Robust Processor Allocation for Independent Tasks When Dollar Cost for Processors is a Constraint

Prasanna Sugavanam[1], H. J. Siegel[1,2], Anthony A. Maciejewski[1], Junxing Zhang[3], Vladimir Shestak[1], Michael Raskey[4], Alan Pippin[5], Ron Pichel[5], Mohana Oltikar[1], Ashish Mehta[1], Panho Lee[1], Yogish Krishnamurthy[1], Aaron Horiuchi[5], Kumara Guru[1], Mahir Aydin[1], Mohammad Al-Otaibi[1], and Syed Ali[1]

Colorado State University
[1]Department of Electrical & Computer Engineering
[2]Department of Computer Science
Fort Collins, CO 80523-1373
{prasanna, hj, aam, shestak, mohana, ammehta, leepanho, yogi, higuru, mahir, motaibi, sdamjad}@colostate.edu

[3]University of Utah
School of Computing
Salt Lake City, UT 84112
junxing@cs.utah.edu

Hewlett-Packard Company
[4]Linux & Open Source Lab
[5]Systems & VLSI Technology Division
Fort Collins, CO 80528
{akh, rgp, ajp}@fc.hp.com
michael.raskey@hp.com

Abstract

In a distributed heterogeneous computing system, the resources have different capabilities and tasks have different requirements. Different classes of machines used in such systems typically vary in dollar cost based on their computing efficiencies. Makespan (defined as the completion time for an entire set of tasks) is often the performance feature that is optimized. Resource allocation is often done based on estimates of the computation time of each task on each class of machines. Hence, it is important that makespan be robust against errors in computation time estimates. The dollar cost to purchase the machines for use can be a constraint such that only a subset of the machines available can be purchased. The goal of this study is to: (1) select a subset of all the machines available so that the cost constraint for the machines is satisfied, and (2) find a static mapping of tasks so that the robustness of the desired system feature, makespan, is maximized against the errors in task execution time estimates. Six heuristic techniques to this problem are presented and evaluated.

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 and Problem Statement

Heterogeneous computing (HC) systems utilize various resources with different capabilities to satisfy the requirements of diverse task mixtures and to maximize the system performance (e.g., [10, 18]). Such systems often operate in an environment where certain desired performance features degrade due to unpredictable circumstances, such as higher than expected work load or inaccuracies in the estimation of system parameters (e.g., [3, 4, 9, 24, 25, 34]). Thus, it is necessary to allocate resources to tasks to maximize the robustness of the allocation.

The act of assigning (matching) each task to a machine and ordering (scheduling) the execution of the tasks on each machine is known as mapping, resource allocation, or resource management. An important research problem is how to determine a mapping so as to maximize the robustness of desired system features against perturbations in system parameters [4]. The general problem of optimally mapping tasks to machines in an HC environment has been shown to be NP-complete (e.g., [12, 19, 21]). Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research (e.g., [1, 2, 7, 8, 11, 18, 20, 29, 36, 37, 41]).

For this research, a metatask composed of a number of independent tasks (i.e., no communication

between tasks are needed) is considered. Makespan is defined as the completion time for the entire metatask. It is often the performance feature that is optimized. Resource allocation is often done based on estimates of the computation time of each task on each class of machines. A mapping is defined to be robust with respect to specified system performance features against perturbations in specified system parameters if degradation in these features is limited when certain perturbations occur [4]. In this system, it is required that the makespan be robust against errors in task execution time estimates. Specifically, the system is considered robust if the actual makespan under the perturbed conditions does not exceed the required time constraint, τ.

The problem studied here is how to select (purchase) a fixed set of machines, within a given dollar cost constraint, to use to comprise an HC system. It is assumed that this fixed HC system will be used to regularly execute metatasks in a production environment, where the metatasks are from a known problem domain with known estimated computational characteristics. The machines to be purchased for the HC suite are to be selected from different classes of machines, where each class consists of machines of the same type. The machines of different classes differ in dollar costs depending upon their task execution speed. The dollar cost of machines within a class is the same. To be able to use a machine for executing tasks, a one time dollar cost is incurred (i.e., to purchase the machines).

The goal of this study is to: (1) select a subset of all the machines available so that the cost constraint for the machines is satisfied, and (2) find a static mapping of all tasks to the subset so that the robustness of the mapping is maximized. Maximizing the robustness here means maximizing the collective allowable error in execution time estimation for the tasks that can occur without the makespan exceeding the constraint.

A set of T tasks in the metatask is required to be allocated to a chosen set of machines, M. The cost constraint for the machines is given by δ. The estimated time to compute (ETC) value for each task on each class of machines is assumed to be known *a priori*. This assumption is commonly made (e.g., [23]). Approaches for doing this estimation are discussed in [30]. Assume 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 μ must be robust against them.

