Dynamic Resource Management Heuristics for Minimizing Makespan while Maintaining an Acceptable Level of Robustness in an Uncertain Environment

Ashish M. Mehta[1], Jay Smith[3], H. J. Siegel[1,2], Anthony A. Maciejewski[1],

Arun Jayaseelan[1], and Bin Ye[1]

Colorado State University

[1]Department of Electrical & Computer Engineering

[2]Department of Computer Science

Fort Collins, CO 80523-1373

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

binye@simla.colostate.edu

[3]IBM

6300 Diagonal Highway

Boulder, CO 80301

bigfun@us.ibm.com

January 2006

Submitted to the 12th International Conference on Parallel and Distributed Systems

(ICPADS 2006)

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), by the DARPA Information Exploitation Office under contract No. NBCHC030137, and by the Colorado State University George T. Abell Endowment. Needs to be approved for public release; distribution unlimited.

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. An important research problem in resource management is how to determine a resource allocation and scheduling of tasks to machines that optimizes a system performance feature while delivering acceptable level of robustness. Makespan (defined as the time required to complete all tasks in a resource allocation) is often the performance parameter that is optimized in such systems. This paper presents a robustness metric for dynamic resource allocations where task execution times are uncertain. The goal of this research is to develop heuristics capable of dynamically mapping tasks to machines such that makespan is minimized and a specified level of robustness is maintained. This research proposes, evaluates, and compares ten different dynamic heuristics for their ability to maintain or maximize the proposed dynamic robustness metric in an uncertain environment. In addition, the makespan results of the proposed heuristics are compared to a lower bound.

Keywords: resource management, robustness, dynamic mapping, resource allocation, makespan.

1. Introduction and Problem Statement

Heterogeneous parallel and distributed computing is the coordinated use of various compute resources of different capabilities to optimize certain system performance features. An important research problem (i.e., resource management) is how to determine a resource allocation and scheduling of tasks to machines (i.e., a mapping) that optimizes a system performance feature while maintaining an acceptable level of quality of service. This research focuses on a dynamic mapping environment where task arrival times are not known *a priori*. A mapping environment is considered dynamic when tasks are mapped as they arrive, e.g., in an on-line fashion [MaA99]. The general problem of optimally mapping tasks to machines (resource management) in heterogeneous parallel and distributed computing environments has been shown in general to be NP-complete (e.g., [Cof76, Fer89, IbK77]). Thus, the development of heuristic techniques to find a near-optimal solution for the mapping problem is an active area of research (e.g., [AlK02, BaS01, BaV01, BrS01, Esh96, FoK99, LeW94, MaA99, MiF00, WuS00]).

Dynamic mapping heuristics can be grouped into two categories: immediate mode and batch mode [MaA99]. In immediate mode, when a task arrives (i.e., a mapping event) it is immediately mapped to some machine in the suite for execution. In batch mode, tasks are accumulated until a specified condition is satisfied (e.g., a certain number of tasks have accumulated, or some amount of time has elapsed); whereupon the entire batch of accumulated tasks and the previously enqueued but not executing tasks are 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) and consisting of all tasks in the parallel and distributed system that have not yet begun execution on some machine. Both immediate mode and pseudo-batch mode heuristics were considered for this research.

The pseudo-batch mode heuristics can allow some tasks to be starved of machines. Some tasks may be remapped at each successive mapping event without actually being executed, i.e., the task is starved for a machine. In this environment, once the mappable tasks have been re-mapped by a pseudo-batch mode mapper they may be reordered, according to their arrival order, in the input queues of their respective machines. This is possible because tasks in this environment are independent, i.e., the environment does not mandate their order of execution. Reordering tasks according to their arrival order ensures that task starvation does not occur.

Heterogeneous parallel and distributed systems may operate in an environment where certain system performance features degrade due to unpredictable circumstances and inaccuracies in estimated system parameters. Robustness is defined as the degree to which a system can function correctly in the presence of parameters different from those assumed [AlM04]. For a given set of tasks, the makespan is defined as the completion time for the entire set of tasks. For this research, makespan is required to be robust against errors in the estimated execution time of each task. For a given application domain, the estimated time to compute (ETC) each task *i* on each machine *j* is assumed known, denoted ETC(*i, j).* However, these estimates may deviate from the *actual* computation times; e.g., the actual times may depend on characteristics of the input data to be processed. The tasks considered in this research are taken from a frequently executed predefined set, such as exists in a lab or business environment. This research focuses on determining a dynamic mapping for a set of tasks that minimizes the predicted makespan (using the provided ETC values) while still being able to tolerate a quantifiable amount of variation in the ETC values of the mapped tasks. Hence, the goal is to obtain a mapping that has the minimum makespan and can still guarantee a certain level of robustness at each mapping event.

