate a cluster-based system for processing radar data sets. The execution time distributions for 28 different types of possible radar ray processing algorithms on $M = 8$ heterogeneous compute nodes were generated by combining experimental data with benchmark results. The experimental data, represented by two execution time sample pmfs, were obtained from experiments conducted on the Colorado MA1 radar [22]. These sample pmfs contain application execution times for 500 different radar data sets of varying complexity by the Pulse-Pair & Attenuation Correction algorithm [23] and by the Random Phase & Attenuation Correction algorithm [23]. Both applications were executed in non-multitasking mode on the Sun Microsystems Sun Fire V20z workstation. To simulate the execution of these applications on a heterogeneous computing system, each sample pmf was scaled by a performance factor corresponding to the performance ratio of a Sun Microsystems Sun Fire V20z to each of eight selected compute nodes[1] based on the results of the fourteen floating point benchmarks from the CFP2000 suite [24]. Combining the results available from the CFP2000 benchmarks with the sample pmfs produced by the two available applications provided a means of generating the 28×8 matrix of application execution times, where the (k, j) element in the matrix corresponds to the application execution time pmf of a possible ray processing algorithm of type k on compute node j.

Each simulation trial consisted of a set of 128 applications ($N = 128$). To evaluate the performance results of each heuristic, 50 simulation trials were conducted. For each *trial*, the type of each application was determined by randomly sampling integers in the range [1, 28].

5.2. Results

The results of the loose Λ simulation trials are presented in Fig. 3. The 50 simulation trials provide a good estimate of the mean as demonstrated by the relatively tight 95% confidence intervals for our results. The SA, GAPR, GA, and GALS heuristics were all able to improve upon the robustness values achieved by the Two-Phase Greedy solution. Note that for these results all of the heuristics were given an identical Λ constraint, thus, a comparison of the mean robustness values achieved by each heuristic is a direct comparison of the stochastic robustness Θ that each heuristic achieves versus the constant constraint Λ.

It is important to note that the confidence intervals of the SA, GAPR, GA, and GALS heuristics all overlap, indicating that no one heuristic is significantly better than another for these simulation trials. The comparable performance of the SA heuristic and the GA variants may be caused by the ease with which each heuristic is able to find solutions with near unity robustness. That is, the confidence intervals for all of these heuristics overlap and include a stochastic robustness value of 100%.

Finally, although the robustness achieved by the Two-Phase Greedy approach was significantly below that of the evolutionary heuristics, the execution time of the Two-Phase Greedy approach was significantly less than that of the other heuristics. For example, a typical run of the Two-Phase Greedy approach in this environment will complete in seconds where as all of the evolutionary approaches require hours to complete processing.

To better visualize the path-relinking procedure, we plotted the robustness attained by each step of the operation. Fig. 4 shows the results of an example application of the path relinking crossover operator taken from an actual simulation trial. The result was obtained by transforming one chromosome into another using our defined path relinking procedure. Recall that after each application to compute node assignment is modified, the local search operator

[1] The eight compute nodes selected to be modeled were: Altos R510, Dell PowerEdge 7150, Dell PowerEdge 2800, Fujitsu PRIMEPOWER 650, HP Workstation i2000, HP ProLiant ML370 G4, Sun Fire V65x, and Sun Fire X4100.

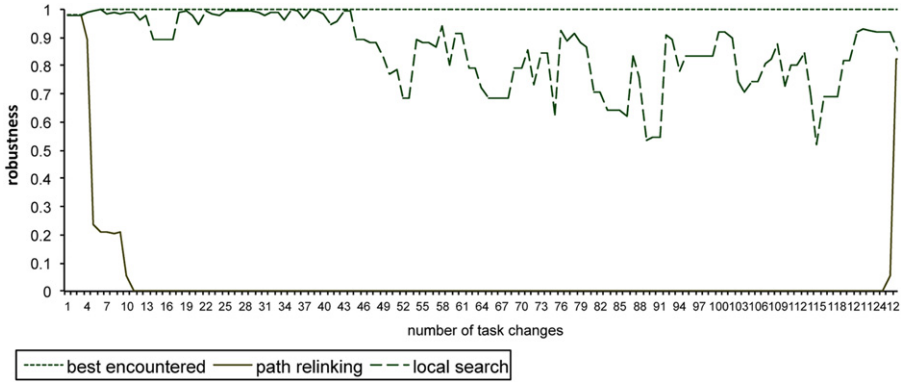

Fig. 4. An example result obtained by applying the path relinking crossover procedure in one direction.

is applied to the intermediate solution. The solid line of the figure corresponds to the robustness of the current chromosome before local search has been applied and the dashed line corresponds to the robustness of the allocation after applying local search. The dotted line provides an indication of the highest robustness value encountered during this run of path relinking. The first robustness value of the dashed line corresponds to the first parent in the crossover and the last robustness value corresponds to the robustness value of the alternate parent. In this example, the initial allocation of the first parent had a robustness value of 0.9808 and the alternate parent had a robustness of 0.8224. Applying the path relinking crossover operator produced a final allocation with a robustness value of 0.9981.