Let C^{est} be the vector of *estimated* computation times for the T tasks on the machine where they are allocated. Let C be the vector of *actual* computation times (C^{est} plus the estimation error for each task). The finishing time of a given machine j, F_j, depends only on the actual computation times of the tasks mapped to that machine. The performance feature (ϕ) that should be limited in variation to ensure that the makespan is robust is the finishing times of the machines. That is, $\phi = \{F_j \mid 1 \le j \le |M|\}$. The FePIA procedure from [4] is applied to determine the robustness metric for this problem.

The robustness radius [4] of F_j against C for mapping μ, $r_\mu(F_j, C)$, is defined as the largest Euclidean distance that C can change from the assumed value of C^{est} without the finishing time of machine j exceeding the tolerable variation. Mathematically,

$$r_\mu(F_j, C) = \min_{C:\, F_j(C) = \tau} \left\| C - C^{est} \right\|_2. \quad (1)$$

That is, if the Euclidean distance between any vector of actual computation times and the vector of estimated computation times is no larger than $r_\mu(F_j, C)$, then the finishing time of the machine j will be at most the makespan constraint τ. As described in [4], equation (1) can be interpreted as the perpendicular distance from C^{est} to the hyperplane described by the equation $\tau - F_j(C^{est}) = 0$. Hence, equation (1) can be rewritten as [35]

$$r_\mu(F_j, C) = \frac{\left(\tau - F_j\left(C^{est}\right)\right)}{\sqrt{\text{number of tasks mapped to machine } j}}. \quad (2)$$

The robustness metric, $\rho_\mu(\phi, C)$, for the mapping is simply the minimum of all robustness radii over all machines [4]. If the Euclidean distance between any vector of the actual execution times and the vector of the estimated execution times is no larger than $\rho_\mu(\phi, C)$, then the actual makespan will be at most the constraint τ. Mathematically,

$$\rho_\mu(\phi, C) = \min_{F_j \in \phi} r_\mu(F_j, C). \quad (3)$$

The performance metric that is used to evaluate the mapping is $\rho_\mu(\phi, C)$. It is obvious that the larger the robustness metric, the better the mapping.

The goal for this study is to determine the set of machines such that: (1) the makespan is within τ, (2) the cost for the chosen set of machines is within δ, and (3) the robustness metric for actual makespan against ETC errors is maximized. Six static mapping schemes are studied in this research: Negative Impact Greedy Iterative Maximization, Parition/Merge Greedy Iterative Maximization, Sum Iterative Maximization, GENITOR, Memetic Algorithm, and Hereboy Evolutionary Algorithm. The emphasis of this paper is on selecting the set of machines to accomplish the above stated goal. All of the heuristics in the current paper use as a component machine assignment heuristics from our earlier work in [39], which assumed a given, fixed set of machines (the earlier work did not involve selecting a set of machines for purchasing which is the focus of the current paper). Simulations are

used to evaluate and compare the six static heuristics studied in this research.

The next section describes the simulation setup used for this research. Section 3 provides literature related to this work. In Section 4, the heuristics studied in this research and an upper bound are presented. Section 5 discusses the results, and the last section gives a brief summary of this research work.

2. Simulation Setup

An HC system with five different classes of machines, eight machines in each class, and $T = 1024$ independent tasks is simulated. This large number of machines and tasks are chosen to present a significant mapping challenge for the heuristics.

The estimated execution times of all tasks taking heterogeneity into consideration are generated using the gamma distribution method described in [5]. The ETCs used in this research are of the high task and low machine (across various classes) heterogeneity (high-low). In this study, the ETCs are of the type consistent [5] across different classes of machines; i.e., if a class *i* machine is faster than a class *j* machine for one task, it is faster for all tasks. All the machines in a given class are homogeneous (the execution time of any given task on all the machines is the same). These assumptions are made to represent a realistic environment. The machines with higher (dollar) cost typically are equipped with faster processor(s), bigger memory etc., and in general, executes tasks faster than the low-end cheaper machines. The estimated execution time of task *i* on machine *j* is given by $ETC(i, j)$. For this study, heuristics are run for a total of 100 scenarios, where each scenario corresponds to a different ETC matrix.

A task mean and coefficient of variation (COV) are used to generate the ETC matrices. A mean task execution time of 180 seconds and COV of 0.9 (task heterogeneity) is used to calculate the values for all the elements in a task vector (where the number of elements is equal to the number of tasks). Then using the i^{th} element of the vector as the mean and a COV of 0.3 (machine heterogeneity), the ETC values for task *i* on all the different classes of machines are calculated. The ETC values are then sorted in ascending order to obtain the consistent heterogeneity. Class 1 is the fastest machine, Class 2 is the second fastest, and so on. The cost values are in accordance with their execution speeds, with Class 1 being the most expensive and Class 5 the cheapest. The cost values of different classes of machines are shown in Table 1. These values are based on specific configurations of DELL desktop, workstation, and server products. It is assumed that all machines in a class use the same software environment.