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 executes 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. Let $T(t)$ be the set of tasks either currently executing or pending execution on any machine at time t ($T(t)$ does not include the tasks that have already completed execution). Let $F_j(t)$ be the predicted finishing time of machine j for a given resource allocation based on the given ETC values. Let $MQ_j(t)$ denote the subset of *T*(*t*) that has been mapped to machine *j* and let $scet_j(t)$ denote the starting time of the currently executing task on machine *j*. Mathematically, given some machine *j*

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

Let $\beta(t)$ denote the maximum of the finishing times $F_j(t)$ at each time *t* over all machines. This is the predicted makespan at time *t*. Mathematically,

$$\beta(t) = \max_{\forall j \in M} \left\{ F_j(t) \right\}.$$

The robustness metric for this work has been derived using the procedure defined in [AlM04]. 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 $r_\mu\left(F_j(t)\right)$ of machine *j* can be quantitatively defined as the maximum collective Euclidean error in the estimated task computation times that can occur and still have the *actual* finishing times be within τ plus the *predicted* makespan. Mathematically, building on a result in [AIM04],

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

The robustness metric $\rho_\mu(t)$ for a given mapping μ is simply the minimum of the robustness radii over all machines [AIM04]. Mathematically,

$$\rho_\mu(t) = \min_{\forall j \in M} \left\{ r_\mu\left(F_j(t)\right) \right\}.$$

With the robustness metric defined in this way, $\rho_\mu(t)$ corresponds to the collective deviation from assumed circumstances that the resource allocation can tolerate and still ensure that system performance will be acceptable.

This work is directly applicable to resource allocation in enterprise systems designed to support transactional computations sensitive to response time constraints, e.g., time sensitive business processes [NaS03]. 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. Further, the heuristics of Section 2 will use the dynamic robustness metric as a constraint during mapping.

To define the dynamic robustness metric as a constraint let $\underline{\alpha}$ be the <u>minimum</u> <u>acceptable</u> <u>robustness</u> of a resource allocation at any mapping event; i.e., the constraint requires that the robustness metric at each mapping event be at least α. Thus, the goal of the heuristics in this research is to dynamically map incoming tasks to machines such that the total makespan is minimized, while maintaining a robustness of at least α, i.e., $\rho_{\mu}(t) \geq \alpha$ for all mapping events. The larger α is, the more robust the resource allocation is.

The contributions of this paper include:

- a model for quantifying dynamic robustness in this environment,
- heuristics for solving the above resource management problem, and
- simulation results for the proposed heuristics.

The remainder of the paper is organized as follows. Section 2 briefly discusses the heuristics studied in this research including the definition of a lower bound on the total makespan of the mapping problem. Section 3 outlines the simulation setup. The simulation results are presented and discussed in Section 4. The related work is considered in Section 5.

2. Heuristics

2.1. Overview

Five immediate mode and five pseudo-batch mode heuristics were studied for this research. For the task under consideration, a feasible machine is defined to be a machine that will satisfy the robustness constraint if the considered task is assigned to it. This subset of machines is referred to as the feasible set of machines.

2.2. Immediate Mode Heuristics

The following is a brief description of the immediate mode heuristics. Pseudocode for each heuristic is also given.

Feasible robustness minimum execution time (FRMET**)** (Figure 1) is based on the MET concept in [BrS01, MaA99, YAD00]. For each incoming task, FRMET first identifies the feasible set of machines. From the feasible set of machines the incoming task is assigned to its minimum *execution* time machine.

Feasible robustness minimum completion time (FRMCT**)** (Figure 2) is based on the MCT concept in [BrS01, MaA99, YAD00]. For each incoming task, FRMCT first identifies the feasible set of machines for the incoming task. From the feasible set of machines the incoming task is assigned to its minimum *completion* time machine.

Feasible robustness k-percent best (FRKPB) (Figure 3) is based on the KPB concept in [KiS03, MaA99]. FRKPB first finds the feasible set of machines for the newly arrived task. From this set, FRKPB identifies the *k*-percent that have the smallest *execution* times for the task. The task is then assigned to the machine in the set with the minimum *completion* time for the task. For a given α the value of *k* was varied between 0 and 100, in steps of 12.5, for sample training data to determine the value that provided the minimum makespan. A value of 50 was found to give the best results.

Feasible robustness switching (FRSW) (Figure 4) is based on the SW concept in [KiS03, MaA99]. As applied in this research, FRSW combines aspects of both the FRMET and the FRMCT heuristics. A load balance ratio is defined to be the ratio of the minimum number of tasks enqueued on any machine to the maximum number of tasks enqueued on any machine. FRSW then switches between FRMET and FRMCT based on the value of the load balance ratio. The heuristic starts by mapping tasks using FRMCT. When the ratio raises above a defined set point $\underline{T_{high}}$ FRSW switches to the FRMET heuristic. When the ratio falls below a defined set point $\underline{T_{low}}$ FRSW switches to the FRMCT heuristic. The values for the switching set points were determined experimentally using sample training data.

Maximum robustness (MaxRobust) (Figure 5) has been implemented for comparison only, trying to greedily maximize robustness without considering makespan. MaxRobust calculates the robustness radius of each machine for the newly arrived task, assigning the task to the machine with the maximum robustness radius.

2.3. Pseudo-Batch Heuristics

The pseudo-batch mode heuristics implement two sub-heuristics, one to map the task as it arrives, and a second to remap pending tasks. For the pseudo-batch mode heuristics, the initial mapping is performed by the previously described FRMCT heuristic (except for the MRMR heuristic). The remapping heuristics each operate on a set of mappable tasks; a mappable task is defined as any task pending execution that is not next in line to begin execution(to avoid idle machines). The following is a brief description of the pseudo-batch mode re-mapping heuristics.