6. Maximizing robustness for tighter Λ values

6.1. Overview

Maximizing robustness using the heuristic approaches described earlier can function properly only if the robustness of the initial and intermediate solutions is non-zero. That is, in all of the presented techniques, intermediate resource allocations are directly compared to one another based on robustness and ranked accordingly. Thus, if the robustness of two allocations is equal to zero, then we cannot discriminate between them based on robustness alone. For this case, we would like to identify an alternative metric for evaluating resource allocations that can always discriminate between resource allocations whose robustness value is zero. The metric that we use is the expected makespan of a given resource allocation, which allows us to differentiate between solutions that both have a zero robustness value.

In general, this approach enables us to transform the resource allocation problem from the robustness space, where the fitness of each solution is limited to the range [0, 1], into a makespan problem where the dynamic range is not limited and can be any positive real number. In this section, we prove that if the makespan of a resource allocation is less than or equal to Λ then the robustness of that allocation will be non-zero. Using this technique, we can generate a population where all resource allocations have a non-zero robustness value. We then evaluate the robustness of each allocation to transform the population back into the robustness space and begin to directly maximize robustness using the complex heuristics of Section 4.

In the next subsection, we prove that if the expected makespan for a given resource allocation is less than or equal to Λ, then the stochastic robustness value of that allocation will be non-zero. In Section 6.3, we define a procedure for using this technique to seed our GA heuristics, ensuring that all members of the initial population have non-zero robustness values. This section concludes with our results in Section 6.4.

6.2. Alternative metric for evaluating allocations

Define the *expected* makespan of a resource allocation, denoted by $\underline{\mu}$, as the makespan of the resource allocation as found using the expectation of machine completion time distributions. Let $\underline{\mu_j}$ be the expected completion time of machine j, i.e., $\mu_j = \mathbb{E}[\psi_j] = \sum_{i=1}^{n_j} \mathbb{E}[\eta_{ij}]$. Thus, μ can be found by taking the max over all μ_j,

$$\mu = \max_{1 \le j \le M} \mu_j. \tag{5}$$

A useful property of the expected makespan of an allocation is that if $\mu \le \Lambda$ then θ is bounded below by $(0.5)^M$ and thus the robustness of the allocation is guaranteed to be non-zero. To prove this property, assume that $\mu \le \Lambda$. Define $\underline{\theta_j}$ to be the robustness of the completion time distribution for machine j relative to Λ. Recall that μ is the maximum expected completion time over all machines, which implies that $\mu_j \le \mu \le \Lambda (\forall j)$. If $\mu_j = \Lambda$, then θ_j must equal 0.5 because μ_j is the mean of the completion time distribution for machine j. Alternatively, if $\mu_j < \Lambda$, then $\theta_j > 0.5$. Thus,

$$\theta = \left(\prod_{1 \le j \le M} \theta_j \right) \ge (0.5)^M. \tag{6}$$

Therefore, if there are M compute nodes with $\mu \le \Lambda$, then $\theta \ge (0.5)^M$. This proves that if the expected makespan of a resource allocation is less than or equal to Λ, then the robustness of that allocation is non-zero. Because each pmf consists of probabilities for discrete time values (it is not continuous) and the expected completion time may not correspond to exactly one of these values, we replace the expected value of the pmf with the earliest possible outcome that is greater than the expected value, ensuring that Eq. (6) still holds.

In this case, if we can establish an initial population where each resource allocation satisfies $\mu \le \Lambda$, then our earlier robustness GA heuristics can be applied to cases where Λ is tight. We make the simplifying assumption that Λ has been selected such that resource allocations exist that can satisfy this constraint on μ.

To establish an initial population using this method, we can employ simpler minimization techniques that operate on the mean execution times of applications as opposed to the entire distribution of possible execution times. In the next subsection, we describe a GA that minimizes the expected makespan of each resource allocation (referred to as the Makespan-GA).

