The Robustness of Resource Allocations in Parallel and Distributed Computing Systems

Vladimir Shestak[1], Howard Jay Siegel[1,2],
Anthony A. Maciejewski[1], and Shoukat Ali[3]

[1] Department of Electrical & Computer Engineering
[2] Department of Computer Science,
Colorado State University, Fort Collins, CO 80523-1373, USA
{Shestak, HJ, AAM}@colostate.edu
[3] Department of Electrical & Computer Engineering,
University of Missouri-Rolla, Rolla, MO 65409-0040, USA
shoukat@umr.edu

Abstract. This corresponds to the material in the invited keynote presentation by H. J. Siegel, summarizing the research in [2, 23].

Resource allocation decisions in heterogeneous parallel and distributed computer systems and associated performance prediction are often based on estimated values of application and system parameters, whose actual values are uncertain and may be differ from the estimates. We have designed a model for deriving the degree of robustness of a resource allocation—the maximum amount of collective uncertainty in parameters within which a user-specified level of system performance can be guaranteed. The model will be presented, and we will demonstrate its ability to select the most robust resource allocation from among those that otherwise perform similarly (based on the primary performance criterion). We will show how the model can be used in off-line allocation heuristics to maximize the robustness of makespan against inaccuracies in estimates of application execution times in a cluster.

1 Introduction

This is an overview of the material to be discussed in the invited keynote presentation by H. J. Siegel; it summarizes our research in [2, 23].

This research focuses on the robustness of a resource allocation in parallel and distributed computing systems. What does robustness mean? Some dictionary definitions of robustness are: (a) strong and healthy, as in "a robust person" or "a robust mind," (b) sturdy or strongly formed, as in "a robust plastic," (c) suited to or requiring strength as in "a robust exercise" or "robust work," (d) firm in purpose or outlook as in "robust faith," (e) full-bodied as in "robust coffee," and (f) rough or rude as in "stories laden with robust humor." In the context of resource allocation in parallel and distributed computing systems, how is the concept of robustness defined?

The allocation of resources to computational applications in heterogeneous parallel and distributed computer systems should maximize some system performance measure. Allocation decisions and associated performance prediction are often based

W. Grass et al. (Eds.): ARCS 2006, LNCS 3894, pp. 17 – 30, 2006.

on estimated values of application parameters, whose actual values may differ; for example, the estimates may represent only average values, or the models used to generate the estimates may have limited accuracy. Furthermore, parallel and distributed systems may operate in an environment where certain system performance features degrade due to unpredictable circumstances, such as sudden machine failures, higher than expected system load, or inaccuracies in the estimation of system parameters (e.g., [1, 3, 4, 5, 8, 11, 13, 14, 16, 17, 22]). Thus, an important research problem is the development of resource management strategies that can guarantee a particular system performance given bounds on such uncertainties. A resource allocation is defined to be *robust with respect to specified system performance features against perturbations (uncertainties) in specified system parameters* if degradation in these features is constrained when limited perturbations occur. An important question then arises: given a resource allocation, what extent of departure from the assumed circumstances will cause a performance feature to be unacceptably degraded? That is, how robust is the system?

Any claim of robustness for a given system must answer these three questions: (a) what behavior of the system makes it robust? (b) what uncertainties is the system robust against? (c) quantitatively, exactly how robust is the system? To address these questions, we have designed a model for deriving the degree of robustness of a resource allocation—the maximum amount of collective uncertainty in system parameters within which a user-specified level of system performance can be guaranteed. The model will be presented and we will demonstrate its ability to select the most robust resource allocation from among those that otherwise perform similarly (based on the primary performance criterion). The model's use in static (off-line) allocation heuristics also will be demonstrated. In particular, we will show how to maximize the robustness of makespan against inaccuracies in estimates of application execution times in a heterogeneous cluster. In general, this work is applicable to different types of computing and communication environments, including parallel, distributed, cluster, grid, Internet, embedded, and wireless.

