Dynamic Resource Allocation Heuristics for Maximizing Robustness with an Overall Makespan Constraint in an Uncertain Environment

Ashish M. Mehta*, Jay Smith†, H. J. Siegel*‡, Anthony A. Maciejewski*, Arun Jayaseelan*, and Bin Ye*

*Electrical and Computer Engineering Department

‡Computer Science Department

Colorado State University, Fort Collins, CO 80523–1373

Email: {ammehta, hj, aam, arunj}@engr.colostate.edu

binye@simla.colostate.edu

†IBM 6300 Diagonal Highway Boulder, CO 80301

Email: bigfun@us.ibm.com

***Abstract*— Heterogeneous parallel and distributed computing systems may operate in an environment where certain system performance features degrade due to unpredictable circumstances. Robustness can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed. This work uses a mathematic expression of robustness for a dynamic environment where task execution time estimates are known to contain errors. Several heuristic solutions to the problem are presented that utilize this expression of robustness to influence mapping decisions. These solutions are then compared to a bound on the highest attainable robustness of the described system.**

***Index Terms*— robustness, resource allocation, makespan, dynamic heuristics.**

I. Introduction

Heterogeneous parallel and distributed computing is defined as the coordinated use of compute resources that have different capabilities to optimize certain system performance features. Heterogeneous systems may operate in an environment where certain system performance features degrade due to unpredictable circumstances or inaccuracies in estimated system parameters. Robustness can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed [3]. An important research problem in resource management is how to determine an assignment of tasks to machines and scheduling of tasks (i.e., a mapping or resource allocation) that maximizes the robustness of a system performance feature against perturbations in system parameters.

This research focuses on a dynamic heterogeneous mapping environment where task arrival times are not known *a priori*. A mapping environment is considered dynamic when tasks are mapped as they arrive, i.e., in an on-line fashion [20]. The general problem of optimally mapping tasks to machines in heterogeneous parallel and distributed computing environments has been shown in general to be NP-complete (e.g., [7], [10], [14]). Thus, the development of heuristic techniques to find a near-optimal solution for the mapping problem is an active area of research (e.g., [1], [2], [9], [11], [18], [20], [23], [28]).

The target hardware platform assumed is a dedicated cluster of heterogeneous machines (as opposed to a geographically dispersed, loosely connected grid). Such a cluster may be found in a military command post. The tasks considered in this research are assumed to be taken from a frequently executed predefined set, such as may exist in a military, lab or business computing environment. The estimated time to compute (ETC) values of each task on each machine are assumed to be known based on user supplied information, experiential data, task profiling and analytical benchmarking, or other techniques (e.g., [12], [13], [16], [21], [30]). Determination of ETC values is a separate research problem, and the assumption of such ETC information is a common practice in mapping research (e.g., [13], [15]–[17], [26], [29]).

For a given set of tasks, estimated makespan is defined as the completion time for the entire set of tasks based on ETC values. However, these ETC estimates may deviate from actual computation times; e.g., actual task computation times may depend on characteristics of the input data to be processed. For this research, the actual makespan of a resource allocation is required to be robust against errors in estimated task execution times.

This research builds on earlier work in [22] that uses robustness in a dynamic environment. In [22], makespan was minimized while attempting to maintain a given robustness constraint. In this research, robustness will be maximized while maintaining a constraint on the allowable makespan for the resource allocation. Maximizing robustness in this context is equivalent to maximizing the amount of tolerable variation

This research was supported by the DARPA Information Exploitation Office under contract No. NBCHC030137, 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. Approved for public release, distribution unlimited.

that can occur in ETC times for mapped tasks while still ensuring that a makespan constraint can be met by the resource allocation. Because the optimization goals and constraints differ between this paper and the work in [22], the heuristics developed are different.

