J Supercomput (2007) 42: 33–58
DOI 10.1007/s11227-006-0035-y

Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness

Ashish M. Mehta · Jay Smith · H.J. Siegel · Anthony A. Maciejewski · Arun Jayaseelan · Bin Ye

Published online: 31 March 2007

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 develops a model for quantifying robustness in a dynamic heterogeneous computing environment where task execution time estimates are known to contain errors. This mathematical expression of robustness is then applied to two different problem environments. Several heuristic solutions to both problem variations are presented that utilize this expression of robustness to influence mapping decisions.

A.M. Mehta · J. Smith · H.J. Siegel · A.A. Maciejewski · A. Jayaseelan · B. Ye
Electrical and Computer Engineering Department, Colorado State University, Fort Collins, CO 80523-1373, USA

A.M. Mehta
e-mail: ammehta@engr.colostate.edu

A.A. Maciejewski
e-mail: aam@engr.colostate.edu

A. Jayaseelan
e-mail: arun@engr.colostate.edu

B. Ye
e-mail: binye@engr.colostate.edu

J. Smith
IBM, 6300 Diagonal Highway, Boulder, CO 80301, USA
e-mail: bigfun@us.ibm.com

H.J. Siegel (✉)
Computer Science Department, Colorado State University, Fort Collins, CO 80523-1373, USA
e-mail: hj@engr.colostate.edu

Keywords Resource management · Robustness · Dynamic mapping · Makespan · Resource allocation

1 Introduction

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

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 [24]. 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., [10, 13, 17]). Thus, the development of heuristic techniques to find a near-optimal solution for the mapping problem is an active area of research (e.g., [2, 5, 6, 8, 12, 14, 22, 24, 26, 32]).

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., [1, 15, 16, 19, 25, 34]). 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., [16, 18, 19, 21, 30, 33]).

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. Two variations to this basic problem are considered in this work.

The first problem variation (robustness constrained) focuses on determining a dynamic mapping for a set of tasks that minimizes the estimated makespan (using the estimated ETC values) while still being able to tolerate a quantifiable amount of variation in the ETC values of the mapped tasks. Therefore, the goal of heuristics in this problem variation is to obtain a mapping that minimizes makespan while maintaining a certain level of robustness at each mapping event.

In the second problem variation (makespan constrained), the goal of the heuristics is to maximize the robustness of a resource allocation while ensuring that the makespan for the resource allocation is below a specified limit. Maximizing robustness in this context is equivalent to maximizing the amount of tolerable variation that can occur in ETC times for mapped tasks while still ensuring that a makespan constraint can be met by the resource allocation.

Dynamic mapping heuristics can be grouped into two categories: immediate mode and batch mode [24]. *Immediate mode* heuristics *immediately* map a task to some machine in the system for execution upon the task's arrival. In contrast, *batch mode* heuristics accumulate tasks until a specified condition is satisfied before mapping tasks—e.g., a certain number of tasks accumulate, or a specified amount of time elapses. When the specified condition is satisfied a mapping event occurs and the entire batch of tasks 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*.

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 [27]. 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. a model for quantifying dynamic robustness,
2. heuristics for solving the two resource management problem variations,
3. simulation results for the proposed heuristics for each problem variation, and
4. a mathematical bound on the performance feature for each of the resource management problem variations.

The remainder of the paper is organized as follows. Section 2 formally states the investigated research problem. Section 3 describes the simulation setup. Heuristic solutions to the robustness constrained problem variation of the presented problem including an upper bound on the attainable robustness value are presented and evaluated in Sect. 4. Section 5 presents heuristics for the makespan constrained problem variation of the dynamic robustness problem along with their evaluation and a bound on the performance feature. Related work is considered in Sect. 6 and Sect. 7 concludes the paper.

2 Problem statement

In this study, T independent tasks (i.e., there is no inter-task 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 *m*achine j's *q*ueue and let $scet_j(t)$ denote the *s*tarting time of the *c*urrently *e*xecuting *t*ask on machine j. Mathematically, given some machine j

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

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,

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

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 at time t, 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,

$$\rho_\mu(t) = \min_{\forall j \in M} \bigl\{ r_\mu\bigl(F_j(t)\bigr) \bigr\}. \tag{4}$$

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, i.e., the actual makespan will be within τ time units of the predicted makespan.