Section 2 describes the FePIA procedure for deriving a robustness metric for an arbitrary system. Derivation of this metric for a given allocation of independent applications in a heterogeneous distributed system is presented in Section 3, with an experiment that highlights the usefulness of the robustness metric. Section 4 discusses heuristics developed to generate static resource allocations of independent applications in distributed systems such that the robustness of the produced resource allocations is maximized. Section 5 extends the work presented in Section 4 for distributed systems where the dollar cost for processors is a constraint. Some future work is described briefly in Section 6.

2 Generalized Robustness Metric

This section presents a general procedure, called *FePIA,* for deriving a general robustness metric for any desired computing environment [2]. The name for the above procedure stands for identifying the performance *fe*atures, the *p*erturbation parameters, the *i*mpact of perturbation parameters on performance features, and the *a*nalysis to determine the robustness. A specific example illustrating the application of

the FePIA procedure to a sample system is given in the next section. Each step of the FePIA procedure is now described, summarized from [2].

1) Describe quantitatively the requirement that makes the system robust (question (a) in Section 1). Based on this *robustness requirement*, determine the QoS performance features that should be limited in variation to ensure that the robustness requirement is met. Identify the acceptable variation for these feature values as a result of uncertainties in system parameters. Consider an example where (a) the QoS performance feature is *makespan* (the total time it takes to complete the execution of a set of applications) for a given resource allocation, (b) the acceptable variation is up to a 20% increase of the makespan that was predicted for the given resource allocation using estimated execution times of applications on the machines they are assigned, and (c) the uncertainties in system parameters are inaccuracies in the estimates of these execution times.

2) Identify the uncertainties to be considered whose values may impact the QoS performance features selected in step 1 (question (b) in Section 1). These are called the *perturbation parameters*, and the performance features are required to be robust with respect to these perturbation parameters. For the makespan example above, the resource allocation (and its associated predicted makespan) was based on the estimated application execution times. It is desired that the makespan be robust (stay within 120% of its estimated value) with respect to uncertainties in these estimated execution times.

3) Identify the impact of the perturbation parameters in step 2 on the system performance features in step 1. For the makespan example, the sum of the *actual* execution times for all of the applications assigned to a given machine is the time when that machine completes its applications. Note that 1(b) states that the actual time each machine finishes its applications must be within the acceptable variation.

4) The last step is to determine the smallest collective variation in the values of perturbation parameters identified in step 2 that will cause any of the performance features identified in step 1 to violate its acceptable variation. Step 4 is done for a given, specific resource allocation. This will be the *degree of robustness* of the given resource allocation (question (c) in Section 1). For the makespan example, this will be some quantification of the total amount of inaccuracy in the execution times estimates allowable before the actual makespan exceeds 120% of its estimated value.

3 Robustness Metric Example

3.1 Derivation of Robustness

In this section summarized from [2], the robustness metric is derived for a system that assigns a set of independent applications to a distributed set of machines. In this system, it is required that the makespan be robust against errors in application execution time estimates. Specifically, the actual makespan under the perturbed execution times must be no more than a certain factor (> 1) times the predicted makespan calculated using the estimated execution times.

A brief description of the system model is now given. The applications are assumed to be independent, i.e., no communications between the applications are

needed. The set A of applications is to be assigned to the set Ω of machines so as to minimize the makespan. Each machine executes a single application at a time (i.e., no multi-tasking). Let C_{ij} be the *estimated time to compute* (*ETC*) for application a_i on machine m_j. It is assumed that C_{ij} values are known *a priori* for all *i*, *j*. This assumption is commonly made (e.g., [15]). Approaches for doing this estimation are discussed in [10]. In addition, let F_j be the time at which m_j finishes executing all of the applications assigned to it.

It is assumed that unknown inaccuracies in the ETC values are expected (e.g., a task's actual exact execution time may be data dependent). Hence, it is required that the mapping, denoted by μ, and based on the ETC values, be robust against them. More specifically, it is required that, for a given resource allocation, its actual makespan value M (calculated using the actual application computation times (not the ETC values)) may be no more than τ (> 1) times its *predicted value*, denoted by M^{pred}. The predicted value of the makespan is the value calculated assuming the estimated ETC values. Following step 1 of the FePIA procedure in Section 2, the system performance features that should be limited in variation to ensure the makespan robustness are the finish times of the machines. That is, $\{F_j \le \tau M^{pred} \text{ for } 1 \le j \le |\Omega|\}$.