Dynamic mapping heuristics can be grouped into two categories: immediate mode and batch mode [20]. Immediate mode heuristics *immediately* map a task to some machine in the system for execution upon the task's arrival. Batch mode heuristics accumulate tasks until a specified condition is satisfied (e.g., a certain number of tasks have been accumulated, or a specified amount of time has elapsed) before mapping tasks. When the specified condition has been satisfied a mapping event occurs and the entire batch of tasks (including any previously enqueued tasks that have not yet begun execution) is considered for mapping. A pseudo-batch mode can be defined where the batch of tasks considered for mapping is determined upon the arrival of a new task (i.e., a mapping event occurs) that consists of all tasks in the system that have not yet begun execution on some machine and are not next in line to begin execution, i.e., previously mapped but unexecuted tasks can be remapped. Pseudo-batch mode heuristics were considered for this research. Therefore, in this system a mapping event occurs whenever a new task arrives.

One of the areas where this work is directly applicable is the development of resource allocations in enterprise systems that support transactional workloads sensitive to response time constraints, e.g., time sensitive business processes [24]. Often, the service provider in these types of systems is contractually bound through a service level agreement to deliver on promised performance. The dynamic robustness metric can be used to measure a resource allocation's ability to deliver on a performance agreement.

The contributions of this paper include:

1) heuristics for solving the above resource management problem,
2) simulation results for the proposed heuristics, and
3) an upper bound on robustness for the resource management problem.

The remainder of the paper is organized as follows. Section II formally states the problem statement for this research. Heuristic solutions to the proposed problem, including the definition of an upper bound on the attainable robustness value are presented in Section III. Section IV describes the simulation setup. The simulation results are outlined and examined in Section V. The related work is considered in Section VI and Section VII concludes the paper.

II. Problem Statement

In this study, T independent tasks (i.e., there is no intertask communication) arrive at a mapper dynamically, where the arrival times of the individual tasks are not known in advance. Arriving tasks are each mapped to one machine in the set of M machines that comprise the heterogeneous computing system. Each machine is assumed to execute a single task at a time (i.e., no multitasking). In this environment, the robustness of a resource allocation must be determined at every mapping event—recall that a mapping event occurs when a new task arrives to the system. Let $T(t)$ be the set of tasks either currently executing or pending execution on any machine at time t, i.e., $T(t)$ does not include tasks that have already completed execution. Let $F_j(t)$ be the predicted finishing time of machine j for a given resource allocation μ based on the given ETC values. Let $MQ_j(t)$ denote the subset of $T(t)$ previously mapped to machine j's queue and let $scet_j(t)$ denote the starting time of the currently executing task on machine j. Mathematically, given some machine j

$$F_j(t) = scet_j(t) + \sum_{\forall i \in MQ_j(t)} ETC(i, j). \tag{1}$$

Let $\beta(t)$ denote the maximum of the finishing times $F_j(t)$ for all machines at time t—i.e., the predicted makespan at time t. Mathematically,

$$\beta(t) = \max_{\forall j \in [1,M]} \{F_j(t)\}. \tag{2}$$

The robustness metric for this work has been derived using the procedure defined in [3]. In our current study, given uncertainties in the ETC values, a resource allocation is considered robust if, at a mapping event, the *actual* makespan is no more than τ seconds greater than the *predicted* makespan. Thus, given a resource allocation μ, the robustness radius $r_\mu(F_j(t))$ of machine j can be quantitatively defined as the maximum collective error in the estimated task computation times that can occur where the actual makespan will be within τ time units of the predicted makespan. Mathematically, building on a result in [3],

$$r_\mu(F_j(t)) = \frac{\tau + \beta(t) - F_j(t)}{\sqrt{|MQ_j(t)|}}. \tag{3}$$

The robustness metric $\rho_\mu(t)$ for a given mapping μ is simply the minimum of the robustness radii over all machines [3]. Mathematically,

$$\rho_\mu(t) = \min_{\forall j \in [1,M]} \left\{ r_\mu\left(F_j(t)\right) \right\}. \tag{4}$$

With the robustness metric defined in this way, $\rho_\mu(t)$ corresponds to the collective deviation from assumed circumstances (relevant ETC values) that the resource allocation can tolerate and still ensure that system performance will be acceptable (the actual makespan is within τ of the predicted).