Feasible robustness minimum completion time-minimum completion time (FMCTMCT) (Figure 6) uses a variant of Min-Min defined in [IbK77]. For each mappable task, FMCTMCT finds the feasible set of machines, then from this set determines the machine that provides the minimum completion time for the task. From these task/machine pairs, the pair that gives the overall minimum completion time is selected and that task is mapped onto that machine. This procedure is repeated until all of the mappable tasks have been remapped.

Feasible robustness maximum robustness-minimum completion time (FMRMCT) (Figure 7) builds on concept of the Max-Min heuristic [IbK77]. For each mappable task, FMRMCT first identifies the feasible set of machines, then from this set determines the machine that provides the minimum completion time. From these task/machine pairs, the pair that provides the maximum robustness radius is selected and the task is assigned to that machine. This procedure is repeated until all of the mappable tasks have been remapped.

Feasible minimum completion time-maximum robustness (FMCTMR) (Figure 8) For each mappable task, FMCTMR first identifies the feasible set of machines, then from this set determines the machine with the maximum robustness radius. From these task/machine pairs, the pair that provides the minimum completion time is selected and the task is mapped to that machine. This procedure is repeated until all of the mappable tasks have been remapped.

Maximum weighted sum-maximum weighted sum (MWMW) (Figure 9) builds on a concept in [ShC06]. It combines the Lagrangian heuristic technique [CaS04, LuZ00] for deriving an objective function with the concept of Min-Min heuristic [IbK77] here to simultaneously minimize makespan and maximize robustness. For each mappable task, the feasible set of machines is identified and the machine in this set that gives the maximum value of the objective function (defined below) is determined. From this collection of task/machine pairs, the pair that provides the maximum value of the objective function is selected and the corresponding assignment is made. This procedure is repeated until all of the mappable tasks have been remapped.

When considering assigning a task *i* to machine *j*, let $\underline{F_j^{'}(t)} = F_j(t) + ETC(i, j).$ for all tasks currently in the machine queue and the task currently under consideration. Let $\underline{\beta^{'}(t)}$ be maximum of the finishing times $F_j^{'}(t)$ at time *t* for all machines. Let $\underline{r_\mu^{'}\left(F_j^{'}(t)\right)}$ be the robustness radius for machine *j*. Let $\underline{maxrob(t)}$ be the maximum of the robustness radii at time *t*. Given $\underline{\eta}$, an experimentally determined constant using training data, the objective function $\underline{s(j,t)}$ for the Simplified Lagrangian is defined as

$$s(j,t) = \eta\left(1 - \frac{F_j^{'}(t)}{\beta^{'}(t)}\right) + (1-\eta)\left(\frac{r_\mu{'}\left(F_j{'}(t)\right)}{maxrob(t)}\right).$$

Maximum robustness-maximum robustness (MRMR) (Figure 10) is provided here for comparison only as it optimizes robustness without considering makespan. As a task arrives it is initially mapped using the MaxRobust heuristic. Task remapping is performed by a variant of the Max-Max [IbK77] heuristic. For each mappable task, the machine that provides the maximum robustness radius is determined. From these task/machine pairs, the pair that provides the maximum overall robustness radius is selected and the task is mapped to that machine. This procedure is then repeated until all of the mappable tasks have been remapped.

2.4. Lower Bound

A lower bound on makespan for the described system can be found by identifying the task whose arrival time plus minimum execution time on any machine is the greatest. More formally, given the entire set of tasks *T* where each task *i* has an arrival time of $\underline{arv(i)}$, the lower bound is given by

$$\text{lower bound} = \max_{\forall i \in T}\left(arv(i) + \min_{\forall j \in M} ETC(i,j)\right).$$

This is a lower bound on makespan. Thus, no heuristic can achieve a smaller makespan. However, it is possible that this lower bound is not achievable even by an optimal mapping.

3. 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, our techniques can be applied to different numbers of tasks and machines). As stated earlier, each task arrives dynamically and arrival times are not known *a priori*. For this study, 100 different ETC matrices were generated, 50 with high task heterogeneity and high machine heterogeneity (HIHI) and 50 with low task heterogeneity and low machine heterogeneity (LOLO) ([BrS01]). All of the ETC matrices generated were inconsistent (i.e., machine A being faster than machine B for task 1 does not imply machine A is faster than machine B for task 2) [BrS01]. All ETC matrices were generated using the gamma distribution method presented in [AlS00]. The arrival time of each task was generated according to a Poisson distribution.

In the gamma distribution method of [AlS00], a mean task execution time and coefficient of variation (COV) are used to generate the ETC matrices. In the high-high case, the mean task execution time was set to 100 seconds and a COV of 0.9 (task heterogeneity) was used to calculate the values for all elements in a vector of task execution times. Then using the ith element of the vector as the mean and a COV of 0.9 (machine heterogeneity), the ETC values for task i on all machines are calculated. The low-low heterogeneity case uses a mean task execution time of 100 seconds and a COV of 0.3 for task heterogeneity and a COV of 0.3 for machine heterogeneity.

The value of τ chosen for this study was 120 seconds. The performance of each heuristic was studied across all 100 different trials (ETC matrices).

