Iterative Techniques for Maximizing Stochastic Robustness of a Static Resource Allocation in Periodic Sensor Driven Clusters

Jay Smith[1,2], Howard Jay Siegel[2,3], and Anthony A. Maciejewski[2]

[1]DigitalGlobe
Longmont, CO 80503 USA
Email: jtsmith@digitalglobe.com

Colorado State University
[2]Dept. of Electrical and Computer Engineering
[3]Dept. of Computer Science
Fort Collins, CO 80523–1373, USA
Email: {hj, aam}@colostate.edu

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 uncertainties 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 present two techniques for maximizing the probability that a given level of QoS is achieved. The performance results for our techniques are presented for a simulated environment that models a heterogeneous cluster-based radar data processing center.***

Keywords: robustness, heterogeneous computing, resource management, static resource allocation, distributed computing

1 Introduction

This paper investigates robust resource allocation for a large class of heterogeneous cluster (<u>HC</u>) systems that operate on *periodically updated* sensor data sets. Sensors (e.g., radar systems, sonar) in this environment periodically produce new data sets at a fixed period $\underline{\Lambda}$ (see Fig. 1. Because these sensors typically monitor the physical world, the characteristics of the data sets vary in a manner that impacts the execution times of the applications that must process them. Suppose that the HC system that processes these periodic data sets is composed of $\underline{M}$ compute nodes and that a collection of $\underline{N}$ independent applications must process each data set before the next data set arrives. Thus, the allocation of compute nodes to applications in this environment can be considered to be static, i.e., all of the applications that are to be executed are known in advance and are immediately available for execution. Because this is a heterogeneous computing system, the execution times for each of the N independent applications differs across the M compute nodes. Resource allocation in such computing environments has been shown in general to be an NP-hard problem (e.g., [8, 11]). Thus, designing techniques for resource management in such environments is an active area of research (e.g., [1, 5, 6, 10, 15]).

In this environment, a new data set arrives every Λ time units. Thus, the completion time of the last to finish application must be less than or equal to Λ to ensure that the system is ready to begin processing the next data set upon its arrival. However, unpredictable variance 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. This variance may cause the makespan of the resource allocation to exceed Λ, which is unacceptable in this environment. This complicates the process of resource allocation (i.e., assignment of applications to compute nodes). *Robust* design for such systems involves determining a resource allocation that can account for uncertainty in application execution times in a way that enables a probabilistic guarantee that all of the applications will complete within Λ time units.

This research was supported by the NSF under Grant CNS-0615170 and by the Colorado State University George T. Abell Endowment.

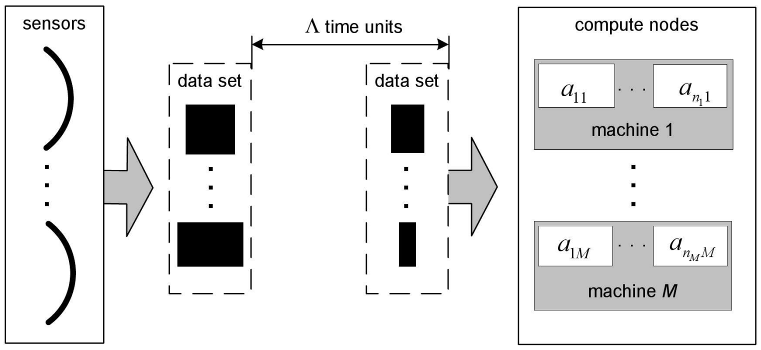

Figure 1: Major functional units and data flow for a class of systems that must periodically process data sets from a collection of sensors.

The major contribution of this paper is the design of resource allocation techniques that maximize the robustness of resource allocations subject to a constraint on the maximum allowable Λ. The notion of stochastic robustness was established in [17] based on a mathematical model of HC systems. That research presented three *greedy* approaches to resource allocation and three more sophisticated iterative algorithms. The emphasis of that research was on minimizing Λ subject to a constraint on the robustness of the resulting resource allocation. In this paper, we present two techniques for directly maximizing the robustness of a resource allocation subject to a constraint on the time interval required to process the N applications, i.e., Λ.