Let T_e be the set of all mapping event times. The robustness value of the mapping is defined as the smallest robustness metric that occurs at any mapping event time in T_e. The primary objective of heuristics in this research is to maximize the robustness value, i.e.,

$$\text{maximize}\left(\min_{\forall t_e \in T_e} \rho_\mu(t_e) \right). \tag{5}$$

In addition to maximizing robustness, heuristics in this research must complete all T incoming tasks within an overall makespan constraint (α). Therefore, the goal of heuristics in this research is to dynamically map incoming tasks to machines such that the robustness value is maximized while completing all tasks within an overall makespan constraint (based on ETC values).

III. Heuristics

A. Overview

Five pseudo-batch mode heuristics were studied for this research. All of the heuristics used a common procedure to identify a set of feasible machines, where a machine is considered feasible if it can execute the task without violating the makespan constraint that is, for a task under consideration, a machine is considered feasible if that machine can satisfy the makespan constraint when the task is assigned to it. The subset of machines that are feasible for the task is referred to as the feasible set of machines. To avoid idle machines in the system caused by waiting for a heuristic to complete a mapping event, a task that is next in line to begin execution on any machine will not be considered for mapping at a mapping event. While it is still possible that a machine may become idle, it is highly unlikely for the assumptions in this research (the average execution time of a task is 120 seconds while the average execution time of a mapping event is less than 0.6 seconds)

B. Heuristic Descriptions

1) Minimum Completion Time-Minimum Completion Time (MinCT-MinCT): The MinCT-MinCT heuristic uses a variant of the Min-Min heuristic defined in [14]. For each mappable task, MinCT-MinCT identifies the set of feasible machines. From each task's set of feasible machines, the machine that minimizes completion time for the task is selected. If for any task there are no feasible machines, then the heuristic will fail. From these task/machine pairs, the pair with the minimum completion time is selected and the task is mapped onto its chosen machine. This procedure is repeated until all mappable tasks have been mapped. The procedure at each mapping event can be summarized as follows:

- i A task list is generated that includes all mappable tasks.
- ii For each task in the task list, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation").
- iii For each mappable task (ignoring other mappable tasks), find the feasible machine that minimizes completion time.
- iv From the above task/machine pairs select the pair that minimizes completion time.
- v Remove the task from the task list and map it onto the chosen machine.
- vi Update the machine available time.
- vii Repeat ii-vi until the task list is empty.

2) Maximum Robustness-Maximum Robustness (MaxR-MaxR): The MaxR-MaxR heuristic builds on the concept of the Max-Max heuristic defined in [14]. For each mappable task, MaxR-MaxR identifies the set of feasible machines. From each task's set of feasible machines, the machine that maximizes the robustness metric for the task is selected. If for any task there are no feasible machines then the heuristic will fail. From these task/machine pairs, the pair that maximizes the robustness metric is selected and that task is mapped onto its chosen machine. This procedure is repeated until all of the mappable tasks have been mapped. The procedure at each mapping event can be summarized as follows:

- i A task list is generated that includes all mappable tasks.
- ii For each task in the task list, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation").
- iii For each mappable task (ignoring other mappable tasks), find the feasible machine that maximizes the robustness radius.
- iv From the above task/machine pairs select the pair that maximizes the robustness radius.
- v Remove the task from the task list and map it onto the chosen machine.
- vi Update the machine available time.
- vii Repeat ii-vi until task list is empty.

3) Maximum Robustness-Minimum Completion Time (MaxR-MinCT): MaxR-MinCT builds on the concept of the Max-Min heuristic defined in [14]. For each mappable task, MaxR-MinCT identifies the set of feasible machines. From each task's set of feasible machines, the machine that minimizes completion time for the task is selected. If for any task there are no feasible machines, then the heuristic will fail. From these task/machine pairs, the pair that maximizes the robustness metric is selected and that task is mapped onto its chosen machine. This procedure is repeated until all of the mappable tasks have been mapped. The procedure at each mapping event can be summarized as follows:

- i A task list is generated that includes all mappable tasks.
- ii For each task in the task list, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation").
- iii For each mappable task (ignoring other mappable tasks), find the feasible machine that minimizes completion time.
- iv From the above task/machine pairs select the pair that maximizes the robustness radius.
- v Remove the task from the task list and map it onto the chosen machine.
- vi Update the machine available time.
- vii Repeat ii-vi until task list is empty.