4. Results

In Figures 11 and 12, the average makespan results (with 95% confidence interval bars) are plotted, along with a lower bound on makespan, for the immediate mode and pseudo-batch mode heuristics, respectively. Each of the heuristics was simulated using multiple values for the robustness constraint α. For each α, the performance of the heuristics was observed over 100 trials—50 for each HIHI and LOLO heterogeneity trials. In Figure 11, the number of failed trials (out of 50) is indicated above the makespan 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 robustness constraint α. Unlike the immediate mode heuristics, the pseudo-batch mode heuristics did not fail for any of the considered cases. The average execution time of each heuristic over all mapping events in all 100 trials is shown in Table 1. For the immediate mode heuristics, this is the average time for a heuristic to map an incoming task. For the pseudo-batch mode heuristics, this is the average time for a heuristic to map an entire batch of tasks.

For FRSW, for HIHI T_{low} was set to 0.6 and T_{high} was set to 0.9, and for LOLO T_{low} was set to 0.3 and T_{high} was set to 0.6. The MWMW heuristic used a value of $\eta = 0.7$ for HIHI and $\eta = 0.6$ for LOLO.

For the immediate mode heuristics, FRMET performed the best for HIHI, and FRMET and FRSW performed the best for LOLO. The immediate mode FRMET heuristic for both HIHI and LOLO heterogeneity performed better than anticipated given prior studies including a minimum execution time (MET) heuristic in other environment (that do no involve robustness and had different arrival rates and ETC matrices). It has been shown, in general, that the minimum execution time heuristic is not a good choice for minimizing makespan for both the static and dynamic environments [BrS01, MaA99], because it ignores machine loads and machine available times when making a mapping decision. The establishment of a feasible set of machines by the FRMET heuristic indirectly balances the incoming task load across all of the machines. Table 2 shows the maximum and average number of mapping events (out of a possible 1024) over successful trials (out of 50) for which the MET machine was not feasible. That is, the table values were calculated based on only the subset of the 50 trials for which FRMET could determine a mapping that met the constraint. For each of these trials, there were 1024 mapping events. The FRMET heuristic is able to achieve a better makespan than any of the heuristics studied and can even maintain a higher robustness constraint than MaxRobust heuristic for LOLO heterogeneity. For the HIHI case all the heuristics (except MaxRobust) failed for at least 4% (20% on average) of the trials (out of 50) for the robustness constraint achieved by MaxRobust heuristic. An interesting observation was that the FRMCT heuristic was able to achieve a robustness constraint of α=27 for the 50 trials used in this study, but only for 48 trials for α=26 (for HIHI heterogeneity). This could be attributed to the volatile nature of the greedy heuristics. The looser robustness constraint (α=26) allowed for a paring of task to machine that was disallowed for a tighter robustness constraint (α=27). That is, the early greedy selection proved to be a poor decision because it ultimately led to a mapping failure. Among the pseudo-batch mode heuristics, for the HIHI heterogeneity trials, FMRMCT performed the best on average, while FMCTMCT gave comparable results. For the LOLO heterogeneity trials, all of the heuristics performed comparably. For the robustness constraint achieved by MRMR heuristic (i.e., $\alpha = 36.18$ for HIHI and $\alpha = 28.28$ for LOLO) all the heuristics failed for at least 4% (10% on average) of the trials (out of 50).

5. Related work

The work presented in this paper was built on the four step FePIA procedure described in [AIM04]. A number of papers in the literature have studied the issue of robustness in distributed systems (e.g., [BeB91, BoHo4, DaC97, DoK95, Gho96, Pol05, SuS05]). A brief comparison of the studies is presented below.

The research in [BeB91] considers rescheduling of operations with release date and multiple resources when disruptions prevent the use of a preplanned schedule. The overall strategy is to follow a preschedule until a disruption occurs. After a disruption, part of the schedule is reconstructed to match up with the preschedule at some future time. Our work considers a dynamic environment and thus there will be no preplanned schedule, and hence [BeB91] is not usable in our work.

The research in [BoH04] proposes two heuristics, Dynamic Adaptive Random scheduler (*DARS*) and a new Load Balancing algorithm to dynamically schedule dependent tasks with imprecise execution time estimates. This research, unlike in [BoH04], considers independent set of tasks and is thus different form the above work.

The research in [DaC97] considers a single machine scheduling environment where the processing times of individual jobs are uncertain. Given the probabilistic information about processing time for each job, the authors in [DaC97] determine the normal distribution that approximates the flow time associated with a given schedule. The risk value is calculated by using the approximate distribution of flow time (i.e., sum of *completion* times of all jobs). The robustness of a given schedule is then given by 1 minus the risk of achieving substandard flow time performance. In our work, no such stochastic specification of the uncertainties is assumed. Furthermore, our environment involves multiple machines.

The research in [DoK95] considers reactive scheduling to unexpected events that may cause a performance constraint violation in a shop floor environment. They define repair steps so that the new performance would either improve or remain unchanged. Our work explores robust resource allocation techniques to maximize the cumulative allowable errors in ETCs so that the specified performance is guaranteed in a heterogeneous parallel or distributed computing environment; thus, our problem differs in many ways from scheduling machines in a shop.