Class	1	2	3	4	5
Cost (dollars)	1800	1500	1200	800	500

Table 1: The cost values for different classes of machines.

The cost constraint δ is chosen so that not all machines in the suite can be used, and the actual makespan constraint τ is chosen so that it adds significant mapping challenge to the problem. Experiments with simple greedy heuristics were used to decide the value of the cost constraint to be 34,800 dollars and the time constraint 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. Because the tasks are independent, there is no communication between tasks. The time and resources required for loading the task executable file is simply assumed to be the same on all the machines. Hence, the network characteristics will not affect the solution of the problem and so it is ignored. The performance of each heuristic is studied across all 100 different ETCs. In this study, the wall clock time for the mapper itself to execute is arbitrarily required to be less than or equal to 60 minutes for any scenario on a typical unloaded 3GHz Intel Pentium 4 machine.

3. Related Work

The current work is an extension of our earlier work in [39], where robust static mapping heuristics against errors in ETCs were derived. The tasks were mapped to a given, fixed set of machines in [39] and dollar cost was not a constraint in that environment. The research environment here differs from [39] with the addition of the cost constraint for the machines and choosing a subset of all the available machines to be used. Moreover, in this study, the machines are divided into classes and the machines are of consistent heterogeneity across the different classes. In [39], the machines were of inconsistent heterogeneity (i.e., the consistency property did not hold). The robustness metric used in the current work and in [39] is derived using the four step FePIA procedure detailed in [4]. All of the heuristics in the current paper use as a component machine assignment heuristics from our earlier work in [39].

A number of papers have studied the issue of robustness in distributed systems (e.g., [9, 13, 14, 17, 25, 28, 38]). Robust decision making formulations presented in [13, 24, 25] motivate building a robust suboptimal solution over a better performing solution that is less robust. A detailed discussion of how the assignment portion of the current work differs from the examples above is given in [39].

The literature was examined to select a set of heuristics appropriate for the HC environment considered here. The Iterative Maximization (IM) techniques are a variation of the iterative deepening and random search techniques used in [16]. The partition/merge methods used in [26] are adapted to our environment to find the set of machines to be used for the Greedy Iterative Maximization heuristic. The GENITOR-style genetic algorithm used here is an adaptation of [40]. GENITOR is a steady-state genetic algorithm (GA) that has been shown to work well for several problem domains, including resource allocation and job shop scheduling, and hence was chosen for this problem. The research works in [15, 33] have used variations of GA for the synthesis of heterogeneous multiprocessors in embedded systems. Memetic Algorithm (MA) [6, 31, 32], also called the hybrid GA, applies a separate local search process (hill-climbing) to refine individuals. Combining global and local search is a strategy used by many successful global optimization approaches [6]. In this study, the MA heuristic is applied to maximize robustness using a specific set of machines. The HereBoy Evolutionary Algorithm used here is a combination of GA and Simulated Annealing (SA) and is an adaptation of [27] that was applied to the evolvable hardware problem. This fast evolutionary algorithm is shown to be well suited for exploring large spaces and can be applied to a wide range of optimization problems. The HereBoy is used for maximizing robustness with a specific set of machines that is determined using a search similar to partition/merge method [26].

4. Heuristics Descriptions

4.1. Overview

Five of the six heuristics studied for this problem, Negative Impact Greedy Iterative Maximization, Partition/Merge Greedy Iterative Maximization, GENITOR, Memetic Algorithm, and Hereboy Evolutionary Algorithm, involve two phases. In phase 1, a subset of machines is selected using specific heuristic techniques to meet the cost and makespan constraints, and to maximize robustness. In phase 2, tasks are mapped to the set of machines found in phase 1 to further maximize the robustness metric for the mapping. Recall from Section 1 that all of the heuristics in the current paper use as a component machine assignment heuristics from our earlier work in [39], which assumed a given, fixed set of machines. The Sum Iterative Maximization heuristic involves only one phase where a robustness maximization criterion is used to select machines such that the cost constraint is always satisfied. Throughout the description of the heuristics, Class 1 of machines is referred to as the highest class and Class 5 of machines is referred to as the lowest class.

4.2. Negative Impact Greedy Iterative Maximization

The Negative Impact Greedy Iterative Maximization (NI-GIM) heuristic used here is a modification of GIM described in [39]. The NI-GIM heuristic performs a Min-Min [21] mapping (procedure described in Figure 1) based on the completion times assuming all machines to be available, irrespective of the cost constraint.

1. A task list is generated that includes all unmapped tasks.
2. Find the completion time of each unmapped task on each machine (ignoring other unmapped tasks).
3. Find the machine that gives the minimum completion time for each task.
4. Among all the task/machine pairs found in 3, find the pair that gives the minimum completion time.
5. Remove the above task from the task list and map it to the chosen machine.
6. Update the available time of the machine on which the task is mapped.
7. Repeat steps 2-6 until all the tasks have been mapped.