For the robustness constrained problem variation, the dynamic robustness metric is used as a constraint. Let α be the *minimum acceptable robustness* 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 the robustness constrained problem variation 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.

For the makespan constrained problem variation, let T_e be the set of all mapping event times. The *robustness value* of the final mapping is defined as the smallest robustness metric that occurs at any mapping event time in T_e. The primary objective of heuristics in the makespan constrained problem variation 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 problem variation must complete all T incoming tasks within an overall makespan constraint (γ). Therefore, the goal of heuristics in this problem variation 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).

3 Simulation setup

The simulated environment consists of $T = 1024$ independent tasks and $M = 8$ machines for both problem variations. This number of tasks and machines was chosen to present a significant mapping challenge for each heuristic and to make an exhaustive search for an optimal solution infeasible (however, the presented techniques can be applied to environments with different numbers of tasks and machines). As stated earlier, each task arrives dynamically and the arrival times are not known *a priori*. For the robustness constrained problem variation, 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*) [8]. While for the makespan constrained problem variation, 200 different ETC matrices were generated, 100 each for HIHI, and LOLO. The larger number of ETC matrices (for the makespan constrained problem variation) was needed to produce statistically reliable results. The LOLO ETC matrices model an environment where different tasks have similar execution times on a machine and also, the machines have similar capabilities, e.g., a cluster of workstations employed to support transactional data processing. In contrast, the HIHI ETC matrices model an environment where the computational requirements of tasks vary greatly and there is a set of machines with diverse capabilities, e.g., a computational grid comprised of SMPs, workstations, and supercomputers, supporting fast compilations of small programs as well as time-consuming complex simulations.

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) [8]. 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 with a mean task inter-arrival rate of eight seconds. In order to accentuate the difference in performance of the pseudo-batch mode heuristics in the robustness constrained problem variation, the mean task inter-arrival rate was decreased to six 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 robustness constrained problem variation, the mean task execution time was set to 100 seconds while, for the makespan constrained problem variation, the mean task execution time was 120 seconds. For both problem variations, a COV value of 0.9 was used for HIHI and a value of 0.3 was used for LOLO. The value of τ chosen for this study was 120 seconds. The performance of each heuristic, was studied across all simulation *trials*, i.e., a trial corresponds to a different ETC matrix.

4 Robustness constrained heuristics

4.1 Heuristics overview

Five immediate mode and five pseudo-batch mode heuristics were studied for this variation of the problem. 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*.

4.2 Immediate mode heuristics

The following is a brief description of the immediate mode heuristics for this problem variation. Recall that in immediate mode, only the new incoming task is considered for mapping. Thus, the behavior of the heuristic is highly influenced by the order in which the tasks arrive.

4.2.1 Feasible robustness minimum execution time (FRMET)

FRMET is based on the MET concept in [8, 24] where each incoming task is mapped to its minimum execution time machine regardless of the number of pending tasks on that machine. However, for each incoming task, FRMET first identifies the feasible set of machines. The incoming task is assigned to the machine in the feasible set of machines that provides the minimum *execution* time for the task. The procedure at each mapping event can be summarized as follows:

i. for the new incoming task find the feasible set of machines. If the set is empty, exit with error (“constraint violation”)
ii. from the above set, find the minimum execution time machine
iii. assign the task to the machine
iv. update the machine available time

4.2.2 Feasible robustness minimum completion time (FRMCT)

FRMCT is based on the MCT concept in [8, 24] where each incoming task is mapped to its minimum completion time machine. However, 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. The procedure at each mapping event can be summarized as follows:

i. for the new incoming task find the feasible set of machines. If the set is empty, exit with error (“constraint violation”)
ii. from the above set, find the minimum completion time machine
iii. assign the task to the machine
iv. update the machine available time

