Robust Resource Allocations in Parallel Computing Systems: Model and Heuristics

Vladimir Shestak[1],
Howard Jay Siegel[1,2],
and Anthony A. Maciejewski[1]
Colorado State University
[1]Department of Electrical & Computer Engineering
[2]Department of Computer Science
Fort Collins, CO 80523-1373
Email:{shestak, hj, aam}@colostate.edu

Shoukat Ali
University of Missouri-Rolla
Department of Electrical and Computer Engineering
Rolla, MO 65409–0040
Email: shoukat@umr.edu

Abstract

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, 16, and 17].

The resources in parallel computer systems (including heterogeneous clusters) should be allocated to the computational applications in a way that maximizes some system performance measure. However, allocation decisions and associated performance prediction are often based on estimated values of application and system parameters. The actual values of these parameters may differ from the estimates; for example, the estimates may represent only average values, the models used to generate the estimates may have limited accuracy, and there may be changes in the environment. Thus, an important research problem is the development of resource management strategies that can guarantee a particular system performance given such uncertainties. To address this problem, 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 (QoS) 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 allocation heuristics also will be demonstrated. This model is applicable to different types of computing and communication environments, including parallel, distributed, cluster, grid, Internet, embedded, and wireless.

This research was supported by the Colorado State University Center for Robustness in Computer Systems (funded by the Colorado Commission on Higher Education Technology Advancement Group through the Colorado Institute of Technology), and by the Colorado State University George T. Abell Endowment.

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, 16, and 17].

In the context of resource allocation in parallel computing systems, including heterogeneous clusters, how is the concept of robustness defined? Parallel 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., [4, 5, 7, 8, 10, 11, 12, 13, 15]). 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 limited when the 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?

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 parallel system is presented in Section 3, with an experiment that highlights the usefulness of the robustness metric. Section 4 discusses heuristics developed to generate mappings of independent applications in parallel systems such that the robustness of the produced mappings is maximized. Section 5 extends the work presented in Section 4 for parallel systems where the dollar cost for processors is a constraint. Some future work is outlined 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 sample systems 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 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 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 $\underline{A}$ of applications is to be assigned to a set $\underline{\Omega}$ of machines so as to minimize the makespan. Each machine executes a single application at a time (i.e., no multi-tasking). Let $\underline{C_{ij}}$ be the *estimated time to compute* (*ETC*) for application $\underline{a_i}$ on machine $\underline{m_j}$. It is assumed that C_{ij} values are known for all *i*, *j*, and a resource allocation $\underline{\mu}$ is determined based on the ETC values. In addition, let $\underline{F_j}$ be the time at which m_j *finishes* executing all of the applications assigned to it.

Assume that unknown inaccuracies in the ETC values are expected, requiring that the resource allocation μ be robust against them. More specifically, it is required that, for a given resource allocation, its actual makespan value $\underline{M}$ (calculated using the actual application computation times (not the ETC values)) may be no more than $\underline{\tau}$ times its *predicted value*, $\underline{M^{pred}}$. 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 finishing 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 $\underline{C_i^{est}}$ be the ETC value for application a_i on the machine where it is assigned. Let $\underline{C_i}$ be the actual computation time value. Let $\underline{\boldsymbol{C}}$ be the vector of the C_i values, and $\underline{\boldsymbol{C}^{est}}$ 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* $\underline{r_\mu(F_j, \boldsymbol{C})}$ 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 $\boldsymbol{C}^{est}$ without the finishing time $F_j(\boldsymbol{C})$ exceeding the tolerable variation:

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

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 finishing time of machine m_j will be at most τ times the estimated makespan value.