According to step 2 of the FePIA procedure, the perturbation parameter needs to be defined. Let C_i^{est} be the ETC value for application a_i on the machine where it is assigned. Let C_i be the actual computation time value. Let $\boldsymbol{C}$ be the vector of the C_i values, and $\boldsymbol{C}^{\boldsymbol{est}}$ be the vector of the C_i^{est} values. The vector $\boldsymbol{C}$ is the perturbation parameter for this analysis.

In accordance with step 3 of the FePIA procedure, F_j has to be expressed as a function of $\boldsymbol{C}$. To that end,

$$F_j(\boldsymbol{C}) = \sum_{i:\, a_i \text{ is assigned to } m_j} C_i. \tag{1}$$

Following step 4 of the FePIA procedure, the set of boundary relationships corresponding to the set of performance features is given by $\{F_j(\boldsymbol{C}) = \tau M^{pred} \text{ for } 1 \le j \le |\Omega|\}$.

The *robustness radius*, denoted by $r_\mu(F_j, \boldsymbol{C})$, for machine *j* provides the largest Euclidian distance, i.e., l_2-norm, at which variable $\boldsymbol{C}$ can change in any direction from the assumed point $\boldsymbol{C}^{\boldsymbol{est}}$ without the finish time $F_j(\boldsymbol{C})$ exceeding the tolerable variation:

$$r_\mu(F_j, \boldsymbol{C}) = \min_{\boldsymbol{C}:\, F_j(\boldsymbol{C}) = \tau M^{pred}} \left\| \boldsymbol{C} - \boldsymbol{C}^{\boldsymbol{est}} \right\|_2. \tag{2}$$

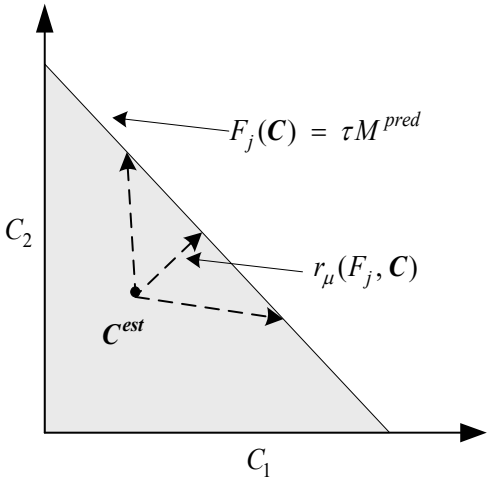

Fig. 1. Some possible directions of increase of the perturbation parameter C. The set of boundary points is given by $F_j(C^{est}) = \tau M^{pred}$. The robustness radius $r_\mu(F_j, C)$ corresponds to the smallest increase that can reach the boundary. The shaded region represents the area of robust operation.

That is, if the Euclidean distance between any vector of the actual execution times and the vector of the estimated execution times is no larger than $r_\mu(F_j, \boldsymbol{C})$, then the finish time of machine m_j will be at most τ times the estimated makespan value.

For example, assume only applications a_1 and a_2 have been assigned to machine j (depicted in Fig. 1), and $\boldsymbol{C}$ has two components C_1 and C_2 that correspond to execution times of a_1 and a_2 on machine j, respectively. The term $F_j(\boldsymbol{C}^{est})$ is a finish time for machine j computed based on the ETC values of applications a_1 and a_2. The boundary line is determined by $F_j(\boldsymbol{C}) = \tau M^{pred}$. Note that the right hand side in Equation 2 can be interpreted as the perpendicular distance from the point $\boldsymbol{C}^{est}$ to the hyperplane described by the equation $F_j(\boldsymbol{C}) = \tau M^{pred}$. Using the point-to-plane distance formula [21], Equation 2 reduces to

$$r_\mu(F_j, \boldsymbol{C}) = \frac{\tau M^{pred} - F_j(\boldsymbol{C}^{est})}{\sqrt{\text{number of applications assigned to } m_j}}. \tag{3}$$

The *robustness metric*, denoted by $\underline{\rho_\mu}$, is given as

$$\rho_\mu = \min_{1 \le j \le |\Omega|} \{r_\mu(F_j, \boldsymbol{C})\}. \tag{4}$$

That is, if the Euclidean distance between any vector of the actual execution times and the vector of the estimated execution times is no larger than ρ_μ, then the actual makespan will be at most τ times the predicted makespan value.