Figure 1: Pseudo-code for the Min-Min heuristic.

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 tasks mapped onto machine j is evaluated for reassignment to all the other machines. The decrease in the robustness radius of each available machine i if a task t is reassigned from machine j is calculated; call this $\Delta_{i,t}$. Define average decrease in the robustness radii across all the available machines due to reassignment of task t to be $\alpha_t = \sum_{i=0}^{|M|-1} \Delta_{i,t} \Big/ \text{number of available machines}$. The negative impact of removing machine j, NI_j, is $NI_j = \sum_{t \in \text{tasks on } j} \alpha_t$.

The ratio of negative impact to cost is obtained by simply dividing the negative impact by the cost of the machine j. The machine that has the least value of the

negative impact to cost 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 satisfied.

For the set of machines determined above that meets the cost constraint, the GIM heuristic (please refer to [39]) is run to determine a mapping that maximizes robustness for the given machine set.

4.3. Partition/Merge Greedy Iterative Maximization

The phase 1 of Partition/Merge Greedy Iterative Maximization (P/M-GIM) starts with a random number of machines chosen from each class. The tasks are then mapped to the selected machines using the Min-Min heuristic. The makespan for the Min-Min mapping is calculated. It was observed that makespan constraint in this study is such that if the cost constraint is violated, the makespan constraint is always satisfied using Min-Min. Hence, either both of the constraints are satisfied or only one of the two constraints is violated using Min-Min. If the cost constraint is violated, then the task-merge (machine removal) [26] technique is executed. Otherwise, the task-partition (machine addition) [26] technique is executed to improve the makespan. Partitioning is stopped if addition of another machine will violate the cost constraint and merging is stopped once the cost constraint is satisfied.

Five different methods for partitioning and merging are implemented: (a) cheap, (b) expensive, (c) even distribution, (d) most common, and (e) random. In the cheap variation, the partition step added the cheapest available machine, or the merge step removed a machine in the most expensive class. The expensive variation did exactly the opposite (removed a cheapest machine or added the most expensive). Even distribution attempted to merge or partition so that a similar number of machines from each class would be available (ties are broken arbitrarily). The most common approach attempted to add machines to the class that already had the most machines or to remove from the class that had the least number of machines (ties are broken arbitrarily). The random variation simply involved partitioning or merging an available machine from a randomly selected class.

After generating a valid mapping that satisfies the cost and makespan constraints using one of the above techniques, reassignment and swapping of the GIM heuristic [39] are executed in an attempt to improve the robustness metric of the mapping. This constitutes phase 2. The reassignment and swapping of the GIM heuristic is executed for 20 unique machine combinations (found using phase 1) and the best solution is output.

4.4. Sum Iterative Maximization

The Sum Iterative Maximization (SIM) heuristic starts with a cost lower bound (CLB) mapping where all the tasks are mapped onto a single lowest cost machine. There cannot be a mapping that has a lower cost than the cost lower bound mapping. However, because this mapping is not guaranteed to have a makespan less than τ, reassignment of some tasks to other machines may be necessary before continuing on to the next step of improving the robustness metric. It is assumed that all the machines are available for the reassignment of tasks. When a machine is used for the first time, the cost for using the machine is paid and the total cost of all the machines used in the suite must be less than δ. After the reassignment procedure, a task swapping procedure is executed. For this heuristic, the task-execution improvement, defined as the decrease in the sum of the completion times of the machines after reassignment or swapping, and the robustness improvement, defined as the increase in the sum of the robustness radius of the machines after reassignment or swapping, are maximized. The worst robustness machine is defined as the machine with the least robustness radius. The general procedure of the SIM heuristic used here is similar to that used in [39]. The SIM procedure can be summarized as follows:

1. Begin with the CLB mapping.
2. Find the makespan machine (the machine that finishes last) for the current mapping.
   a. For each task on the makespan machine, consider relocating it to a different machine. If the relocation will not reduce the makespan and keep the cost $\leq \delta$, it is ignored. Otherwise, the task-execution improvement of the particular relocation is recorded in a list H.
   b. Select the relocation in H that has the maximum improvement.
   c. Relocate the task, and empty the list H.
3. Repeat step 2 until makespan $\leq \tau$ or no task can be relocated.
4. If makespan $\leq \tau$, go to step 7, otherwise go to step 5.
5. Find the makespan machine for the current mapping.
   a. For each task on the makespan machine, consider swapping it with a task on a different machine. If the swap will not reduce the makespan, it is ignored. Otherwise, the task-execution improvement of the relocation is recorded in the list H.
   b. Select the relocation in H that has the maximum improvement.
   c. Relocate the task, and empty the list H.