The study in [Gho96] explores slack-based techniques to achieve robust schedules in a job-shop environment. The main idea is to provide each task with extra time (defined as slack) to execute so that it can tolerate some level of uncertainty without having to relocate. It uses slack as a measure of robustness which is simpler and different from our measure.

The study in [Pol05] 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 different problem domain from the environment considered here.

In [SuS05], the robustness is derived using the same FePIA procedure used here. However the environment considered is static (off-line), as opposed to dynamic (on-line) in our work. Hence, the robustness metrics and heuristics employed differ.

6. Summary

This research establishes a method for defining the robustness of a resource allocation in a dynamic environment. Ten different heuristics were designed, developed, and simulated for the presented parallel and distributed environment. The results of the heuristics were compared with a theoretical lower bound. The immediate mode heuristics described here can be used when the individual guarantee for the submitted jobs is to be maintained (as there is no reordering of the submitted jobs), while the pseudo-batch heuristics can be used when the overall system performance is of importance. Future work may include designing heuristics capable of maximizing robustness, but with makespan as a constraint.

Acknowledgment: The authors thank Jennifer Hale for her valuable comments.

References

[AlK02] 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.

[AlM04] 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.

[AlS00] 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).

[BaS01] 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), *15th International Parallel and Distributed Processing Symposium* (*IPDPS 2001*), Apr. 2001.

[BaV01] I. Banicescu and V. Velusamy, "Performance of scheduling scientific applications with adaptive weighted factoring," 10th IEEE Heterogeneous Computing Workshop (HCW 2001), *15th International Parallel and Distributed Processing Symposium* (*IPDPS 2001*), Apr. 2001.

[BeB91] 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.

[BoH04] W.F. Boyer and G.S. Hura, "Dynamic scheduling in distributed heterogeneous systems with dependent tasks and imprecise execution time estimates," *16th IASTED*

International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), Nov. 2004.

[BrS01] 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.

[CaS04] R. Castain, W. W. Saylor, and H. J. Siegel, "Application of lagrangian receding horizon techniques to resource management in ad-hoc grid environments,"*13th Heterogeneous Computing Workshop (HCW 2004)*, in the proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), Apr. 2004.

[Cof76] E. G. Coffman, Jr. ed., *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, NY, 1976.

[DaC97] 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.

[DoK95] 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.

[Esh96] M. M. Eshaghian, ed., *Heterogeneous Computing*. Norwood, MA, Artech House, 1996.

[Fer89] 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.

[FoK99] I. Foster and C. Kesselman, eds., *The Grid: Blueprint for a New Computing Infrastructure*, San Fransisco, CA, Morgan Kaufmann, 1999.

[Gho96] S. Ghosh, *Guaranteeing Fault-tolerance through Scheduling in Real-time Systems*, PhD thesis, Faculty of Arts and Sciences, University of Pittsburgh, 1996.

[IbK77] 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.

[KiS03] J. -K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S. S. Yellampalli, "Dynamic mapping in a heterogeneous environment with tasks having priorities and multiple deadlines," *12th Heterogeneous Computing Workshop (HCW 2003),* in the proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), Apr. 2003.

[LeW94] 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.

[LuZ00] 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.

[MaA99] 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.

[MiF00] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*, New York, NY, Springer-Verlag, 2000.

[NaS03] Naik, V. K., Sivasubramanian, S., Bantz, D., and Krishnan, S., "Harmony: A desktop grid for delivering enterprise computations," *Fourth International Workshop on Grid Computing (GRID '03)*, Nov. 2003.

[Pol05] 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.

[ShC06] S. Shivle, H. J. Siegel, A. A. Maciejewski, P. Sugavanam, T. Banka, R. Castain, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, and J. Velazco, "Static allocation of resources to communicating subtasks in a heterogeneous ad hoc grid environment," *Journal of Parallel and Distributed Computing, Special Issue on Algorithms for Wireless and Ad-hoc Networks*, accepted, to appear.

[SuS05] 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.

[WuS00] 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.

[YaD00] 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.

```
1. for the new incoming task find the feasible set of machines
2. from the above set, find the minimum execution time machine
3. assign the task to the machine
4. repeat steps 1-3 until all $T$ incoming tasks are mapped
```

Figure 1: Pseudocode for FRMET heuristic.

```
1. for the new incoming task find the feasible set of machines
2. from the above set find the minimum completion time machine
3. assign the task to the machine
4. repeat steps 1-3 until all $T$ incoming tasks are mapped
```

Figure 2: Pseudocode for FRMCT heuristic.

```
1. for the new incoming task find the feasible set of machines
2. from the above set, find the top m = 4 machines based on execution time
3. from the above find the minimum completion time machine
4. assign the task to the machine
5. repeat steps 1-3 until all $T$ incoming tasks are mapped
```

Figure 3: Pseudocode for FRKPB heuristic.

```
1. for the new incoming task find the feasible set of machines
2. calculate the load balance ratio (LBR)
3. Initial mapping heuristic – FeasibleMCT
    If LBR > $T_{high}$ map using FeasibleMET
    If LBR < $T_{low}$ map using FeasibleMCT
    If $T_{low} \leq$ LBR $\leq T_{high}$ map using previous mapping heuristic
4. repeat steps 1-3 until all $T$ incoming tasks are mapped
```