4) Minimum Completion Time-Maximum Robustness (MinCT-MaxR): For each mappable task, MinCT-MaxR identifies the set of feasible machines. From each task's set of feasible machines, the machine that maximizes the robustness metric is selected. If for any task there are no feasible machines, then the heuristic will fail. From these task/machine pairs, the pair that minimizes completion time is selected and that task is mapped onto its chosen machine. This procedure is repeated until all of the mappable tasks have been mapped. The procedure at each mapping event can be summarized as follows:

i A task list is generated that includes all mappable tasks.
ii For each task in the task list, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation").
iii For each mappable task (ignoring other mappable tasks), find the feasible machine that maximizes the robustness radius.
iv From the above task/machine pairs select the pair that minimizes completion time.
v Remove the task from the task list and map it onto the chosen machine.
vi Update the machine available time.
vii Repeat ii-vi until task list is empty.

5) MaxMaxMinMin (MxMxMnMn): This heuristic makes use of two sub-heuristics to obtain a mapping. It uses a combination of Min-Min with a robustness constraint (to minimize makespan while maintaining the current robustness value) and Max-Max (based on robustness) to maximize robustness while still finishing all T tasks within the overall makespan constraint. The mapping procedure begins execution using the Min-Min heuristic with τ as the robustness level to be maintained—τ was chosen based on the upper bound discussion presented in Subsection III-D. The procedure at each mapping event can be summarized as follows:

i A task list is generated that includes all mappable tasks.
ii Min-Min component
   a) For each task in the task list, find the set of machines that satisfy the robustness level if the considered task is assigned to it. If the set is empty for any task, go to step iii.
   b) From the above set of machines, for each mappable task (ignoring other mappable tasks), find the feasible machine that minimizes the completion time.
   c) From the above task/machine pairs select the pair that minimizes completion time.
   d) Remove the task from the task list and map it onto its chosen machine.
   e) Update the machine available time.
   f) Repeat a-e until task list is empty, exit.
iii Max-Max component
   g) A task list is generated that includes all mappable tasks (any task mapped by Min-Min in this mapping event are remapped).
   h) For each task in the task list, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation")
   i) For each mappable task (ignoring other mappable tasks), find the feasible machine that maximizes the robustness radius.
   j) From the above task/machine pairs select the pair that maximizes the robustness radius.
   k) Remove the task from the task list and map it onto the chosen machine.
   l) Update the machine available time.
   m) Repeat h-l until task list is empty.
iv Update the robustness level to the new robustness value (the smallest robustness metric that has occurred).

C. Fine Tuning (FT)

A post-processing step, referred to as fine tuning (FT) was employed to improve the robustness value produced by a mapping. Fine tuning reorders tasks in the machine queues in ascending order of execution time on that machine (as done for a different problem environment in [31]), i.e., smaller tasks are placed in the front of the queues. This procedure is performed at each mapping event after executing one of the above heuristics. This procedure will not directly impact the overall finishing times of the machines, but does help in getting the smaller tasks out of the machine queues faster and thus helps reduce the numerator in equation 3, which correspondingly improves the robustness metric.

D. Upper Bound

Let the provided constant τ be the upper bound on robustness. To prove that robustness can be no higher than τ is to show that at least one machine will have at least one task assigned to it during the course of the simulation. When the first task is assigned to some machine in the system the robustness radius of that machine becomes τ. In equation 3, $\beta(t) - F_j(t)$ goes to zero for the makespan machine. Because the machine with the first and only task assigned to it is now the makespan defining machine, its robustness radius is now τ. The robustness radius of this machine defines the robustness metric for the system because it is the smallest of the robustness radii at this mapping event. Because the robustness value is defined as the smallest robustness metric over all mapping events, that value can be no greater than τ.