6. Repeat step 5 until makespan $\leq \tau$. If this is not possible, the mapping procedure fails (for our study, this never happened).
7. Find the worst robustness machine for the current mapping.
    a. For each task on the worst robustness machine, consider relocating it to a different machine. If the relocation will not increase the robustness and keep the cost $\leq \delta$, it is ignored. Otherwise, the robustness improvement of the relocation is recorded in a list H.
    b. Select the relocation in H that has the maximum improvement.
    c. Relocate the task, and empty the list H.
8. Repeat step 7 until no task can be relocated.
9. Find the worst robustness machine for the current mapping.
    a. For each task on the worst robustness machine, consider swapping it with a task on another machine. If the swap will not increase the robustness, it is ignored. Otherwise, the robustness improvement of the relocation is recorded in a list H.
    b. Select the relocation in H that has the maximum improvement.
    c. Relocate the task, and empty the list H.
10. Repeat step 9 until no task can be swapped.

A variation of this heuristic uses a predetermined set of minimum cost machines such that adding another machine will violate the cost constraint. For this set of lowest cost machines chosen that meets the cost constraint, relocations are made based on the task-execution or robustness improvement as before. For another variation, define cost performance index (CPI) of machine j as the product of the cost of machine j and the average ETC of all tasks on machine j. The machines with the lowest CPI are selected until the cost is less than or equal to δ for mapping tasks. For this machine set, the relocation and swapping are done as explained above.

4.5. GENITOR

GENITOR is a general optimization technique that is a variation of the genetic algorithm approach. It manipulates a set of possible solutions. For phase 1, a chromosome is a vector of length five, where i^{th} element is the number of machines 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. The chromosomes are evaluated using the robustness metric based on a machine assignment using the Max-Max mapping from [39]. The entire population is sorted in descending order based on the robustness metric of the Max-Max heuristic.

The special function for selecting parent chromosomes is a linear bias function, used to provide a specific selective pressure [40]. The linear bias value of 1.5 was used to select chromosomes for crossover and mutation. A bias of 1.5 implies that the top ranked chromosome in the population is 1.5 times more likely to be selected for a crossover or mutation than the median chromosome. Elitism, the property of guaranteeing the best solution remains in the population, is implicitly implemented by always maintaining the ranked list.

In the crossover step, for the pair of the selected parent chromosomes, 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 increases the number of machines of the first chosen class by one and decreases the number of machines of the other by one. If the chromosome is infeasible, that is, if it violates the cost constraint or the possible number of machines in each class, 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 [39] (a total of 100,000 iterations is used here to stop the phase 2 of GENITOR).

4.6. Memetic Algorithm

The Memetic Algorithm (MA) metaheuristic [31] combines population-based global search with local search made by each of the individuals. In phase 1, 100 random combinations of machines from each class are chosen such that the cost constraint is satisfied. Each of the 100 combinations is evaluated using the Max-Max heuristic [39] and the machine combination that has the highest robustness metric is selected. In phase 2, for the best machine combination found in phase 1, the MA heuristic identical to that described in [39] is executed, the only difference being the stopping criterion. That is,

a total of 40,000 iterations is used here to stop the phase 2 of MA.

4.7. HereBoy Evolutionary Algorithm

HereBoy is a fast evolutionary algorithm that combines the features of GA and SA [27]. Phase 1 of HereBoy starts with adding one machine in each class (starting from the lowest class) in a round robin fashion until the cost constraint is violated. The current machine combination is evaluated using the robustness metric based on a machine assignment made by the Max-Max mapping [39].

Now, starting from the highest class, a new machine is considered to be included in the existing machine set in a round robin fashion (unless no more machines from a particular class can be added). Adding another machine will violate the cost constraint. Hence, to be able to accommodate the inclusion of a machine, one or more machines from other classes should be removed. Machines are considered to be removed from a single class or from two different classes (this is sufficient to add a machine of any class). All such combinations are considered and if removing a particular combination of machines allows adding another machine of a lower class (after adding the higher class machine under consideration), then a machine is added. For each combination of machines that is removed, and replaced by other machines, a new set of working machines is formed. All machine sets are evaluated using the mapping produced by Max-Max and the set that gives the highest robustness metric is stored as the best. For the current best machine set, the above described procedure is repeated until addition of a machine from any class will not improve the robustness metric.

For the best combination of phase 1 procedure, HereBoy Evolutionary Algorithm [39] is executed as phase 2 to determine the task to machine mapping for that combination of machines.

4.8. Upper Bound

The upper bound on the robustness metric for this study is similar to that for [39]. It assumes a homogeneous MET system in which the execution time for each task on all machines is the same and equal to the minimum time that the task would take to execute across the original set of machines. The minimum execution time of task i, $\underline{MET_i}$, is given by the following equation.

$MET_i = \min ETC(i, j)$ over all j.

The upper bound for the robustness metric of the homogeneous MET system is equal to or better than the upper bound for the robustness metric of the original system because of the impact of the MET values in equation (2).