Assume only applications a_1 and a_2 have been assigned to machine j, depicted in Figure 1, $\boldsymbol{C}$ 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 finishing time for machine j computed based on ETC values of applications a_1 and a_2. The boundary line is determined by to $F_j(\boldsymbol{C}) = \tau M^{pred}$. 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 [14], Equation 2 reduces to

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

The *robustness metric,* $\underline{\rho_\mu}$, 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

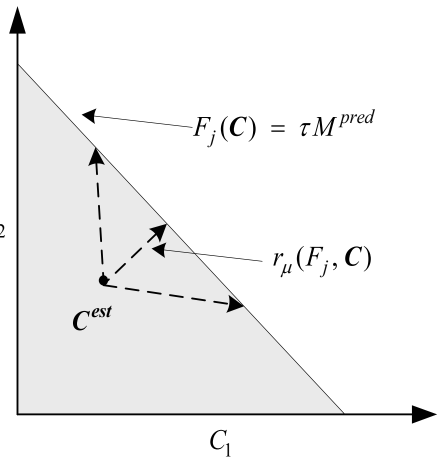

$F_j(\boldsymbol{C}) = \tau M^{pred}$
C_2
$r_\mu(F_j, \boldsymbol{C})$
$\boldsymbol{C}^{est}$
C_1

Figure 1: Some possible directions of increase of the perturbation parameter C. Robustness radius $r_\mu(F_j, C)$ corresponds to the smallest increase. The set of boundary points is given by $F_j(C^{est}) = \tau M^{pred}$.

actual makespan will be at most τ times the predicted makespan value.

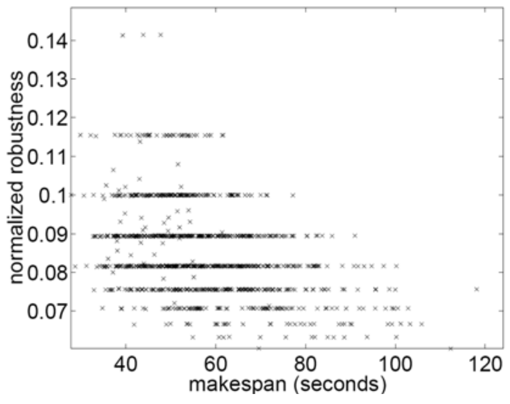
3.2. Utility of Robustness

The experiment in this subsection seeks 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).

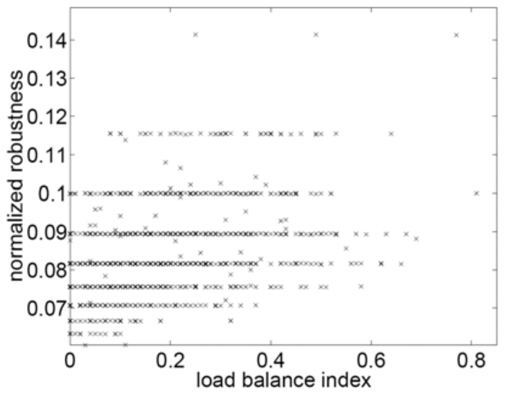
The resource allocations were evaluated for robustness, makespan, and *load balance index* (defined as the ratio of the finishing time of the machine that finishes first to the makespan). The larger the value of the load balance index, the more balanced the load (the largest value being 1). The tolerance, 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.

Figure 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 similar graph for the normalized robustness against the load balance index is shown in Figure 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



(a)



(b)

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

calculation allows one to select an allocation that also provides high robustness. Figure 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.

4. Mapping under Makespan Constraint

4.1. Problem Statement

This section summarizes the research described in [16]. 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 [9]. Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research (e.g., [1, 6, 13]). *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 Ω 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

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

A parallel 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 16 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 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.

Seven static mapping schemes were developed in this study: Max-Max, Greedy Iterative Maximization (GIM), Overhead Iterative Maximization (OIM), GENITOR, Memetic Algorithm (MA), Ant Colony Optimization (ACO), and Hereboy Evolutionary Algorithm. Two are described here.

4.2. Greedy Iterative Maximization

The GIM heuristic can be summarized as follows.
1) A mapping is generated using the Min-Min heuristic [6, 9], based on completion times.
2) Find the robustness metric and the machine with the smallest robustness radius among all machines (*min-radius* machine) for the current mapping.