IV. SIMULATION SETUP

The simulated environment consists of $T = 1024$ independent tasks and $M = 8$ machines. This number of tasks and machines was chosen to present a significant mapping challenge for each heuristic and to prevent an exhaustive search for an optimal solution (however, the presented techniques can be applied to environments with different number of tasks and machines). As stated earlier, each task arrives dynamically and the arrival times are not known *a priori*. For this study, 200 different ETC matrices were generated, 100 with high task, high machine heterogeneity (HIHI) and 100 with low task, low machine heterogeneity (LOLO)([6]). All of the ETC matrices generated were inconsistent (i.e., machine A being faster than machine B for task 1 does not imply that machine A is faster than machine B for task 2) [6]. All ETC matrices were generated using the gamma distribution method presented in [4]. The arrival time of each incoming task was generated according to a Poisson distribution. The mean task inter-arrival rate was eight seconds.

In the gamma distribution method of [4], a mean task execution time and coefficient of variation (COV) are used to generate ETC matrices. In the HIHI heterogeneity case, the mean task execution time was set to 120 seconds and a COV of

0.9 was used for both task and machine heterogeneity. For the LOLO heterogeneity case, a mean task execution time of 120 seconds was used with a COV of 0.3 for both task and machine heterogeneity. The value of τ chosen for this study was 120 seconds. The performance of each heuristic was studied across all 200 different simulation trials (ETC matrices).

V. Results

In Figures 1 and 2, the average robustness value (over all mapping events) for each heuristic is plotted with their 95% confidence intervals. The average makespan results for each of the heuristics (with 95% confidence intervals) are shown in Figures 3 and 4. The value of α was set to 14000 for HIHI and 12500 for LOLO. In Figures 1, 2, 3, and 4 the number of failed trials (out of 100) is indicated above the results for each heuristic, i.e., the number of trials for which the heuristic was unable to successfully find a mapping for every task given the overall makespan constraint. The effects of fine tuning on each of the heuristics are also plotted in Figures 1–4 including the 95% confidence intervals.

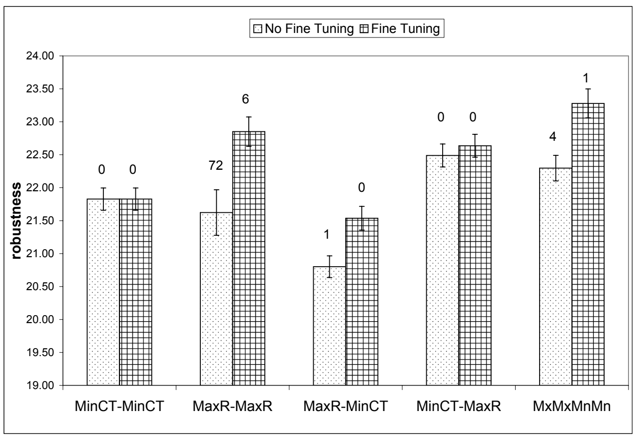

Fig. 1. Average robustness value (over all mapping events) for the LOLO case with $\alpha = 12500$.

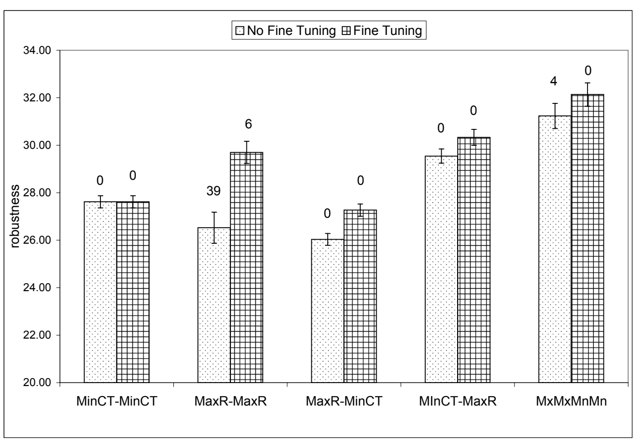

Fig. 2. Average robustness value (over all mapping events) for the HIHI case with $\alpha = 14000$.