For this problem, there cannot be more than 33 machines in the system for the given cost constraint. This includes the 33 machines of the lowest class possible in the entire HC suite. Following equation (2) and our assumption of the homogeneous MET system, having more machines in the suite gives a better robustness metric than having fewer machines in the suite (due to the impact of number of tasks on each machine).

The tasks in the MET system are now arranged in ascending order of their execution times. Then, the robustness upper bound is calculated as follows. Let $\underline{N} = \lfloor |T|/|M| \rfloor$. Here, $|M| = 33$. The first N tasks in the sorted order are stored in a list $\underline{S}$. For the purposes of this mathematical upper bound derivation, the same N tasks in S are assumed to be on all the machines so that $F_j = F_i$, $1 \leq i,j \leq |M|$. Thus, a very loose upper bound for robustness is given by the following equation.

$$\text{UB} = \frac{\left(\tau - \sum_{i=0}^{N-1} MET_{S(i)} \right)}{\sqrt{N}}$$

The proof of this upper bound is identical to that given in [39].

5. Experimental Results

The simulation results are shown in Figures 2 and 3. All the heuristics are run for 100 different scenarios and the average values and 95% confidence intervals [22] are plotted. The running times of the heuristics averaged over 100 trials, mapping 1024 tasks in each trial, are shown in Table 2.

The GENITOR and "cheap" variation of the P/M-GIM heuristic are the best among all the heuristics studied for this problem (the cheap variation is shown in the figures). Both of these heuristics, on average, had all of the available machines from Class 4 and Class 5. The "cheap" variation of P/M-GIM heuristic always removed machines from Class 1 if the cost constraint was violated. But GENITOR explored the search space better and on average used more machines in Class 1 than in Class 2. The "most common" and "random" variations of P/M-GIM heuristic were within 10% of the "cheap" variation. The "expensive" variation performed the worst among all the variations of P/M-GIM and "even distribution" was slightly better than the "expensive" variation. These two variations did not have as many machines in the suite as compared to the other variations. For this problem, having a good balance between the execution speed of machines and the number of machines in the HC suite proved to be

important for maximizing the robustness of the mapping.

The NI-GIM heuristic performed comparably to P/M-GIM and GENITOR. 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.

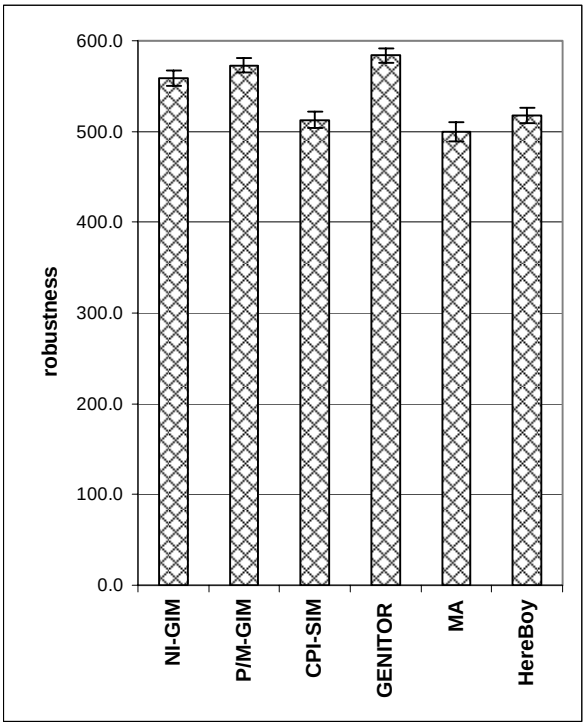

Figure 2: The simulation results for robustness. The average UB value is 2019.3.

The SIM heuristic by itself did not perform well (an average of 252 for the robustness metric across 100 scenarios). The poor performance is because it always selected machines for relocation that will maximize task-execution or robustness improvement. Therefore, SIM typically picked machines in the order of the highest class to the lowest. The SIM heuristic does not consider replacing a fast machine with multiple slower machines. The cost performance index variation of SIM (CPI-SIM) performed within 12% of GENITOR. The lowest cost variation also performed similarly and is within 2% of the CPI-SIM variation. The lowest cost variation always picked machines in the order of Class 5, 4, 3, 2, and 1. The CPI variation always picked machines in the order of Class 5, 4, 3, 1, and 2. No machines from Class 2 are included in the CPI variation of SIM because of the cost constraint.

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 MA heuristic that made use of the random search approach to find the set of machines in phase 1 performed the worst among all the heuristics. The MA optimization heuristic has proved to work well for a similar environment in [39]. However, the machine selection by the random approach proved to be ineffective for this kind of an environment.