The remainder of this work is organized in the following manner. A brief introduction to the stochastic robustness framework is presented in Section 2 along with a brief introduction to using the stochastic robustness metric in a heuristic. Two iterative algorithms designed for this environment and one sample greedy heuristic are described in Section 3. The parameters of the simulation used to evaluate the heuristics are discussed in Section 4 along with the simulation results and a performance evaluation of the presented heuristics in Section 5. A sampling of relevant related work is presented in Section 6. Section 7 concludes the paper.

2 Stochastic Robustness

In this environment, we are concerned with allocating a set of N independent applications to M heterogeneous compute nodes so that the robustness of the resulting resource allocation is maximized subject to a constraint on the overall makespan (i.e., the completion time of the last to finish compute node). A robustness metric for this environment can be derived using the FePIA procedure first presented in [2].

For this system, the performance feature of interest is system makespan, denoted $\underline{\psi}$. 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., $\psi \leq \Lambda$.

Uncertainty in this system arises because the exact execution time for each application is not known in advance of its execution. We can 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 [20], denoted $\underline{\eta_{ij}}$. We assume that probability mass functions (pmfs) exist and are available that describe the possible execution times for each η_{ij}.

The finishing time of each compute node in this environment is calculated as the sum of the execution time random variables for each application assigned to that compute node [17]. Let $\underline{n_j}$ be the number of applications assigned to compute node j. The finishing time of compute node j, referred to as a local performance characteristic $\underline{\psi_j}$, can be expressed as follows:

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

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

$$\psi = \max\{\psi_1, \cdots, \psi_M\}. \tag{2}$$

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. The stochastic robustness metric, denoted θ is defined as the probability that the performance characteristic of the system is less than or equal to Λ, i.e., $\theta = \mathbb{P}[\psi \leq \Lambda]$. For a given resource allocation, the stochastic robustness metric measures the probability that the generated system performance will satisfy our robustness requirement. Clearly, unity is the most desirable stochastic robustness metric value, i.e., there is a zero probability that the system will violate the established robustness requirement.

Because there are no inter-task data transfers among the applications to be assigned, the random variables for the local performance characteristics $(\psi_1, \psi_2, \cdots, \psi_M)$ 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 feature is less than or equal to Λ. Mathematically, this is given as:

$$\theta = \prod_{\forall j} \Bigg(\mathbb{P}[\psi_j \leq \Lambda] \Bigg). \tag{3}$$

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 [13, 17].

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 a constraint on the maximum time period required to process all applications and would like to maximize the probability that all of the applications will complete by this deadline.

3 Heuristics

3.1 Two-Phase Greedy

The Two-Phase Greedy heuristic is based on the principles of the Min-Min algorithm (first presented in [11], and shown to perform well in many environments, e.g., [6, 16]). The heuristic requires N iterations to complete, resolving a single application to compute node assignment during each iteration.

The performance objective for our implementation was chosen to be the minimization of the system makespan subject to $\theta = 0.8$. In the first phase of each iteration, the heuristic determines the application to compute node assignment that minimizes the expected completion time for each of the applications left unmapped. In the second phase, the heuristic selects the application to compute node assignment from the first phase that increases the expected makespan the least.

An application of the Min-Min algorithm that directly maximizes robustness in this environment is ineffective because early in an allocation many of the applications have yet to be assigned. Thus, many of the competing application to compute node assignments will have identical robustness values, e.g., often unity.

3.2 Genetic Algorithm

The adopted genetic algorithm (GA) was motivated by the Genitor evolutionary heuristic first introduced in [21]. Our GA implementation models a complete resource allocation as a sequence of numbers, referred to as a chromosome, where the i^{th} entry in the sequence corresponds to the compute node assignment for the i^{th} application, denoted a_i[1]. The fitness of each chromosome is determined according to the robustness of the underlying resource allocation. That is, a higher robustness value θ indicates a more fit chromosome.