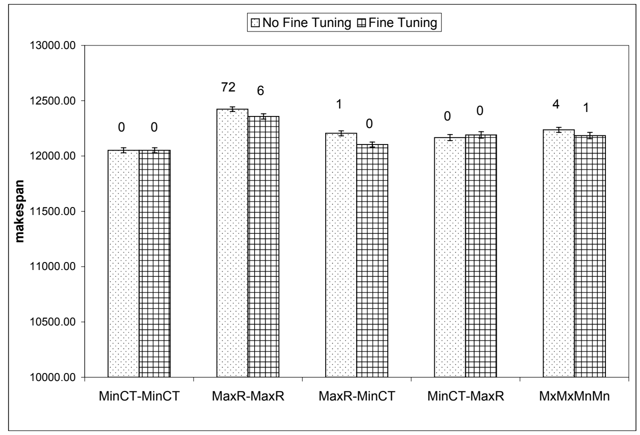

Fig. 3. Average makespan for the LOLO case with $\alpha = 12500$.

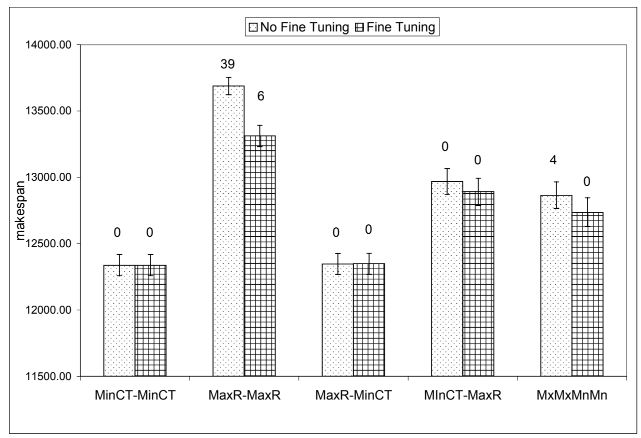

Fig. 4. Average makespan for the HIHI case with $\alpha = 14000$.

The average execution time of each heuristic over all mapping events (on a typical unloaded 3GHz Intel Pentium 4 desktop machine) in all 200 trials is shown in Table I. Recall that the heuristics operate in a pseudo-batch mode, therefore, the times in Table I are the average time for each heuristic to map an entire batch of tasks.

As can be seen from Figures 1 and 2, MxMxMnMn with fine tuning gives the best robustness result for both the HIHI and LOLO cases (although there is one failure).

TABLE I
AVERAGE EXECUTION TIMES, IN SECONDS, OF A MAPPING EVENT FOR THE PROPOSED HEURISTICS.

heuristic	average execution time (sec.)
MinCT-MinCT	0.023
MaxR-MinCT	0.028
MinCT-MaxR	0.028
MaxR-MaxR	0.0563
MxMxMnMn	0.0457

The good performance of MxMxMnMn can be attributed to the fact that the maintainable robustness value is by definition monotonically decreasing, and its approach tries to minimize makespan (using Min-Min) while maintaining the current robustness value. If that is not possible it instead maximizes robustness using Max-Max–attempting to minimize the degradation in the robustness value.

The high number of failed trials for MaxR-MaxR for both the HIHI and LOLO cases can be attributed to the fact that the heuristic tries to maximize the robustness metric at all mapping events, but in doing so neglects the corresponding increase in machine finishing times. For example, consider the following two machine system with a current robustness value of 60 and machine queues with the task execution times as shown,

m_1:	$t_1(10)$	$t_3(10)$
m_2:	$t_2(50)$	

Assume that a new task t_4 arrives with execution times of 10 and 50 time units on machines m_1 and m_2, respectively. The MaxR-MaxR heuristic will map task t_4 to machine m_2, which increases makespan because assigning t_4 to machine m_1 would decrease the robustness metric. However, mapping t_4 to m_1 would give a new robustness metric of 80.8 that is still greater than the current robustness value of 60.

Although, MinCT-MinCT is able to achieve one of the best makespan (Figures 3 and 4) for both the HIHI and LOLO cases, its robustness value is not one of the best, which confirms the fact that just minimizing the finishing times of the machines does not guarantee a higher value of robustness.