3.2 Utility of Robustness

The experiments in this subsection seek to establish the utility of the robustness metric. The experiments were performed for a system with five machines and 20 applications. A total of 1000 resource allocations were generated by assigning a randomly chosen machine to each application (see [2] for details).

The resource allocations were evaluated for robustness, makespan, and *load balance index* (defined as the ratio of the finish time of the machine that finishes first to the makespan). The larger the value of the load balance index, the more balanced the load (the largest value being 1). The tolerance, τ, was set to 120%. In this context, a robustness metric value of x for a given resource allocation means that the resource allocation can endure any combination of ETC errors without the makespan increasing beyond 1.2 times its estimated value as long as the Euclidean distance of the errors is no larger than x seconds.

Fig. 2(a) shows the "normalized robustness" of a resource allocation against its makespan. The *normalized robustness* equals the robustness metric value divided by the predicted makespan. A graph for the normalized robustness against the load balance index is shown in Fig. 2(b).

There are large differences in the robustness of some resource allocations that have very similar values of makespan. Thus, when selecting a resource allocation with low makespan, the robustness calculation allows one to select an allocation that also provides high robustness. Fig. 2(b) shows that load balancing does not provide an accurate measure of robustness. These observations highlight the fact that the information given by the robustness metric could not be obtained from the makespan and load balance performance measures.

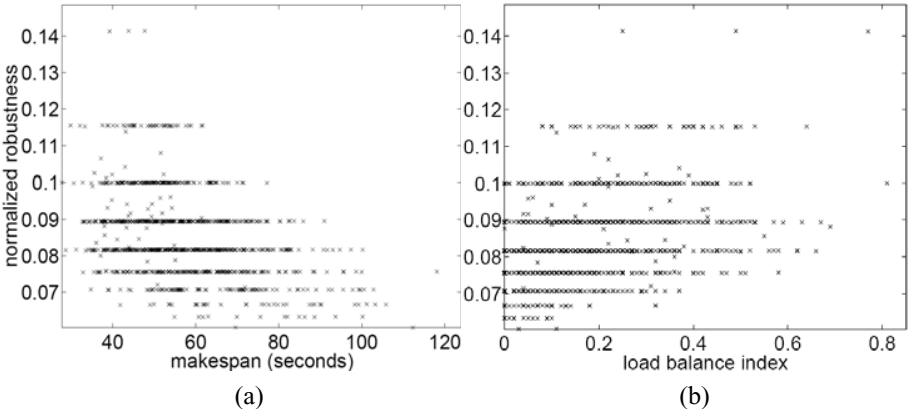

Fig. 2. Normalized robustness against (*a*) makespan and (*b*) load balance index for 1000 randomly generated resource allocations

4 Robust Resource Allocation Under a Makespan Constraint

4.1 Problem Statement

This section summarizes a part of the research described in [23]. An important research problem is how to determine a *mapping* (*resource allocation*) so as to maximize the robustness of desired system features against perturbations in system parameters. The general problem of optimally mapping applications to machines has been shown to be NP-complete [7, 9, 12]. Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research (e.g., [6, 18, 19, 20]). *Static* mapping is performed when the applications are mapped in an off-line planning phase such as in a production environment. Static mapping techniques take a set of applications, a set of machines, and generate a mapping. These heuristics determine a mapping off-line, and must use estimated values of application computation times.