6.3. GA for establishing initial population

In our original robustness GA implementation from Section 4.2, we employed a simple greedy heuristic to generate the initial

population. Unfortunately, if the Λ constraint is too small, then this simple method of population generation will not produce solutions with non-zero robustness values. Based on the discussion of the previous subsection, we implemented a Makespan-GA that minimizes the expected makespan of allocations. The final population of the Makespan-GA can be used to define an initial population for the robustness-based GAs whose members are known to all have non-zero robustness values.

The implementation of the Makespan-GA operates in a manner similar to the GAPR heuristic of Section 4.4, using both the path relinking crossover operator and local search (but based on minimizing the expected makespan). The fitness of each resource allocation in the Makespan-GA is measured by the expected makespan of the allocation instead of its robustness. Based on our earlier analysis, if the expected makespan can be reduced to at least Λ, then the resulting robustness of the allocation will be non-zero. Thus, the Makespan-GA terminates when the expected makespan of the worst member of the population is less than or equal to Λ or a maximum number of iterations has elapsed. Recall that the population is maintained in a sorted order according to decreasing fitness value, i.e., the least fit chromosome is the last member of the population. If the Makespan-GA terminates because a maximum number of iterations has elapsed, then the robustness GA can be populated with any members produced by the Makespan-GA whose $\mu \leq \Lambda$. In addition, it may be possible to identify chromosomes where $\mu > \Lambda$ and the robustness of the solution is still non-zero. In our simulations, the Makespan-GA was always able to find a complete initial population whose expected makespan was less than or equal to Λ.

After the Makespan-GA terminates, we are left with a population where the robustness value of every chromosome in the population is non-zero. There are some final considerations that must be accounted for prior to using the population to seed our robustness-based GAs. Recall that in our earlier GA implementations we maintained the population in sorted order according to the fitness of each chromosome. However, the order of the chromosomes in the Makespan-GA is based on the expected makespan and not robustness. Thus, before the population can be used in our robustness-based GAs, each chromosome must be evaluated based on robustness and the population re-sorted. Further, although the Makespan-GA applied local search to each member of the population, the local search was based on the expected makespan. To ensure that the population consists only of local minima relative to robustness, we apply the robustness based local search operator to each chromosome, then insert the resulting chromosome into the population, using the insertion procedure defined earlier. Consequently, it is possible that after applying the robustness-based local search to each chromosome and rebuilding the population that some of the chromosomes will result in clones (i.e., duplicate chromosomes). Because the insertion procedure of the robustness-based GAs explicitly disallows clones, the initial population may begin with fewer chromosomes than our desired population size. However, the population will typically steadily grow back to the predetermined population size in the first few iterations of the GA and will remain fixed at our chosen population size for the remainder of the simulation. To demonstrate the application of this procedure, we use this technique to generate initial populations for all of the GA variants.

6.4. Results for tighter Λ values

To evaluate the effectiveness of generating an initial population for tighter Λ values using the Makespan-GA approach, we reran our simulation trials for the GAPR and GALS heuristics with a new Λ value that is tailored to each simulation trial. Because these heuristics are capable of leveraging the entire Makespan-GA population, as opposed to SA that modifies a single solution, they are ideal for use in this context. Although the basic GA can also leverage the population produced by the Makespan-GA, we chose not to use it in this context because it does not leverage the local search operator of GAPR and GALS. In these trials, the Λ constraint was uniquely set for each simulation trial based on the collection of tasks to be executed for that trial. In [13], we presented a mechanism for calculating a lower bound on Λ. In this work, we applied the lower bound calculation to each trial and used 120% of that lower bound value as the Λ constraint. This resulted in an average Λ constraint of 506 time units as opposed to the fixed Λ constraint of Section 5.2, i.e., $\Lambda = 540$.

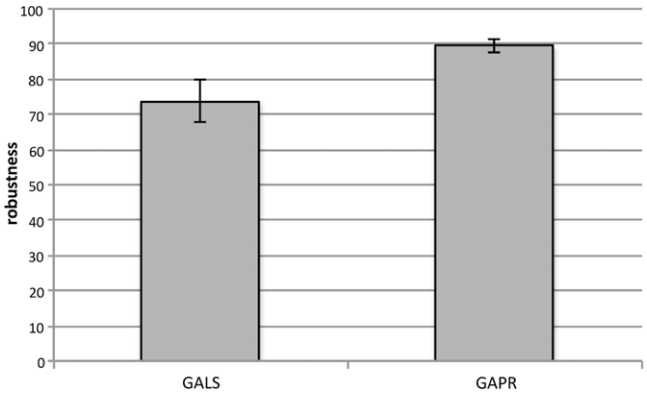

Fig. 5. A comparison of the average robustness values over 50 simulation trials for the GALS, and GAPR heuristics using the modified initial population. In these simulation trials, Λ was set using 120% of a known lower bound.