For both the HIHI and LOLO cases, MinCT-MaxR performed relatively better than MaxR-MinCT in terms of robustness. This can be explained in terms of the first stage choice of machines for this pair of two-stage greedy heuristics. MinCT-MaxR places more emphasis on directly optimizing the primary objective of maximizing the robustness value as opposed to minimizing makespan. By minimizing completion time in the second stage, MinCT-MaxR is able to stay within the overall makespan constraint while still maximizing robustness. This is evident from zero failures that occurred for MinCT-MaxR in both the LOLO and HIHI cases.

The process of fine tuning did improve the results of the heuristics, though not substantially (less than 12% for the best HIHI case and less than 5% for the best LOLO case). Further, it is possible that fine tuning when used with MxMxMnMn can cause some trials to fail to meet the makespan constraint. This occurs because fine tuning attempts to reduce the number of tasks in the machine queues by moving small tasks up in the queues. Thus, it is possible for the heuristic to maintain a higher robustness value over its execution, but at certain mapping events when the Min-Min component of the heuristic tries to map a task using a higher robustness constraint, it is likely that it will not choose the minimum completion time machine for the task because it is not feasible, which results in a higher finishing time. For example, consider a two machine system with the following machine queues,

m_1:	$t_1(150)$	
m_2:	$t_2(30)$	$t_3(80)$

Assume that a new task t_4 arrives with execution times of 80 and 20 on machines m_1 and m_2, respectively. If MxMxMnMn maps this task using the Min-Min component with a robustness level of $\tau/\sqrt{2}$, the mapping would be:

m_1:	$t_1(150)$	$t_4(80)$
m_2:	$t_2(30)$	$t_3(80)$

But if MxMxMnMn uses the Min-Min component with a robustness level of $\tau/2$, the mapping would be:

m_1:	$t_1(150)$		
m_2:	$t_2(30)$	$t_3(80)$	$t_4(20)$

Finally, because MxMxMnMn uses a Max-Max heuristic to maximize robustness it is prone to the same issues discussed previously for the MaxR-MaxR heuristic.

The proposed pseudo-batch mode heuristics can allow some tasks to be starved of machines. That is, some tasks may be continually remapped at each successive mapping event without actually being executed. In this environment, once the mappable tasks have been remapped by a pseudo-batch mode mapper they may be reordered, according to their arrival order, in the input queues of their respective machines. Reordering tasks according to their arrival order ensures that task starvation does not occur. However, this may impact the robustness metric at later mapping events, so this is not done given the current performance metric.

VI. Related Work

The research presented in this paper was designed using the four step FePIA procedure described in [3]. A number of papers in the literature have studied robustness in distributed systems (e.g., [5], [8], [25], [27]).

The research in [5] considers rescheduling of operations with release dates using multiple resources when disruptions prevent the use of a preplanned schedule. The overall strategy is to follow a preplanned schedule until a disruption occurs. After a disruption, part of the schedule is reconstructed to match up with the pre-planned schedule at some future time. Our work considers a slightly different environment where task arrivals are not known in advance. Consequently, in our work it was not possible to generate a preplanned schedule.

The research in [8] considers a single machine scheduling environment where processing times of individual jobs are uncertain. Given the probabilistic information about processing times for each job, the authors in [8] determine a normal distribution that approximates the flow time associated with a given schedule. The risk value for a schedule is calculated by using the approximate distribution of flow time (i.e., the sum of the completion times of all jobs). The robustness of a schedule is then given by one minus the risk of achieving sub-standard flow time performance. In our work, no such stochastic specification of the uncertainties is assumed.

The study in [25] defines a robust schedule in terms of identifying a Partial Order Schedule (POS). A POS is defined as a set of solutions for the scheduling problem that can be compactly represented within a temporal graph. However, the study considers the Resource Constrained Project Scheduling Problem with minimum and maximum time lags, (RCPSP/max), as a reference, which is a different problem domain from the environment considered here.