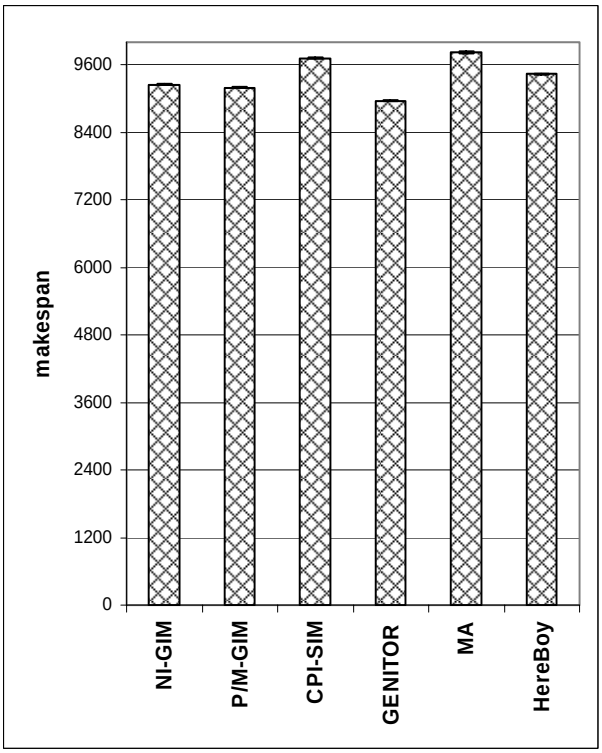

Figure 3: The simulation results for makespan (τ = 12,000).

The phase 2 of all the heuristics discussed in this research is similar to the heuristics studied in [39]. The SIM heuristic performed well for the problem in [39], where inconsistent heterogeneity between machines is considered. However, due to consistent heterogeneity considered in this study, the sum of the task-execution or robustness improvement of machines did not help to find a good solution. The GIM heuristic performed well here because it focused on maximizing the robustness metric itself unlike SIM. The discussion on the performance of phase 2 of GENITOR, MA, and HereBoy are similar to those discussed in [39].

Comparing the robustness metric and makespan results of CPI-SIM and HereBoy, it can be easily noticed that for a similar robustness metric, the makespan results vary about 3% on average for the scenarios studied. Similar differences in robustness metric and makespan results were also shown in [39].

These results prove that minimizing makespan is not the same as maximizing robustness and vice versa.

heuristic	average execution times (seconds)
NI-GIM	3600
P/M-GIM	3600
CPI-SIM	780
GENITOR	3420
Memetic Algorithm	3000
HereBoy	26

Table 2: The average execution times of the heuristics averaged over 100 trials (using a typical unloaded 3 GHz Intel Pentium 4 machine).

6. Summary

This study presents six static heuristics for selecting a set of machines, under a given dollar cost constraint, that will maximize the robustness of a mapping against errors in the ETC. A collection of independent tasks is mapped onto a set of heterogeneous classes of machines using the heuristics described in this research.

The best robustness metric is obtained by using the GENITOR heuristic. The Partition/Merge Greedy Iterative Maximization heuristic performed comparably with its robustness metric within 2% of GENITOR. The execution times for both of the heuristics themselves are also comparable. Thus, both GENITOR and Partition/Merge Greedy Iterative Maximization are a good choice for the given problem.

In this study, a suite of at most 33 machines from five classes were used to execute 1024 tasks. Future work could include examining bigger scenarios, where all of the above parameters are larger.

Acknowledgments: The authors thank Shoukat Ali and Jay Smith for their valuable comments.

References

[1] S. Ali, T. D. Braun, H. J. Siegel, A. A.Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "Characterizing resource allocation heuristics for heterogeneous computing systems," *Computer Architecture*, A. R. Hurson, ed., a volume of Advances in Computers, Elsevier, New York, NY, to appear in 2005.

[2] S. Ali, J.-K. Kim, H. J. Siegel, A. A. Maciejewski, Y. Yu, S. B. Gundala, S. Gertphol, and V. Prasanna, "Greedy heuristics for resource allocation in dynamic distributed real-time heterogeneous computing systems," *2002 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA 2002*), June 2002, pp. 519–530.

[3] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Robust resource allocation for sensor-actuator distributed computing systems," *The 2004 International Conference on Parallel Processing (ICPP 2004)*, Aug. 2004, pp. 174–185.

[4] 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, July 2004, pp. 630–641.

[5] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering*, Special 50th Anniversary Issue, Vol. 3, No. 3, Nov. 2000, pp. 195–207 (invited).

[6] S. Areibi, M. Moussa, and H. Abdullah, "A comparison of genetic/memetic algorithms and heuristic searching," *International Conference on Artificial Intelligence (IC-AI 2001)*, June 2001.

[7] H. Barada, S. M. Sait, and N. Baig, "Task matching and scheduling in heterogeneous systems using simulated evolution," *10th IEEE Heterogeneous Computing Workshop (HCW 2001)*, Apr. 2001.

[8] I. Banicescu and V. Velusamy, "Performance of scheduling scientific applications with adaptive weighted factoring," *10th IEEE Heterogeneous Computing Workshop (HCW 2001)*, Apr. 2001.