As described in the previous section, the allocation of independent applications in parallel systems is considered robust if the actual makespan under the perturbed conditions does not exceed the required time constraint. The goal of this study was to find a static mapping of all applications to machines so that the robustness of the mapping is maximized; i.e., to maximize the collective allowable error in execution time estimation for the applications that can occur without the actual makespan exceeding the constraint. Mathematically, this problem can be stated as finding a mapping of $|A|$ applications to $|\Omega|$ machines such that the actual makespan is within the *absolute time constraint* $\underline{\alpha}$ while maximizing ρ_μ, given by (4). Equation (3) is restated in this study as

$$r_\mu(F_j, \boldsymbol{C}) = \frac{\alpha - F_j(\boldsymbol{C}^{est})}{\sqrt{\text{number of applications asiigned to } m_j}}. \quad (5)$$

A distributed system with eight machines and 1024 independent applications was simulated in this study. Two different cases of ETC heterogeneities were used in this research, the high application and high machine heterogeneity (*high-high*) case and the low application and low machine heterogeneity (*low-low*) case (see [23] for details about the simulation setup). The value of the time constraint α of 5000 seconds was chosen so that it presents a feasible mapping problem for the heuristics to solve. A total of 100 trials (50 trails for each of the cases) were performed, where each trial corresponded to a different ETC of C_{ij} values matrix. The wall clock time for each of the heuristics to determine a mapping was arbitrarily required to be less than or equal to 60 minutes to establish a basis for comparison.

Six static mapping schemes were developed in this study: Max-Max, Greedy Iterative Maximization (GIM), Sum Iterative Maximization (SIM), Genitor, Memetic Algorithm (MA), and Hereboy Evolutionary Algorithm (Hereboy). Two are described here.

4.2 Max-Max

The Max-Max heuristic is based on the Min-Min (greedy) concept in [12]. In step 2 of the Max-Max heuristic, to find the *fitness* function for assigning a given application *i* to a given machine *j*, the robustness radius of machine *j* given by equation (5) is evaluated based on the applications already assigned to machine *j* and the possible assignment of application *i* to machine *j*.

The Max-Max heuristic can be summarized by the following procedure:

1) An application list is generated that includes all the unmapped applications.
2) For each application in the application list, the machine that gives the application its maximum fitness value (first "Max") is determined (ignoring other unmapped applications).
3) Among all the application/machine pairs found in the above step, the pair that gives the maximum fitness value (second "Max") is chosen.
4) The application found in step 3 is then removed from the application list and is mapped to its paired machine.
5) Repeat steps 2 to 4 until all the applications are mapped.

4.3 Genitor

This heuristic is a general optimization technique that is a variation of the genetic algorithm approach. It manipulates a set of possible solutions. The framework used here is based on the Genitor approach used in [24]. In our study, each *chromosome* represents a possible complete mapping of applications to machines. Specifically, the chromosome is a vector of length $|A|$. The i^{th} element of the vector is the number of the machine to which application *i* is assigned. A fixed population of 200 chromosomes is used. The population includes one chromosome (seed) that is the Max-Max solution based on robustness (described above) and the rest of the chromosomes are generated by randomly assigning applications to machines. The entire population is sorted (ranked) based on their robustness metric values given by (4). Chromosomes that do not meet the makespan constraint are allowed to be included in the population. The ranking is constructed so that all chromosomes that meet the constraint are listed first, ordered by their robustness metric value (highest first). The chromosomes that do not meet the makespan constraint are then listed, again ordered by their robustness metric value (which will be negative).

Next, a special linear bias function [24] is used to select two chromosomes to act as parents. These two parents perform a crossover operation, and two new offspring are generated. For the pair of the selected parent chromosomes a random cut-off point is generated that divides the chromosomes into top and bottom parts. For the parts of both chromosomes from that point to the end of each chromosome, crossover exchanges machine assignments between corresponding applications producing two new offspring. The offspring are then inserted in the population in ranked order, and the two lowest ranked chromosomes are dropped.

After each crossover, the linear bias function is applied again to select a chromosome for mutation. A mutation operator generates a single offspring by perturbing the original chromosome. A random application is chosen from the

chromosome and the mutation operator randomly reassigns it to a new machine. The resultant offspring is considered for inclusion in the population in the same fashion as for an offspring generated by crossover.

This completes one iteration of the Genitor. The heuristic stops after 250,000 total iterations.

4.4 Experimental Results

The simulation results are shown in Fig. 3. All the heuristics run for 50 different high-high and 50 different low-low scenarios, and the average values and 95% confidence intervals are plotted. The Genitor performed among the best, comparable to GIM, SIM, and MA (i.e., overlapping confidence intervals). A discussion of all the results is in [23].