4.2.3 Feasible robustness K-percent best (FRKPB)

FRKPB is based on the KPB concept in [20, 24]. FRKPB tries to combine the aspects of both MET and MCT. FRKPB first finds the feasible set of machines for the newly arrived task. From this set, FRKPB identifies the k-percent feasible machines that have the smallest *execution* time 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 $k = 50$ was found to give the best results. The procedure at each mapping event can be summarized as follows:

i. for the new incoming task find the feasible set of machines. If the set is empty, exit with error ("constraint violation")
ii. from the above set, find the top $m = 4$ machines based on execution time
iii. from the above find the minimum completion time machine
iv. assign the task to the machine
v. update the machine available time

4.2.4 Feasible robustness switching (FRSW)

FRSW is based on the SW concept in [20, 24]. As applied in this research, FRSW combines aspects of both the FRMET and the FRMCT heuristics. A *load balance ratio* (LBR) 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 rises above a high set point, denoted T_{high}, FRSW switches to the FRMET heuristic. When the ratio falls below a low set point, denoted T_{low}, FRSW switches to the FRMCT heuristic. The values for the switching set points were determined experimentally using sample training data. The procedure at each mapping event can be summarized as follows:

```
i.   for the new incoming task find the feasible set of machines. If the set is empty,
     exit with error ("constraint violation")
ii.  calculate the load balance ratio (LBR)
iii. initial mapping heuristic—FRMCT
     if LBR > $T_{\text{high}}$ map using FRMET
        else if LBR < $T_{\text{low}}$ map using FRMCT
          else if $T_{\text{low}} \le$ LBR $\le T_{\text{high}}$ map using previous mapping heuristic
```

4.2.5 Maximum robustness (MaxRobust)

MaxRobust 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. The procedure at each mapping event can be summarized as follows:

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

ii. assign task to maximum robustness radius machine
iii. update the machine available time

4.3 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. The following is a brief description of the pseudo-batch mode re-mapping heuristics.

4.3.1 Feasible robustness minimum completion time-minimum completion time (FMCTMCT)

FMCTMCT uses a variant of Min-Min heuristic defined in [2, 17]. 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. The procedure at each mapping event can be summarized as follows:

i. map the new incoming task using FRMCT
ii. if set of mappable tasks is not empty
   (a) for each task, find the set of feasible machines. If the set is empty for any task, exit with error (“constraint violation”)
   (b) for each task find the feasible machine that minimizes computation time (first Min), ignoring other mappable tasks
   (c) from the above task/machine pairs, find the pair that gives the minimum completion time (second Min)
   (d) assign the task to the machine and remove it from the set of mappable tasks
   (e) update the machine available time
   (f) repeat a–e until all tasks are remapped

4.3.2 Feasible robustness maximum robustness-minimum completion time (FMRMCT)

FMRMCT builds on concept of the Max-Min heuristic [2, 17]. 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. The procedure at each mapping event can be summarized as follows:

i. map the new incoming task using FRMCT

ii. if set of mappable tasks is not empty
   (a) for each task, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation")
   (b) for each task find the feasible machine that minimizes computation time (Min), ignoring other mappable tasks
   (c) from the above task/machine pairs, find the pair that gives the maximum robustness radius (Max)
   (d) assign the task to the machine and remove it from the set of mappable tasks
   (e) update the machine available time
   (f) repeat a-e until all tasks are remapped

4.3.3 Feasible minimum completion time-maximum robustness (FMCTMR)

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. The procedure at each mapping event can be summarized as follows:

i. map the new incoming task using FRMCT
ii. if set of mappable tasks is not empty
   (a) for each task, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation")
   (b) for each mappable task find the feasible machine that gives maximum robustness radius (Max), ignoring other mappable tasks
   (c) from the above task/machine pairs, find the pair that gives the minimum completion time (Min)
   (d) assign the task to the machine and remove it from the set of mappable tasks
   (e) update the machine available time
   (f) repeat a-e until all tasks are remapped