3) Generate an application list containing all applications on the min-radius machine not yet considered for reassignment.
4) An application is chosen arbitrarily from the application list and considered for reassignment to all other machines.
5) Reassign the application to the machine that improves the robustness metric the most and go to step 2; if the reassignment does not improve the mapping, remove the application from the application list and go to step 4 until there are no applications in the application list.
6) The robustness metric and min-radius machine for the current mapping is determined.
7) Generate an application list containing all applications on the min-radius machine not yet considered for swapping.
8) An application is chosen arbitrarily from the application list and considered to be swapped to all applications on all other machines.
9) The chosen application from the application list is swapped with the first application that will increase the robustness metric by traversing through all the applications in arbitrary order on all other machines and go to step 6; if the chosen application could not be swapped with any other application, remove the application from the application list and go to step 8 until the application list is empty.
10) A new mapping is generated using the MCT heuristic [6, 9] based on completion times. Applications are considered in a different order every time a new mapping is generated by MCT.
11) Repeat steps 2–10 until the one hour time constraint has expired.

One variation tried was to select the "best" application that improves the robustness during swapping in step 9 and was found to perform slightly worse than the "arbitrary order" swap method. It is observed that, in general, the robustness of the initial mapping did not impact the robustness of the final mapping; however, if the robustness of the initial mappings are good, more iterations of steps 2 through 9 can be performed in the given time constraint.

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 method studied here is similar to the standard GENITOR approach used in [18]. 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. The GENITOR operates on a fixed population of 200 chromosomes. The population includes one chromosome (seed) that is the Max-Max [6, 9] solution based on robustness 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.

Next, a special linear bias function [6] 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 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 Figure 3. All the heuristics are run for 100 different scenarios and the average values and 95% confidence intervals are plotted. The GIM and OIM are among the best heuristics for both of the high-high and low-low cases studied here. The IM heuristics that make use of the tailored search technique (as opposed to the general search used by GENITOR) proved to be very effective. The "best" swap variation of the GIM arrived at a good solution faster than the "arbitrary order" swap; however, the latter performed more beneficial swaps and showed a gradual increase in the robustness better than the former. The GENITOR and MA performed comparably to the IM heuristics. Both of the heuristics are seeded with the Max-Max solution. The ACO solution was within 12% of the best heuristic (OIM) solution. In the ACO heuristic, seeding the pheromone trial with the

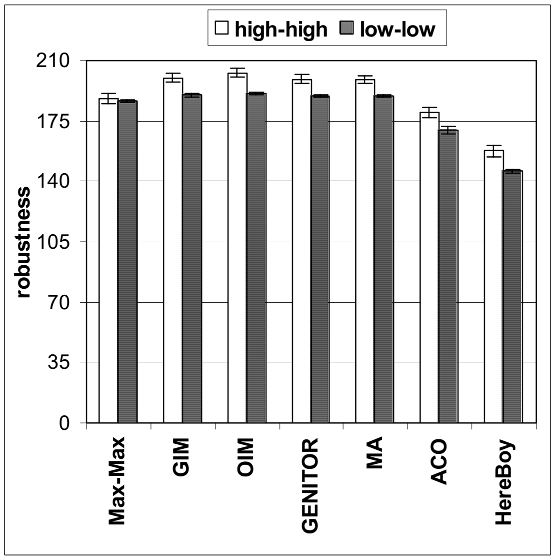


Figure 3: Simulation results for robustness for a given fixed set of machines.

Max-Max mapping and the use of the local search technique improved the solution on average by 27%.

5. Mapping under Makespan and Dollar Cost Constraints

5.1. Problem Statement

This section summarizes the study described in [17], which was an extension of [16]. The research environment here differs from the previous study with the addition of the *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 use 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 [17]). 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 [16].