In our implementation, the 100 most fit chromosomes encountered, referred to as the population, are retained from each iteration for use in the next iteration. The initial members of the population were generated by applying the simple greedy sorting heuristic from [17], where the ordering of application assignments is chosen randomly.

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. Chromosome selection is performed using a linear bias function [21], where the rank of each chromosome is determined by its fitness. Each new chromosome generated is inserted into the population in sorted order according to its fitness value. For each chromosome inserted, the least fit chromosome in the population is removed, and the size of the population is held fixed. In an effort to maintain a diverse population, the insertion process prevents duplicate chromosomes from being inserted into the population.

The crossover operator was implemented using the two-point reduced surrogate procedure [21]. In two-point reduced surrogate crossover, the parent chromosomes are compared to identify the chromosome entries that differ between them. Crossover

[1]The ordering of tasks in a chromosome is not significant for this environment and can be considered arbitrary.

points are 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. Following crossover, a local search procedure, conceptually analogous to the steepest descent technique [7], is applied to each of the produced offspring prior to their insertion into the population.

The local search implemented in our GA is similar to the coarse refinement presented as part of the GIM heuristic in [19]. 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 machine, would increase the overall θ the most. This requires re-evaluating θ every time an application-compute node re-assignment is considered. Once an application-compute node pair has been identified, the chosen application is moved to its chosen compute node. Finally, the procedure repeats from the first step until there are no application moves from the lowest probability compute node that would improve θ. For this procedure, it is assumed that $\theta < 1$; otherwise, no improvements can be made through local search.

The final step within each iteration of our GA implementation applies a mutation operator. For each iteration of the GA, the mutation operator is applied to a small percentage of the population. In our implementation, we selected chromosomes from the population with a rate of 0.1, i.e., during each iteration we selected 10% of the population for mutation. During mutation, each application assignment of 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. 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 and local search is applied to the result prior to its insertion into the population.

We chose to limit the number of chromosome evaluations in our GA implementation to 4,000,000. For the simulation trials tested, this number of chromosome evaluations enabled the GA to find a resource allocation that provided a near unity robustness value.

3.3 Simulated Annealing

The Simulated Annealing (SA) algorithm—also known in the literature as Monte Carlo annealing or probabilistic hill-climbing [16]—is based on an analogy taken from thermodynamics. In SA, a randomly generated solution, structured as the chromosome from our GA, is iteratively modified and refined. Thus, SA in general can be considered as an iterative technique that operates with one possible solution (i.e., resource allocation) at a time.

To deviate from the current solution in an attempt to find a better one, SA repetitively applies the mutation operation in the same manner as in the GA, including the local search procedure. Once a new solution, denoted as S_{new}, is produced, a decision regarding the replacement of the previous solution, denoted S_{old}, with the new one has to be made. If the fitness of the new solution, denoted $\theta(S_{new})$, 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 [16]. The probability of replacement is based on a system temperature, denoted $\mathbb{T}$, 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

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

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 poorer solution is rejected is rather high, especially when the system temperature becomes relatively small.

After each mutation (described in Subsection 3.2), 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 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 was to 0.1.

For the simulation trials tested, our implementation of SA was terminated after 4,000,000 chromosome evaluations or a system temperature of 0.0001 was reached. Limiting the overall number of chromosome evaluations to 4,000,000 enabled a fair comparison with our GA result. For all simulation trials run, the system temperature never reached 0.0001, i.e., every trial was limited by the number of chromosome evaluations available.

4 Simulation Setup

For our simulations, the periodicity of data set arrivals Λ was assumed fixed at 540 time units. The value for the constraint on 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 resource allocation heuristics is to find resource allocations that have the highest probability to complete all of the applications within the given period Λ.

To evaluate the performance of the heuristics described in Section 3, the following approach was used to simulate a cluster-based radar system. The execution time distributions for 28 different types of possible radar ray processing algorithms on eight ($M = 8$) 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 [12]. These sample pmfs contain application execution times for 500 different radar data sets of varying complexity by the Pulse-Pair & Attenuation Correction algorithm [4] and by the Random Phase & Attenuation Correction algorithm [4]. 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 [18]. Combining the results available from the CFP2000 benchmarks with the sample pmfs produced by the two available applications provided a means for generating the 28×8 matrix of application execution times, where the kj^{th} element in the matrix corresponds to the application execution time pmf of a possible ray processing algorithm of type k on compute node m_j.