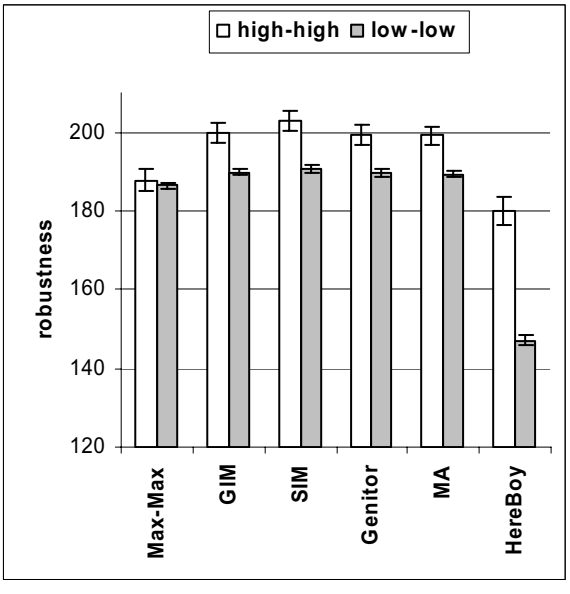

Fig. 3. Simulation results for robustness for a given fixed set of machines

5 Mapping Under Makespan and Dollar Cost Constraints

5.1 Problem Statement

This section summarizes another part of [23], which extends the idea in Section 4. The research environment here differs from Section 4 with the addition of a *cost constraint* for the machines and choosing a subset of all the available machines to be used. Thus, problem addressed here is how to select (purchase) a fixed set of machines, within a given dollar cost constraint to comprise a cluster system. It is assumed that this fixed system will be used in a production environment to regularly

execute the set A of applications with known estimated computational characteristics. The machines to be purchased for the set are to be selected from five different classes of machines, where each class consists of homogeneous machines. The machines of different classes differ in dollar costs depending upon their application execution speed. The dollar cost of machines within a class is the same. Machines in class i are assumed to be faster than machines of class i+1 for all applications, for $1 \le i \le 4$. Correspondingly, class i machines cost more that class i+1 machines.

In this study, one must: (1) select a subset of machines so that the cost constraint for the machines is satisfied, and (2) find a static mapping of all applications to the subset. Sub-problems 1 and 2 must be done in a way so that the robustness of the mapping is maximized. For sub-problem 2, the machine assignment heuristics described in the previous section are used as components of the heuristics developed in this research.

A method used to generate 100 high application and low machine heterogeneity (*high-low*) ETC matrices for 1024 independent applications was identical to that used in the previous work (see the details of the simulation setup in [23]). Experiments with simple greedy heuristics were used to decide the value of the cost constraint to be 34,800 dollars and the time constraint α to be 12,000 seconds. Choosing different values for any of the above parameters will not affect the general approach of the heuristics used in this research. The wall clock time for the mapper itself was set as in Section 4.

Six static mapping schemes were developed in this research: Negative Impact Greedy Iterative Maximization (NI-GIM), Parition/Merge Greedy Iterative Maximization (P/M-GIM), Cost and Robustness Sum Iterative Maximization (CR-SIM), Selection Genitor (S-Genitor), Max-Max Memetic Algorithm (MMMA), and Max-Max Hereboy Evolutionary Algorithm (MM-Hereboy). The S-Genitor heuristic is described next.

5.2 Selection Genitor

The S-Genitor heuristic developed in this work consists of two phases. For phase 1, a chromosome is a vector of length five, where the i^{th} element is the number of machines used in i^{th} class. The phase 1 of S-Genitor operates on a fixed population of 100 chromosomes. The entire population is generated randomly such that the cost constraint is met. To evaluate each chromosome, a mapping was produced using the Max-Max heuristic based on robustness (described in Subsection 4.2). The entire population is sorted in descending order based on the robustness metric.

In the crossover step, for the pair of the parent chromosomes selected by applying the linear bias function, a random cut-off point is generated that divides the chromosomes into top and bottom parts. A new chromosome is formed using the top of one and bottom of another. An offspring is inserted in the population after evaluation only if the cost constraint is satisfied (the worst chromosomes of the population are discarded to maintain a population of only 100).