Six static mapping schemes were developed in this research: Negative Impact Greedy Iterative Maximization (<u>NI-GIM</u>), Parition/Merge Greedy Iterative Maximization (<u>P/M-GIM</u>), Sum Iterative Maximization (<u>SIM</u>), GENITOR, Memetic Algorithm (<u>MA</u>), and Hereboy Evolutionary Algorithm. Two are described here.

5.2. Negative Impact Greedy Iterative Maximization

The NI-GIM heuristic used here is a modification of GIM described in Section 4. The NI-GIM heuristic performs a Min-Min mapping based on completion times assuming all machines to be available, ignoring the dollar cost constraint.

The robustness radius of all the available machines is calculated for the Min-Min mapping. The negative impact of removing machine *j* is determined in the following way. Each of the applications mapped onto machine *j* is evaluated for reassignment to each of the other machines. The decrease in the robustness radius of each available machine *i* if an application *t* is reassigned from machine *j* is calculated; call this $\underline{\Delta_{i,t}}$.

Define average decrease in the robustness radii across all the available machines due to reassignment of application *t* to be

$$\underline{\alpha_t} = \sum_{i=0}^{|\Omega|-1} \Delta_{i,t} \Big/ \text{number of available machines}.$$

The *negative impact* of removing machine *j*, $\underline{NI_j}$, is

$$NI_j = \sum_{t \,\in\, \text{tasks on } j} \alpha_t .$$

The ratio of negative impact to cost is obtained by dividing the negative impact by the cost of the machine *j*. The machine that has the least value of ratio is then removed. The procedure of performing the Min-Min mapping with only the available machines and the ratio calculation to remove another machine is repeated until the cost constraint is met.

For the set of machines determined above that meets the cost constraint, the GIM heuristic is run to determine a mapping that maximizes robustness for the given machine set.

5.3. GENITOR

The GENITOR heuristic developed in this work consists of two phases. For phase 1, a chromosome is a vector of length five, where i^{th} element is the number of machines used in i^{th} class. The phase 1 of 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. 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). Otherwise, it is discarded.

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 GENITOR. The heuristic stops when the criterion of 500 total iterations is met. The 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 in described in Section 4 (a total of 100,000 iterations is used here to stop the phase 2 of GENITOR).

5.4. Experimental Results

The simulation results are shown in Figure 4. All the heuristics are run for 100 different scenarios and the average values and 95% confidence intervals are plotted. The GENITOR and the P/M-GIM heuristic are the best among all the heuristics studied for this problem. Both of these heuristics, on average, had all of

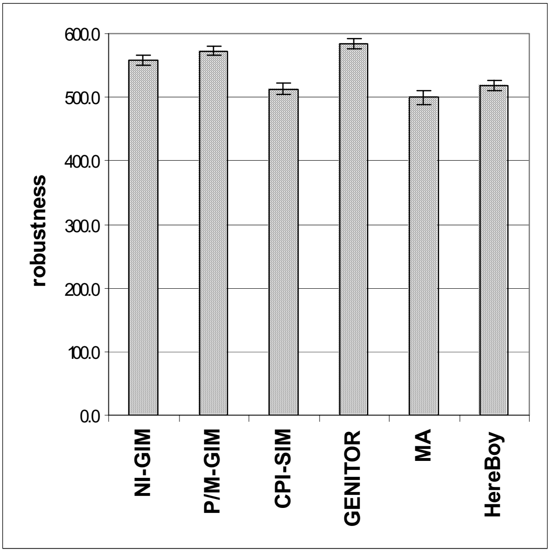


Figure 4: Simulation results for robustness. Machine sets were determined heuristically.

the available machines from Class 4 and Class 5. The NI-GIM heuristic performed comparably to P/M-GIM. The negative impact calculation always forced removal of machines from either Class 2 or 3. All machines from Class 1, 4, and 5 (i.e., the fastest class and the two cheapest classes of machines) were used in more than 90% of the scenarios. The SIM heuristic by itself did not perform well because it always selected machines for relocation that will maximize application-execution or robustness improvement. HereBoy Evolutionary Algorithm is the fastest among all the algorithms and its performance is within 12% of GENITOR. The search technique used for selecting the machines for HereBoy used all of the machines of Class 1, 4, and 5. The machine selection of the MA heuristic based on the random approach proved to be ineffective. Therefore, the robustness achieved on the selected sets was the worst among all the heuristics.