For this tighter Λ constraint, our original method of generating initial population members failed to produce any solutions with a non-zero robustness value. Consequently, given a fixed number of chromosome evaluations, i.e., 4,000,000, none of the GA variants were able to produce resource allocations with non-zero robustness values.

Alternatively, using the same number of chromosome evaluations as in our earlier trials, the GALS heuristic using the initial population from the Makespan-GA was able to produce solutions with an average robustness value of 73.8% with a 95% confidence interval of plus or minus 6% points. In addition, the GAPR heuristic using the new initial population generated an average robustness value of 89.6% with a 95% confidence interval of plus or minus 2% points. These results, plotted in Fig. 5, clearly demonstrate the utility of the proposed approach for generating resource allocations with high robustness values under tighter Λ constraints. From the plot, one may attempt to draw the conclusion that the GAPR heuristic is superior to GALS, however, this may not be universally true as evidenced by our earlier result for looser Λ constraints.

As a final comparison, we compared the results of the GAPR, and GALS heuristics using the initial population produced by the Makespan-GA to those of our initial methods for simulations where Λ was set to 540, as was done in Section 5.2. For these simulations, the GAPR and GALS heuristics both performed comparably using both methods of initial population generation. This suggests that the Makespan-GA method of generating an initial population is effective for problems with both loose Λ constraints and tight Λ constraints.

7. Related work

A universal framework for defining the robustness of resource allocations in heterogeneous computing systems was addressed in [15]. This work referred to the ability of a resource allocation to tolerate uncertainty as the *robustness* of that resource allocation and established the FePIA procedure for deriving a deterministic

robustness metric. In [13], the authors used the FePIA procedure to define a robustness metric for static stochastic resource allocation environments. However, the research in [13] focused on minimizing the makespan of a stochastic resource allocation subject to a constraint on the robustness of that allocation. In this current paper, we have shown that it is possible to instead maximize the robustness of a resource allocation given a constraint on the allowed makespan.

In [6], the problem of robust resource allocation was addressed for scheduling directed acyclic graphs (DAGs) in a heterogeneous computing environment. Robustness was quantitatively measured as the "critical" (i.e., the smallest) slack among all components that comprise a given DAG. The authors focused on designing resource allocations that maximized robustness for a deterministic environment. Whereas, our robustness metric is based on stochastic information about the uncertainties.

In [25], the authors present a comprehensive study of robustness measures suitable to a task graph allocation model. Using this study, they propose a suite of heuristics focused on balancing the tradeoff between robustness and makespan. In contrast, our work is focused solely on maximizing robustness given a constraint on the completion time of the allocation.

In a related area, Lombardi et al., investigate the application of robustness to scheduling tasks with precedence constraints in a multi-core environment where the task mapping is predetermined and each application is subject to a hard deadline. That is, in [26], the authors present a scheduling technique for a single multi-core processor and assume that the set of tasks assigned to that processor is given. As opposed to our research, where we address the mapping of tasks to processors and assume that each processor is single threaded. As such, [26] presents a clearly complimentary research environment to that studied here. In [26], task completion time variability is captured through bounds extracted from a worst case execution time analysis as opposed to directly capturing the stochastic information about execution time uncertainties as is done in this research.

Our methodology requires that the uncertainty in system parameters can be modeled as stochastic variables. A number of methodologies exist for modeling the stochastic behavior of application execution times (e.g., [27,28,12]). In [27], a method is presented for combining stochastic task execution times to determine task completion time distributions. Our work leverages this method of combining independent task execution time distributions and extends it by defining a means for measuring the robustness of a resource allocation against an expressed set of QoS constraints.

In [29], the authors present the application of stochastic robustness to measuring the impact of inaccurate execution time information on stochastic resource allocations. The authors also present three greedy heuristics for producing static resource allocations often used in the literature. The Min–Min heuristic that the authors present is analogous to the Two-Phase Greedy heuristic presented in this work. However, in their use of the Min–Min heuristic the authors limit the pool of compute resources for each allocation decision to those that provide a minimum probability of completing the given task on time. Unlike their Min–Min heuristic design, our SA, GA, GAPR, and GALS heuristics all leverage stochastic robustness information during resource allocation to maximize the probability that all tasks finish by the completion time constraint.