After each crossover, the linear bias function is applied again to select a chromosome for mutation. A mutation operator generates a single offspring by perturbing the original chromosome. Two random classes are chosen for the chromosome and the mutation operator increments the number of machines of the

first chosen class by one and decrements the number of machines of the other by one. If the chromosome violates the cost constraint it is discarded. Otherwise, the resultant offspring is considered for inclusion in the population in the same fashion as for an offspring generated by crossover.

This completes one iteration of phase 1 of S-Genitor. The heuristic stops when the criterion of 500 total iterations is met. The best machine combination found from phase 1 is used in phase 2, which derives a mapping using this combination of machines to maximize robustness based on the Genitor implementation described in Section 4 (a total of 100,000 iterations is used here to stop phase 2 of S-Genitor).

5.3 Experimental Results

The simulation results are shown in Fig. 4. All the heuristics run for 100 different scenarios and the average values and 95% confidence intervals are plotted. The S-Genitor is among and the best heuristics, comparable in performance with the P/M-GIM heuristic. Both of these heuristics, on average, had all of the available machines from Class 4 and Class 5. A discussion of all the results is in [23].

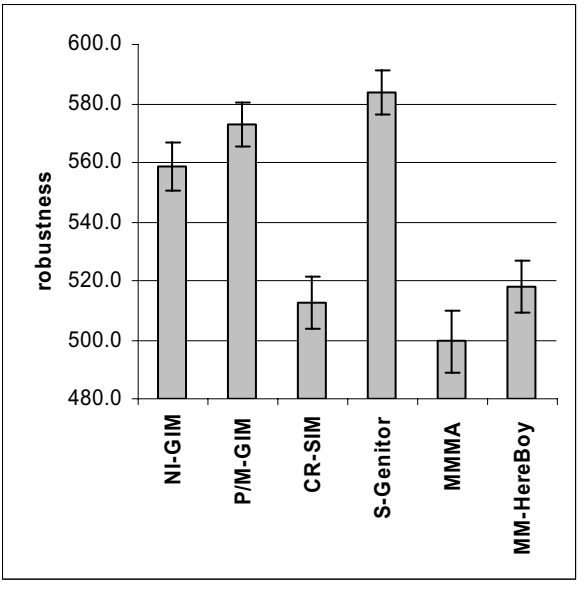

Fig. 4. Simulation results for robustness. Machine sets were determined heuristically.

6 Future Work

There are many directions in which the robustness research presented in the paper can be extended. Examples include the following.

1) Deriving the boundary surfaces for different problem domains.
2) Incorporating multiple types of perturbation parameters (e.g., uncertainties in input sensor loads and uncertainties in estimated execution times). Challenges here are how to define the collective impact to find each robust radius and how to state the combined bound on multiple perturbation parameters to maintain the promised performance.
3) Incorporating probabilistic information about uncertainties. In this case, a perturbation parameter can be represented as a vector of random variables. Then, one might have probabilistic information about random variables in the vector (e.g., probability density functions) or probabilistic information describing the relationship between different random variables in the vector or between different vectors (e.g., a set of correlation coefficients).
4) Determining when to use Euclidean distance versus other distance measures when calculating the collective impact of changes in the perturbation parameter elements.

7 Summary

Any claim of robustness for a given system must answer three questions: (a) what behavior of the system makes it robust? (b) what uncertainties is the system robust against? (c) quantitatively, exactly how robust is the system? This paper, which corresponds to H. J. Siegel's keynote presentation, summarizes the material from two papers related to robustness. A metric for the robustness of a resource allocation with respect to desired system performance features against perturbations in system and environmental conditions, and the experiments conducted to illustrate the utility of the robustness metric, are summarized from [2]. Heuristics developed to generate mappings of independent applications in distributed systems such that the robustness of the produced mappings is maximized are summarized from [23]. Finally, heuristics for (1) selecting a set of machines and (2) mapping applications to the set of machines, both to maximize robustness, also are summarized from [23].

Acknowledgment

An earlier version of portions of this manuscript appeared as an invited keynote paper in *8th International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN 2005)*.