6. Future Work

We are considering extending our current work in different directions, including:

1) Deriving the boundary curves 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. Such information might be available

about individual perturbation parameter elements. One might have only relative information about perturbation parameter elements (e.g., the execution times of different applications). In another case, one might have relative information about different perturbation parameters (e.g., changes in input sensor loads versus changes in the execution times of different applications).
4) Determining when to use Euclidean distance versus a simple sum 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 three 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 parallel systems such that the robustness of the produced mappings is maximized are summarized from [16]. Finally, heuristics for (1) selecting a set of machines and (2) mapping applications to the set of machines, both to maximize robustness, are summarized from [17].

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 Trans. 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 Int'l Conf. 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," *J. 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," *J. Parallel Computing*, Vol. 61, No. 6, June 2001, pp. 810–837.

[7] R. L. Daniels and J. E. Carrillo, "β-Robust scheduling for single-machine systems with uncertain processing times," *IIE Trans.*, Vol. 29, No. 11, Nov. 1997, pp. 977–985,.

[8] S. D. Gribble, "Robustness in complex systems," *8th Workshop Hot Topics in Operating Systems (HotOS-VIII)*, May 2001, pp. 21–26.

[9] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent applications on non-identical processors," *J. ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280–289.

[10] M. Jensen, "Improving robustness and flexibility of tardiness and total flowtime job shops using robustness measures," *J. Applied Soft Computing*, Vol. 1, No. 1, June 2001, pp. 35–52.

[11] E. Jen, "Stable or robust? What is the difference?" *Complexity*, Vol. 8, No. 3, June 2003.

[12] V. J. Leon, S. D. Wu, and R. H. Storer, "Robustness measures and robust scheduling for job shops," *IEE Trans.*, Vol. 26, No. 5, Sept. 1994, pp. 32–43.

[13] M. Sevaux and K. Sorensen, "Genetic algorithm for robust schedules," *8th Int'l Workshop Project Management and Scheduling (PMS 2002)*, Apr. 2002, pp. 330–333.

[14] G. F. Simmons, *Calculus With Analytic Geometry, Second Edition,* McGraw-Hill, New York, NY, 1995.

[15] 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.

[16] P. Sugavanam, H. J. Siegel, A. A. Maciejewski, S. A. Ali, M. Al-Otaibi, M. Aydin, K. Guru, A. Horiuchi, Y. Krishnamurthy, P. Lee, A. Mehta, M. Oltikar, R. Pichel, A. Pippin, M. Raskey, V. Shestak, and J. Zhang, "Processor allocation for applications that is robust against errors in computation time estimates," *14th IEEE Heterogeneous Computing Workshop (HCW 2005) proceedings of 19th Int'l Parallel Processing Symposium (IPDPS 2005)*, April 2005.

[17] P. Sugavanam, H. J. Siegel, A. A. Maciejewski, J. Zhang, V. Shestak, M. Raskey, A. Pippin, R. Pichel, M. Oltikar, A. Mehta, P. Lee, Y. Krishnamurthy, A. Horiuchi, K. Guru, M. Aydin, M. Al-Otaibi, and S. A. Ali, "Robust processor allocation for independent applications when dollar cost for processors is a constraint," *4th Int'l Workshop Algorithms, Models, and Tools for Parallel Computing on Heterogeneous Networks (HeteroPar-05)*, accepted, to appear in 2005.

[18] D. Whitley, "The GENITOR algorithm and selective pressure: Why rank based allocation of reproductive trials is best," *3rd Int'l Conf. Genetic Algorithms*, June 1989, pp. 116–121.