[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.

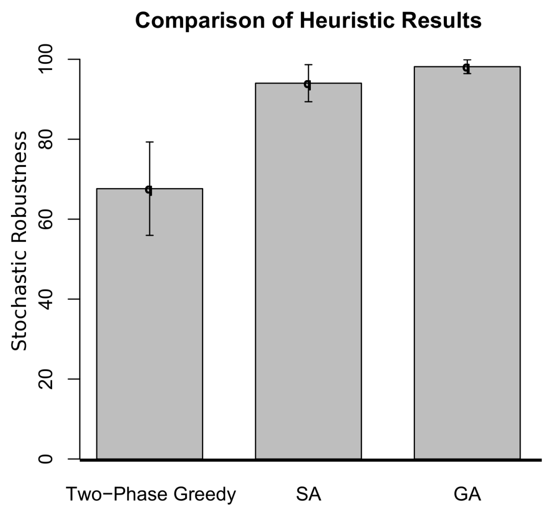

Figure 2: A comparison of the average robustness values attained through simulation for the GA, the SA, and the Two-Phase Greedy heuristics.

Each simulation trial consisted of a set of 128 applications ($N = 128$) to be assigned to the eight available heterogeneous computing systems. 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 Simulation Results

The results of the simulation trials are presented in Figure 2. The 50 simulation trials provide a good estimate of the mean and 95% confidence interval for each of the resource allocation techniques. Both the GA and SA were able to improve upon the robustness values achieved by the Two-Phase Greedy solution. For the 50 conducted trials, the Two-Phase Greedy heuristic produced an average robustness value of 67.63%. The SA results showed a mean robustness value of 94.02%. The GA performed better on average than either of the other two heuristics with an average robustness value of 98.15%. Figure 2 presents a simple plot of the results obtained from the 50 simulation trials.

As can be seen from the results, the GA appeared more capable of maximizing robustness in this environment given the chosen parameters. Although the GA presented a higher mean result than the SA, the confidence intervals of the two results overlap and therefore the difference in their performance is not statistically significant. Both the GA and SA heuristics significantly outperformed the Two-Phase Greedy approach. The comparable performance of the SA and GA approaches may suggest that the local search, common to both techniques, may have a significant impact on their attained results.

There was a significant difference between the SA and GA heuristics in terms of their worst-case completion time performance. The worst-case completion time performance for a resource allocation corresponds to the largest possible completion time impulse from the completion time distributions taken across all compute nodes in the allocation. This value is significant because it also corresponds to a robustness value of 1, i.e., the resource allocation is guaranteed to complete by this time. For our simulation study, the mean worst case completion time for the GA was found to be 543.25 time units with a 95% confidence interval of plus or minus 6.09 time units. The SA mean worst case completion time was found to be 564.84 time units with a 95% confidence interval of plus or minus 4.27 time units.

6 Related Work

A universal framework for defining the robustness of resource allocations in heterogeneous computing systems was addressed in [2]. 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 [17], the authors used the FePIA procedure to define a robustness metric for static stochastic resource allocation environments. The research in [17] 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 directly maximize the robustness of a resource allocation given a constraint on the allowed makespan.

In [5], the problem of robust resource allocation was addressed for scheduling Directed Acyclic Graphs (DAGs) in a heterogeneous computing environment. In [5], robustness was quantitatively measured as the "critical" (i.e., the smallest) slack among all components that comprise a given DAG. Although the authors focused on designing resource allocations that maximized robustness, their research was only demonstrated for a deterministic environment. Our robustness metric is based on stochastic information about the uncertainties.

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., [3, 9, 14]). In [3], 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 [10], 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 adapted to the stochastic domain and used 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. However, the success of the authors' Genetic Algorithm applied to stochastic resource allocation was a motivating factor for our selection of a Genetic Algorithm in this study.