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

[10] T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Heterogeneous computing: Goals, methods, and open problems," *2001 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA 2001*), June 2001, pp. 1–12 (invited keynote paper).

[11] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and Bin 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, June 2001, pp. 810–837.

[12] E. G. Coffman, Jr. ed., *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, NY, 1976.

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

[14] A. J. Davenport, C. Gefflot, and J. C. Beck, "Slack-based techniques for robust schedules," *6th European Conference on Planning*, Sep. 2001, pp. 7–18.

[15] R. P. Dick and N. K. Jha, "MOGAC: A multiobjective genetic algrithm for the co-synthesis of hardware-software embedded systems," IEEE Transactions on Computer-Aided Design, Vol. 17, No. 10, Oct. 1998, pp. 920-935.

[16] J. Dorn, M. Girsch, G. Skele, and W. Slany, "Comparison of iterative improvement techniques for schedule optimization," *European Journal on Operations Research*, Vol. 94, No. 2, Oct. 1996, pp. 349–361.

[17] J. Dorn, R. M. Kerr, and G. Thalhammer, "Reactive scheduling: Improving the robustness of schedules and restricting the effects of shop floor disturbances by fuzzy reasoning," *International Journal on Human-Computer Studies*, Vol. 42, No. 6, June 1995, pp. 687–704.

[18] M. M. Eshaghian, ed., *Heterogeneous Computing*, Norwood, MA, Artech House, 1996.

[19] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transaction on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

[20] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, San Fransisco, CA, Morgan Kaufmann, 1999.

[21] 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, Apr. 1977, pp. 280–289.

[22] R. Jain, *The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling*, Wiley, New York, 1991.

[23] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, Vol. 6, No. 3, July-Sep. 1998, pp. 42–51.

[24] P. Kouvelis and G. Yu, *Robust Discrete Optimization and Its Applications*, Dordrecht, Kluwer, 1997.

[25] P. Kouvelis, R. Daniels, and G. Vairaktarakis, "Robust scheduling of a two-machine flow shop with uncertain processing times," *IIE Transactions*, Vol. 38, No. 5, May 2000, pp. 421–432.

[26] S. M. Kroumba, G. Bois, Y. Savaria, "A Synthesis Approach for the Generation of Parallel Architectures," *37th Midwest Symposium on Circuits and Systems*, Vol. 1, 3-5 Aug. 1994, pp. 323–326.

[27] D. Levi, "Hereboy: A fast evolutionary algorithm," *2nd NASA/DoD Workshop on Evolvable Hardware (EH '00)*, July 2000, pp. 17–24.

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

[29] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, Nov. 1999, pp. 107–121.

[30] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *Encyclopedia of Electrical and Electronics Engineering*, J. G. Webster, ed., Vol. 8, John Wiley & Sons, New York, NY, 1999, pp. 679–690.

[31] P. Moscato, *On Evolution, Search, Optimization, Genetic Algorithms, and Martial Arts: Towards Memetic Algorithms,* Technical Report, Caltech Concurrent Computation Program C3P 826, California Institute of Technology, Pasadena, CA, 1989.

[32] G. C. Onwubolu and B. V. Babu, *New Optimization Techniques in Engineering*, Springer-Verlag, New York, NY, 2004.

[33] A. Rae and S. Parameswaran, "Application-specific heterogeneous multiprocessor synthesis using differential-evolution," *11th International Symposium on System Synthesis*, Dec. 1998, pp. 83–88.

[34] M. Sevaux and K. Sörensen, "Genetic algorithm for robust schedules," *8th International Workshop on Project Management and Scheduling (PMS 2002)*, Apr. 2002, pp. 330–333.

[35] G. F. Simmons, *Calculus with Analytic Geometry*, *Second Edition*, New York: McGraw-Hill, 1995.

[36] S. Shivle, R. Castain, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, P. Sugavanam, and J. Velazco, "Static mapping of subtasks in a heterogeneous ad hoc grid environment," *13th IEEE Heterogeneous Computing Workshop (HCW 2004)*, Santa Fe, NM, Apr. 2004.

[37] S. Shivle, 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, P. Sugavanam, and J. Velazco, "Mapping of subtasks with multiple versions in a heterogeneous ad hoc grid environment," *3rd International Workshop on Algorithms, Models, and Tools for Parallel Computing on Heterogeneous Networks (HetroPar 2004)*, Cork, Ireland, July 2004.

[38] 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.

[39] 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 tasks that is robust against errors in computation time estimates," *14th IEEE Heterogeneous Computing Workshop (HCW 2005)*, in the proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), Apr. 2005.

[40] 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.

[41] M.-Y. Wu, W. Shu, and H. Zhang, "Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems," *9th IEEE Heterogeneous Computing Workshop* (*HCW 2000*), May 2000, pp. 375–385.