Figure 4: Pseudocode for FRSW heuristic.

```
1. for the new incoming task find the robustness radius for each machine, considering the
   previous assignments

2. assign task to maximum robustness radius machine

3. repeat steps 1-2 until all $T$ incoming tasks are mapped
```

Figure 5: Pseudocode for MaxRobust heuristic.

```
1. map the new incoming task using FRMCT

2. if set of mappable tasks is not empty
   a) for each task find the feasible machine that minimizes computation time (first Min),
      ignoring other tasks in the currently mappable set
   b) from the above task/machine pairs, find the pair that gives the minimum completion
      time (second Min)
   c) assign the task to the machine and remove it from the set of mappable tasks
   d) repeat a-c until all tasks are remapped

3. repeat steps 1-2 until all $T$ incoming tasks are mapped
```

Figure 6: Pseudocode for FMCTMCT heuristic.

```
1. map the new incoming task using FRMCT

2. if set of mappable tasks is not empty
   a) for each task find the feasible machine that minimizes computation time (Min),
      ignoring other tasks in the currently mappable set
   b) from the above task/machine pairs, find the pair that gives the maximum robustness
      radius (Max)
   c) assign the task to the machine and remove it from the set of mappable tasks
   d) repeat a-c until all tasks are remapped

3. repeat steps 1-2 until all $T$ incoming tasks are mapped
```

Figure 7: Pseudocode for FMRMCT heuristic.

```
1. map the new incoming task using FRMCT

2. if set of mappable tasks is not empty
   a) for each task find the feasible machine that gives maximum robustness radius (Max),
      ignoring other tasks in the currently mappable set
   b) from the above task/machine pairs, find the pair that gives the minimum completion
      time (Min)
   c) assign the task to the machine and remove it from the set of mappable tasks
   d) repeat a-c until all tasks are remapped

3. repeat steps 1-2 until all $T$ incoming tasks are mapped
```

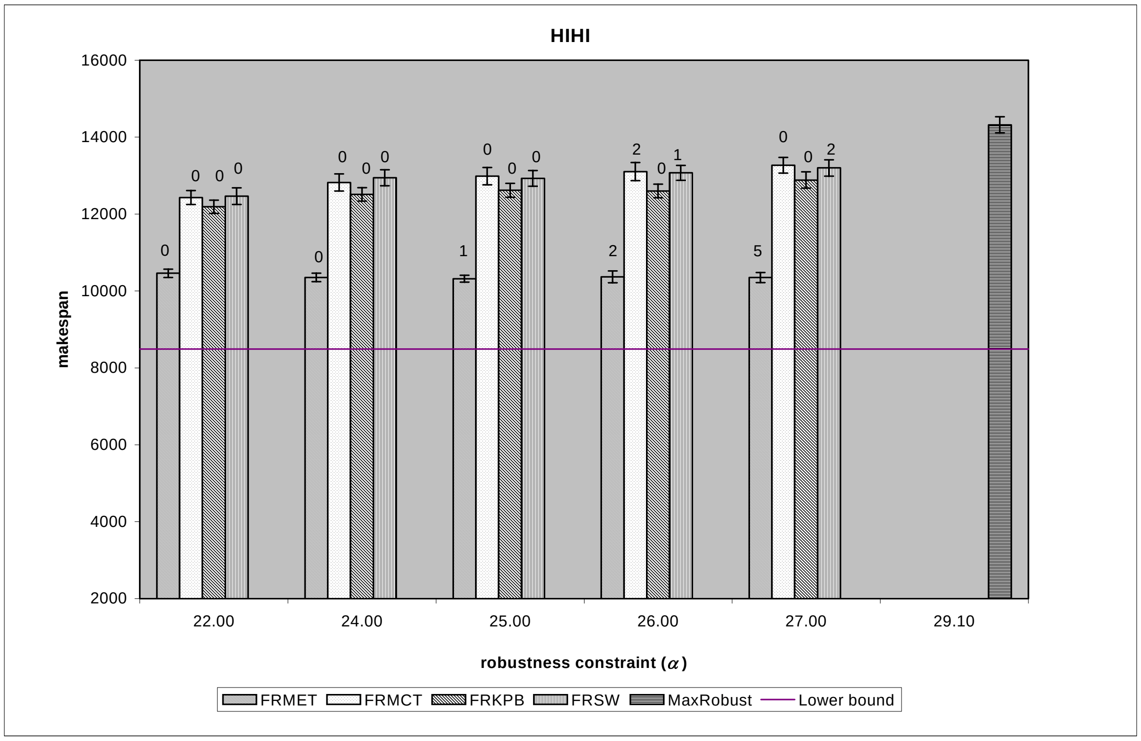
Figure 8: Pseudocode for FMCTMR heuristic.

```
1. map the new incoming task using FRMCT

2. if set of mappable tasks is not empty
   a) for each task find the feasible machine that gives maximum value of the objective
      function ($s(j,t)$), ignoring other tasks in the currently mappable set
   b) from the above task/machine pairs, find the pair that gives the maximum value of
      $s(j,t)$
   c) assign the task to the machine and remove it from the set of mappable tasks
   d) repeat a-c until all tasks are remapped

3. repeat steps 1-2 until all $T$ incoming tasks are mapped
```