7 Conclusions

This research presented two distinct techniques for directly maximizing the robustness of a resource allocation. Both the GA and SA techniques were shown to significantly outperform a simpler Two-Phase greedy heuristic. A comparison of the three heuristics revealed the great potential for the GA and SA algorithms to efficiently manage resources in distributed heterogeneous systems operating under uncertainty.

8 Acknowledgments

The authors would like to thank Samee Khan and Luis Briceño for their valuable comments.

References

[1] 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, and B. Yao, *Parallel, Distributed, and Pervasive Computing*, ser. Advances in Computers. Amsterdam, The Netherlands: Elsevier, 2005, pp. 91–128.

[2] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 7, pp. 630–641, Jul. 2004.

[3] G. Bernat, A. Colin, and S. M. Peters, "WCET analysis of probabilistic hard real-time systems," in *Proceedings of the 23^{rd} IEEE Real-Time Systems Symposium (RTSS '02)*, 2002.

[4] N. Bharadwaj and V. Chandrasekar, "Waveform design for casa x-band radars," in *Proceedings of the 32^{nd} Conference on Radar Meteorology of the American Meteorology Society*, Oct. 2005.

[5] L. Bölöni and D. Marinescu, "Robust scheduling of metaprograms," *Journal of Scheduling*, vol. 5, no. 5, pp. 395–412, Sep. 2002.

[6] 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, and 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*, vol. 61, no. 6, pp. 810–837, Jun. 2001.

[7] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization*, 2nd ed. New York, NY: John Wiley, 2001.

[8] E. G. Coffman, Ed., *Computer and Job-Shop Scheduling Theory*. New York, NY: John Wiley & Sons, 1976.

[9] L. David and I. Puaut, "Static determination of probabilistic execution times," in *Proceedings of the 16^{th} Euromicro Conference on Real-Time Systems (ECRTS '04)*, Jun. 2004.

[10] A. Dogan and F. Ozguner, "Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems," *Cluster Computing*, vol. 7, no. 2, pp. 177–190, Apr. 2004.

[11] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, Apr. 1977.

[12] F. Junyent, V. Chandrasekar, D. McLaughlin, S. Frasier, E. Insanic, R. Ahmed, N. Bharadwaj, E. Knapp, L. Krnan, and 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.

[13] A. Leon-Garcia, *Probability & Random Processes for Electrical Engineering*. Reading, MA: Addison Wesley, 1989.

[14] Y. A. Li, J. K. Antonio, H. J. Siegel, M. Tan, and D. W. Watson, "Determining the execution time distribution for a data parallel program in a heterogeneous computing environment," *Journal of Parallel and Distributed Computing*, vol. 44, no. 1, pp. 35–52, Jul. 1997.

[15] A. M. Mehta, J. Smith, H. J. Siegel, A. A. Maciejewski, A. Jayaseelan, and B. Ye, "Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness," *Journal of Supercomputing, Special Issue on Grid Technology*, vol. 42, no. 1, pp. 33–58, Oct. 2007.

[16] Z. Michalewicz and D. B. Fogel, Eds., *How to Solve It: Modern Heuristics*. New York, NY: Springer-Verlag, 2000.

[17] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *Journal of Parallel and Distributed Computing*, accepted to appear, 2008.

[18] (2006) Standard performance evaluation corporation. Accessed Feb. 2006. [Online]. Available: http://www.spec.org/

[19] 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, and A. Pippin, "Robust static allocation of resources for independent tasks under makespan and dollar cost constraints," *Journal of Parallel and Distributed Computing*, vol. 67, no. 4, pp. 400–416, Apr. 2007.

[20] L. Wasserman, *All of Statistics: A Concise Course in Statistical Inference*. New York, NY: Springer Science+Business Media, 2005.

[21] D. Whitley, "The genitor algorithm and selective pressure: Why rank-based allocation of reproductive trials is best," in *Proceedings of the 3^{rd} International Conference on Genetic Algorithms*, Jun. 1989, pp. 116–121.