In [7], the authors demonstrate the use of a GA to minimize the expected system makespan of a resource allocation in a heterogeneous computing environment where task execution times are modeled as random variables. This research demonstrates the efficacy of a stochastic approach to resource scheduling, by showing that it can significantly reduce system makespan as compared to some well known scheduling heuristics that are based on a deterministic modeling of task execution times. The heuristics presented in that study were used in the stochastic domain to minimize the expected system makespan given a stochastic model of task execution times, i.e., the fitness metric in that approach was based on the first moment of random variables. The emphasis of our approach is on quantitatively comparing one resource allocation to another based on the stochastic robustness metric, i.e., the probability of satisfying a given makespan constraint. The success of the Genetic Algorithm applied to stochastic resource allocation in [7] was a motivating factor for our selection of a Genetic Algorithm in this study; however, our GA methodology differs significantly from that in [7], for both the robustness GA and the Makespan-GA.

8. Conclusions

This research presented several heuristics for directly maximizing the robustness of a resource allocation. The GA, GAPR, GALS, and SA techniques were shown to significantly outperform a simpler Two-Phase greedy heuristic. A comparison of these heuristics revealed the great potential for the GALS and GAPR heuristics to efficiently manage resources in distributed heterogeneous computing systems operating under uncertainty. This gain in performance was achieved through a unique application of local search and path relinking within the context of a Genetic Algorithm.

We also proposed a new method for extending the use of our robustness approach to environments where it is non-trivial to find resource allocations that provide a non-zero robustness value under tight deadline constraints. We applied this methodology to create the Makespan-GA heuristic for generating an initial population for use in our robustness-based GA heuristics that attempt to directly maximize the robustness of resource allocations subject to a constraint on the makespan of the solution. Our simulations clearly indicate the viability of this combined approach for maximizing the stochastic robustness of resource allocations.

Acknowledgments

The authors would like to thank Samee Khan, Abdulla Al-Qawasmeh, and Luis Briceño for their valuable comments. A preliminary version of portions of this material was presented at the Parallel and Distributed Processing Techniques and Applications conference [30]. This research was supported by the United States National Science Foundation (NSF) under grant numbers CNS-0615170, CNS-0905399, and CCF-1302693, and by the Colorado State University George T. Abell Endowment. This research used the CSU ISTeC Cray system supported by NSF grant CNS-0923386.

References

[1] T.D. Braun, H.J. Siegel, N. Beck, L. Bölöni, R.F. Freund, D. Hensgen, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, Journal of Parallel and Distributed Computing 61 (6) (2001) 810–837.
[2] X. Tang, K. Li, G. Liao, K. Fang, F. Wu, A stochastic scheduling algorithm for precedence constrained tasks on grid, Future Generation Computer Systems 27 (8) (2011) 1083–1091.
[3] E.G. Coffman (Ed.), Computer and Job-Shop Scheduling Theory, John Wiley & Sons, New York, NY, 1976.
[4] 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.
[5] S. Ali, T.D. Braun, H.J. Siegel, A.A. Maciejewski, N. Beck, L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, Characterizing Resource Allocation Heuristics for Heterogeneous Computing Systems, in: Advances in Computers, Elsevier, Amsterdam, The Netherlands, 2005, pp. 91–128.
[6] L. Bölöni, D. Marinescu, Robust scheduling of metaprograms, Journal of Scheduling 5 (5) (2002) 395–412.