In [27], the robustness is derived using the same FePIA procedure used here. However the environment considered is static (off-line), as opposed to the dynamic (on-line) environment in this research. The robustness metric and heuristics employed in a dynamic environment are substantially different from those employed in [27].

VII. Conclusions

Five different pseudo-batch mode heuristics were designed to address a dynamic resource allocation problem. A process of fine tuning was also adapted to the problem to maximize the robustness of a mapping in the proposed environment. Each heuristic was analyzed for its ability to maximize robustness given a hard constraint on overall makespan. Of the proposed heuristics, the MxMxMnMn heuristic with fine tuning demonstrated good potential in the simulation environment and should be considered further. This paper also presented an upper bound on the attainable robustness for the considered resource management problem.

VIII. Acknowledgment

The authors thank Luis Briceno for his 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," *Advances in Computers Volume 63: Parallel, Distributed, and Pervasive Computing*, A. R. Hurson, ed., Elsevier, Amsterdam, The Netherlands, 2005, pp. 91-128.

[2] 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*, Special Issue on Parallel Numerical Algorithms on Faster Computers, Vol. 5, No. 4, Dec. 2002.

[3] 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.

[4] 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 50^{th} Anniversary Issue, Vol. 3, No. 3, Nov. 2000, pp. 195-207 (invited).

[5] J. Bean, J. Birge, J. Mittenthal, C. Noon, "Matchup scheduling with multiple resources, release dates and disruptions," *Journal of the Operations Research Society of America*, Vol. 39, No. 3, June. 1991, pp. 470-483.

[6] 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.

[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. Carrilo, "β-Robust scheduling for single-machine systems with uncertain processing times," *IIE Transactions*, Vol. 29, No. 11, Nov. 1997, pp. 977-985.

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

[10] 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.

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

[12] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13-17.

[13] A. Ghafoor and J. Yang, "A distributed heterogeneous supercomputing management system," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 78-86.

[14] 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.

[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] A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18-27.

[17] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," 4^{th} *IEEE Heterogeneous Computing Workshop (HCW '95)*, 1995, pp. 30-34.

[18] 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.

[19] P. Luh, X. Zhao, Y. Wang, and L. Thakur, "Lagrangian relaxation neural networks for job shop scheduling," *IEEE Transactions on Robotics and Automation*, Vol. 16, No. 1, Feb. 2000, pp. 78-88.

[20] 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.

[21] 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.

[22] A. Mehta, J. Smith, H. J. Siegel, A. A. Maciejewski, A. Jayaseelan, B. Ye, "Dynamic resource management heuristics for minimizing makespan while maintaining an acceptable level of robustness in an uncertain environment," 12^{th} *International Conference on Parallel and Distributed Systems*, accepted, to appear.

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

[24] V. K. Naik, S. Sivasubramanian, D. Bantz, and S. Krishnan, "Harmony: A desktop grid for delivering enterprise computations," 4^{th}*International Workshop on Grid Computing (GRID 03)*, Nov. 2003.

[25] N. Policella, *Scheduling with uncertainty, A proactive approach using partial order schedules*, PhD thesis, Dipartimento di Informatica e Sistemistica "Antonio Ruberti" Universit'a degli Studi di Roma "La Sapienza," 2005.

[26] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," 5^{th} *IEEE Heterogeneous Computing Workshop (HCW '96)*, 1996, pp. 86-97.

[27] 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.

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

[29] D. Xu, K. Nahrstedt, and D. Wichadakul, "QoS and contention-aware multi-resource reservation," *Cluster Computing*, Vol. 4, No. 2, Apr. 2001, pp. 95-107.

[30] J. Yang, I. Ahmad, and A. Ghafoor, "Estimation of execution times on heterogeneous supercomputer architectures," *International Conference on Parallel Processing*, Aug. 1993, pp. I-219-I-226.

[31] V. Yarmolenko, J. Duato, D. K. Panda, P. Sadayappan, "Characterization and enhancement of dynamic mapping heuristics for heterogeneous systems," *International Conference on Parallel Processing Workshops (ICPPW 00)*, Aug. 2000, pp. 437-444.