References

1. S. Ali, J.K. Kim, Y. Yu, S. B. Gundala, S. Gertphol, H. J. Siegel, A. A. Maciejewski, and V. Prasanna, "Utilization-based techniques for statically mapping heterogeneous applications onto the HiPer-D heterogeneous computing system," *Parallel and Distributed Computing Practices*, Vol. 5, No. 4, Dec. 2002.

2. S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *IEEE Transactions on Parallel Systems*, Vol. 15, No. 7, July 2004, pp. 630–641.
3. S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Robust resource allocation for distributed computing systems," *2004 International Conference on Parallel Processing (ICPP'04)*, Aug. 2004, pp. 178–185.
4. P. M. Berry, "Uncertainty in scheduling: probability, problem reduction, abstractions and the user," *IEE Computing and Control Division Colloquium Advanced Software Technologies for Scheduling*, Digest No. 1993/163, Apr. 1993.
5. L. Boloni and D. C. Marinescu, "Robust scheduling of metaprograms," *Journal of Scheduling*, Vol. 5, No. 5, Sept. 2002, pp. 395–412,
6. T. D. Braun, H. J. Siegel, N. Beck, L. L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent applications onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, Vol. 61, No. 6, June 2001, pp. 810–837.
7. E. G. Coffman, Jr. ed., *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, NY, 1976.
8. R. L. Daniels and J. E. Carrillo, "β-Robust scheduling for single-machine systems with uncertain processing times," *IIE Transactions*, Vol. 29, No. 11, Nov. 1997, pp. 977–985.
9. D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.
10. A. Ghafoor and J. Yang, "A distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78–86.
11. S. D. Gribble, "Robustness in complex systems," *8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001, pp. 21–26.
12. O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent applications on non-identical processors," *Journal of ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280–289.
13. M. Jensen, "Improving robustness and flexibility of tardiness and total flowtime job shops using robustness measures," *Journal of Applied Soft Computing*, Vol. 1, No. 1, June 2001, pp. 35–52.
14. E. Jen, "Stable or robust? What is the difference?" *Complexity*, Vol. 8, No. 3, June 2003.
15. M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, Vol. 6, No. 3, July. 1998, pp. 42–51.
16. V. J. Leon, S. D. Wu, and R. H. Storer, "Robustness measures and robust scheduling for job shops," *IEE Tranactions*, Vol. 26, No. 5, Sept. 1994, pp. 32–43.
17. M. Sevaux and K. Sorensen, "Genetic algorithm for robust schedules," *8th International Workshop on Project Management and Scheduling (PMS 2002)*, Apr. 2002, pp. 330–333.
18. V. Shestak, E. K. P. Chong, A. A. Maciejewski, H. J. Siegel, L. Benmohamed, I. J. Wang, and R. Daley, "Resource allocation for periodic applications in a shipboard environment," *14th Heterogeneous Computing Workshop (HCW 2005)* in proceedings of 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), Apr. 2005, pp. 122–127.
19. S. Shivle, P. Sugavanam, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, A. Kutruff, P. Penumarthy, P. Pichumani, P. Satyasekaran, D. Sendek, J. Sousa, J. Sridharan, and J. Velazco, "Mapping of subtasks with multiple versions on an *ad hoc* grid environment," *Parallel Computing,* Special Issue on Heterogeneous Computing, Vol. 31, No. 7, July 2005, pp. 671–690.

20. S. Shivle, H. J. Siegel, A. A. Maciejewski, P. Sugavanam,T. Banka, R. Castain, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, and J. Velazco, "Static allocation of resources to communicating subtasks in a heterogeneous *ad hoc* grid environment," *Journal of Parallel and Distributed Computing*, accepted, to appear.
21. G. F. Simmons, *Calculus with Analytic Geometry, Second Edition,* McGraw-Hill, New York, NY, 1995.
22. Y. N. Sotskov, V. S. Tanaev, and F. Werner, "Stability radius of an optimal schedule: A survey and recent developments," *Industrial Applications of Combinatorial Optimization*, Vol. 16, 1998, pp. 72–108.
23. 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*, accepted, to appear in 2006.
24. D. Whitley, "The GENITOR algorithm and selective pressure: Why rank based allocation of reproductive trials is best," *3rd International Conference on Genetic Algorithms*, June 1989, pp. 116–121.