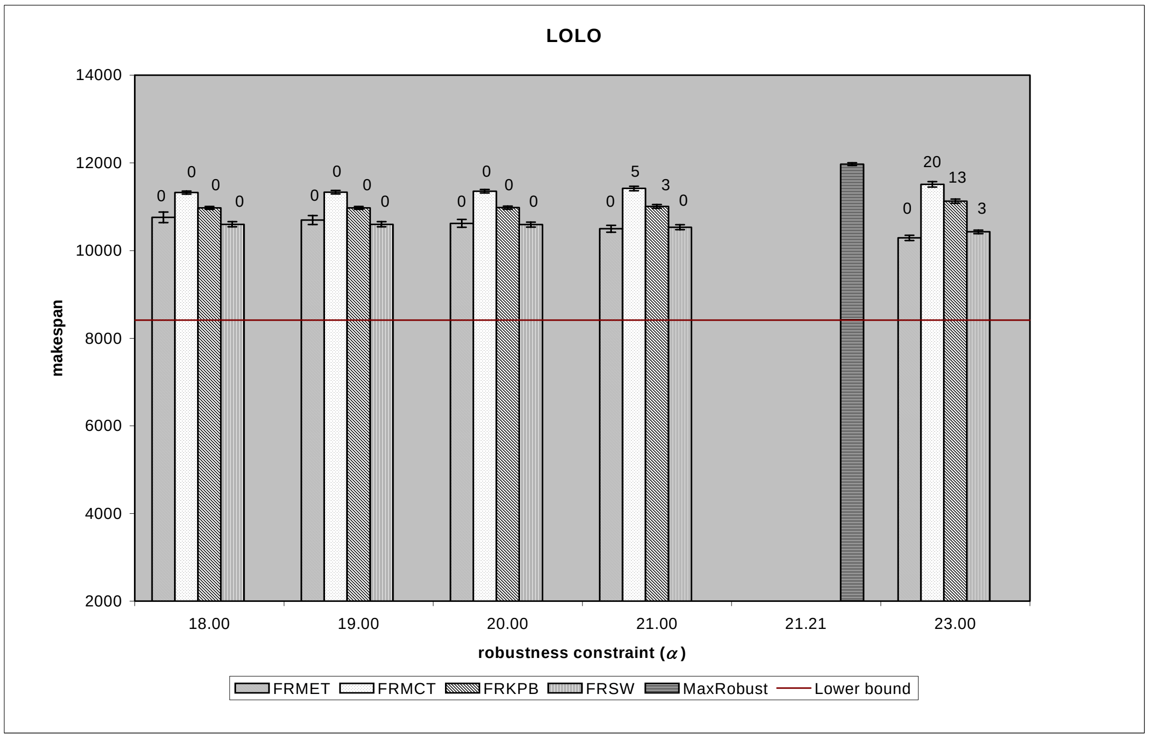
Figure 9: Pseudocode for MWMW heuristic.

```
1. map the new incoming task using MR

2. if set of mappable tasks is not empty
   a) for each task find the machine that gives maximum robustness radius (first Max),
      ignoring other tasks in the currently mappable set
   b) from the above task/machine pairs, find the pair that gives the maximum value
      (second Max)
   c) assign the task to the machine and remove it from the set of mappable task
   d) repeat a-c until all tasks are remapped

3. repeat steps 1-2 until all $T$ incoming tasks are mapped
```

Figure 10: Pseudocode for MRMR heuristic.