[7] A. Dogan, F. Özgüner, Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems, Cluster Computing 2 (2004) 177–190.
[8] Y.C. Lee, A.Y. Zomaya, Rescheduling for reliable job completion with the support of clouds, Future Generation Computer Systems 26 (8) (2012) 1192–1199.
[9] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, Journal of Parallel and Distributed Computing 59 (2) (1999) 107–131.
[10] A.M. Mehta, J. Smith, H.J. Siegel, A.A. Maciejewski, A. Jayaseelan, B. Ye, Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness, Journal of Supercomputing 42 (1) (2007) 33–58.
[11] L. Wasserman, All of Statistics: A Concise Course in Statistical Inference, Springer Science+Business Media, New York, NY, 2005.
[12] Y.A. Li, J.K. Antonio, H.J. Siegel, M. Tan, D.W. Watson, Determining the execution time distribution for a data parallel program in a heterogeneous computing environment, Journal of Parallel and Distributed Computing 44 (1) (1997) 35–52.
[13] 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 (2008).
[14] A. Leon-Garcia, Probability & Random Processes for Electrical Engineering, Addison Wesley, Reading, MA, 1989.
[15] S. Ali, A.A. Maciejewski, H.J. Siegel, J.-K. Kim, Measuring the robustness of a resource allocation, IEEE Transactions on Parallel and Distributed Systems 15 (7) (2004) 630–641.
[16] S. Ali, A.A. Maciejewski, H.J. Siegel, J.-K. Kim, Static heuristics for robust resource allocation of continuously executing applications, Journal of Parallel and Distributed Computing 68 (8) (2008) 1070–1080.
[17] Z. Michalewicz, D.B. Fogel (Eds.), How to Solve It: Modern Heuristics, Springer-Verlag, New York, NY, 2000.
[18] D. Whitley, The Genitor algorithm and selective pressure: Why rank-based allocation of reproductive trials is best, in: Proceedings of the 3rd International Conference on Genetic Algorithms, Jun. 1989, pp. 116–121.
[19] E.K.P. Chong, S.H. Zak, An Introduction to Optimization, second ed., John Wiley, New York, NY, 2001.
[20] P. Sugavanam, H.J. Siegel, A.A. Maciejewski, M. Oltikar, A. Mehta, R. Pichel, A. Horiuchi, V. Shestak, M. Al-Otaibi, Y. Krishnamurthy, S. Ali, J. Zhang, M. Aydin, P. Lee, K. Guru, M. Raskey, A. Pippin, Robust static allocation of resources for independent tasks under makespan and dollar cost constraints, Journal of Parallel and Distributed Computing 67 (4) (2007) 400–416.
[21] C.R. Reeves, T. Yamada, Genetic algorithms, path relinking and the flowshop sequencing problem, Evolutionary Computation Journal 6 (1) (1998) 230–234.
[22] F. Junyent, V. Chandrasekar, D. McLaughlin, S. Frasier, E. Insanic, R. Ahmed, N. Bharadwaj, E. Knapp, L. Krnan, R. Tessler, Salient features of radar nodes of the first generation netrad system, in: Proceedings of the IEEE International Geoscience and Remote Sensing Symposium 2005, IGARSS '05, Jul. 2005, pp. 420–423.
[23] N. Bharadwaj, V. Chandrasekar, Waveform design for casa x-band radars, in: Proceedings of the 32nd Conference on Radar Meteorology of the American Meteorology Society, Oct. 2005.
[24] Standard performance evaluation corporation. Accessed Feb. 2006. (Online). Available: http://www.spec.org/, 2006.
[25] L.C. Canon, E. Jeannot, Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments, IEEE Transactions on Parallel and Distributed Systems 21 (4) (2010) 532–546.
[26] M. Lombardi, M. Milano, L. Benini, Robust scheduling of task graphs under execution time uncertainty, IEEE Transactions on Computers 62 (1) (2013) 98–111.
[27] G. Bernat, A. Colin, S.M. Peters, WCET analysis of probabilistic hard real-time systems, in: Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS '02), 2002.
[28] L. David, I. Puaut, Static determination of probabilistic execution times, in: Proceedings of the 16th Euromicro Conference on Real-Time Systems, ECRTS '04, Jun. 2004.
[29] J. Li, Z. Ming, M. Qiu, G. Quan, X. Qin, T. Chen, Resource allocation robustness in multi-core embedded systems with inaccurate information, Journal of Systems Architecture 57 (9) (2011) 840–849.
[30] J. Smith, H.J. Siegel, A.A. Maciejewski, Iterative techniques for maximizing stochastic robustness of a static resource allocation in periodic sensor driven clusters, in: Proceedings of the 2008 International Conference on Parallel and Distributed Processing Technologies and Applications, PDPTA 2008, Vol. 1, 2008, pp. 3–9.

Jay Smith received his Ph.D. in Electrical and Computer Engineering from Colorado State University in 2008. Jay is currently a researcher for Lagrange Systems. In addition to his position at Lagrange Systems, Jay is a research faculty member in the Electrical and Computer Engineering Department at Colorado State University. His research interests include high performance computing and resource management. He is a member of the IEEE and the ACM.

Anthony A. Maciejewski received the B.S.E.E., M.S, 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 B.S. degrees from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from 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 has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. Homepage: www.engr.colostate.edu/~hj.