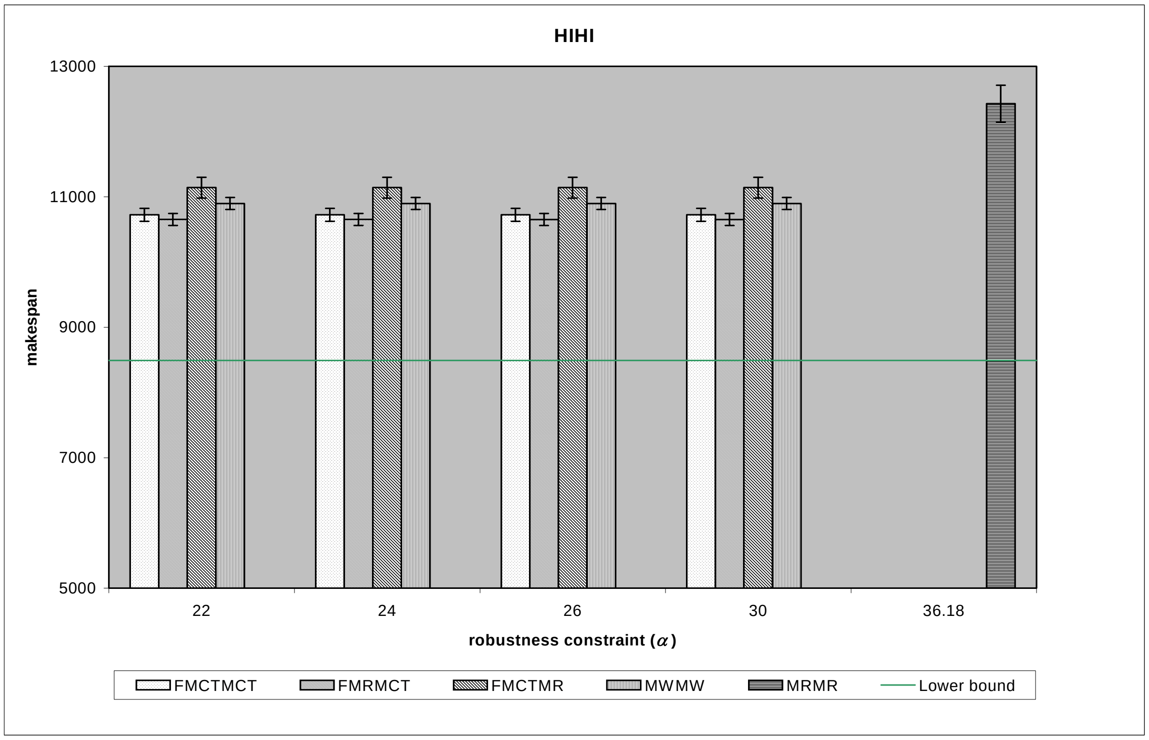


(a)

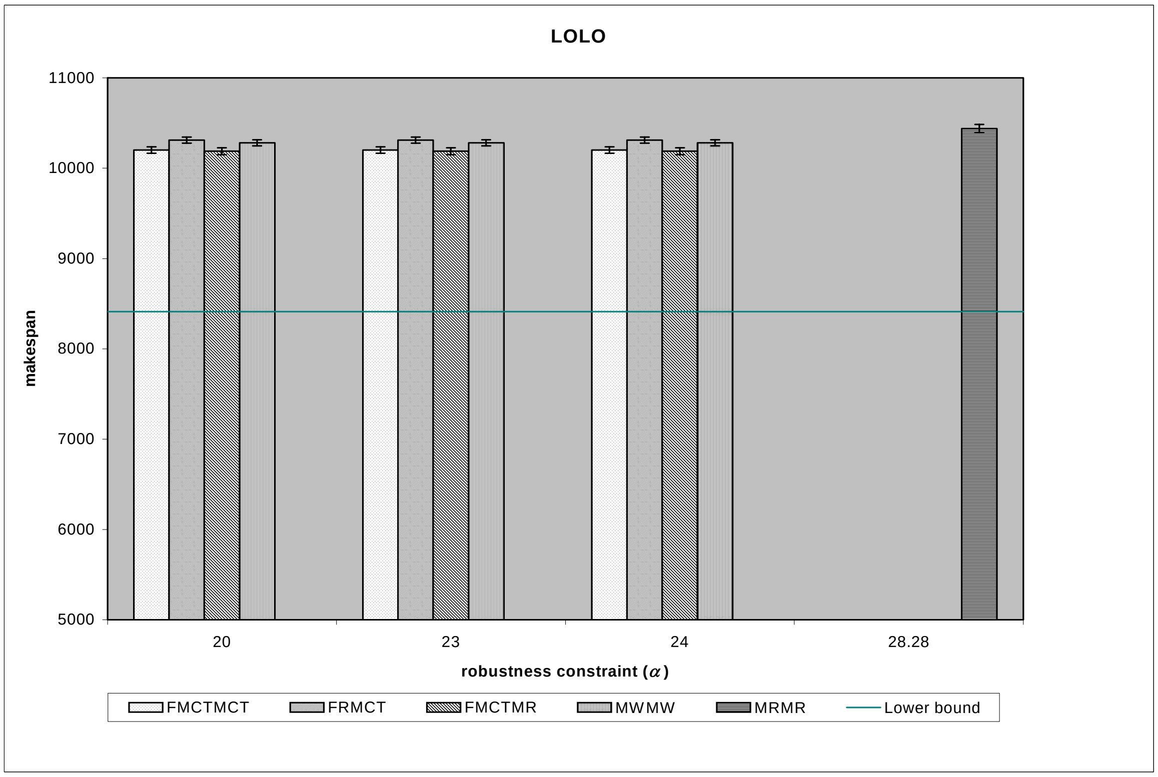


(b)

Figure 11: Simulation results for immediate mode heuristics for (a) HIHI heterogeneity, and (b) LOLO heterogeneity.



(a)



(b)

Figure 12: Simulation results for pseudo-batch mode heuristics for (a) HIHI heterogeneity, and (b) LOLO heterogeneity.

Table 1: Average execution times, in seconds, of a mapping event for the proposed heuristics.

heuristic	average execution time
FRMET	0.0010
FRMCT	0.0019
FRKPB	0.0019
FRSW	0.0015
MaxRobust	0.0059
FMCTMCT	0.023
FMRMCT	0.28
FMCTMR	0.28
MWMW	0.211
MRMR	0.563

Table 2: Maximum and average number of mapping events for which the MET machine was not feasible for HIHI and LOLO heterogeneity.

HIHI		
robustness constraint (α)	maximum (over successful trials)	average (over successful trials)
22.00	41	14
24.00	54	22
25.00	73	30
26.00	79	36
27.00	88	42

LOLO		
robustness constraint (α)	maximum (over successful trials)	average (over successful trials)
18.00	5	0
19.00	10	1
20.00	14	3
21.00	26	6
21.21	26	6
23.00	56	17