4.3.4 Maximum weighted Sum-maximum weighted Sum (MWMW)

MWMW builds on a concept in [29]. It combines the Lagrangian heuristic technique [9, 23] for deriving an objective function with the concept of Min-Min heuristic [17], 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 $F'_j(t) = F_j(t) + \sum \text{ETC}(i, j)$ for all tasks currently in the machine queue and the task currently under consideration. Let $\beta'(t)$ be the maximum of the finishing times $F'_j(t)$ at time t for all machines. Let $r'_\mu(F'_j(t))$ be the robustness radius for machine j. Let maxrob(t) be the maximum of the robustness radii at time t. Given η, an experimentally determined constant using training data, the *objective function* for MWMW is defined as

$$s(j,t) = \eta\left(1 - \frac{F_j'(t)}{\beta'(t)}\right) + (1-\eta)\left(\frac{r_\mu'(F_j'(t))}{\text{maxrob}(t)}\right) \tag{6}$$

The procedure at each mapping event can be summarized as follows:

i. map the new incoming task using FRMCT
ii. if set of mappable tasks is not empty
   (a) for each task, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation")
   (b) for each task find the feasible machine that gives maximum value of the objective function ($s(j,t)$), ignoring other mappable tasks
   (c) from the above task/machine pairs, find the pair that gives the maximum value of $s(j,t)$
   (d) assign the task to the machine and remove it from the set of mappable tasks
   (e) update the machine available time
   (f) repeat a-e until all tasks are remapped

4.3.5 Maximum robustness-maximum robustness (MRMR)

MRMR is provided here for comparison only as it optimizes robustness without considering makespan. When a task arrives it is initially mapped using the MaxRobust heuristic. Task remapping is performed by a variant of the Max-Max heuristic [17]. 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. The procedure at each mapping event can be summarized as follows:

i. map the new incoming task using MaxRobust
ii. if set of mappable tasks is not empty
   (a) for each task find the machine that gives maximum robustness radius (first Max), ignoring other mappable tasks
   (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) update the machine available time
   (e) repeat a-d until all tasks are remapped

4.4 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 $\text{arv}(i)$, the lower bound is given by

$$LB_1 = \max_{\forall i \in T}\left((\text{arv}(i) + \min_{\forall j \in M} \text{ETC}(i,j)\right). \tag{7}$$

Unfortunately, this bound neglects any time that the task spends waiting to execute. This can be significant in highly loaded systems. Therefore, a second lower bound that considers the total computational load was also used. This bound is given by,

$$LB_2 = \frac{\sum_{i=0}^{T} \{ \min_{\forall j \in M} \mathrm{ETC}(i,j) \}}{M}. \tag{8}$$

The lower bound on makespan can then be given by the maximum of the two bounds, i.e.,

$$LB = \max(LB_1, LB_2). \tag{9}$$

Clearly, this lower bound may not be achievable even by an optimal mapping, however, it is a tight lower bound because the case described by LB_1 is possible if a system is very lightly loaded.

4.5 Results

In Figs. 1 through 4, the average makespan results (with 95% confidence interval bars) are plotted, along with a lower bound on makespan. Figures 1 and 2 present the makespan results for the immediate mode heuristics for HIHI and LOLO heterogeneity, respectively. While, Figs. 3 and 4 present the makespan results for the pseudo-batch mode heuristics for HIHI and LOLO heterogeneity, 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 for 50 HIHI and 50 LOLO heterogeneity trials. In Figs. 1 through 4, 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.

The average execution times for each heuristic over all mapping events (on a typical unloaded 3 GHz Intel Pentium 4 desktop machine) in all 100 trials are shown in Table 1 and Table 2 for immediate and pseudo-batch mode, respectively. 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 the immediate mode heuristics, FRMET resulted in the lowest makespan 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 based on prior studies including a minimum execution time (*MET*) heuristic in other environments (that did not involve robustness and had different arrival rates and ETC matrices). It should be noted, however, that its performance in the HIHI case did result in multiple instances where it failed to find a mapping.

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 [8, 24], because it ignores machine loads and machine available times when making a mapping decision. The establishment of a feasible set of machines by the

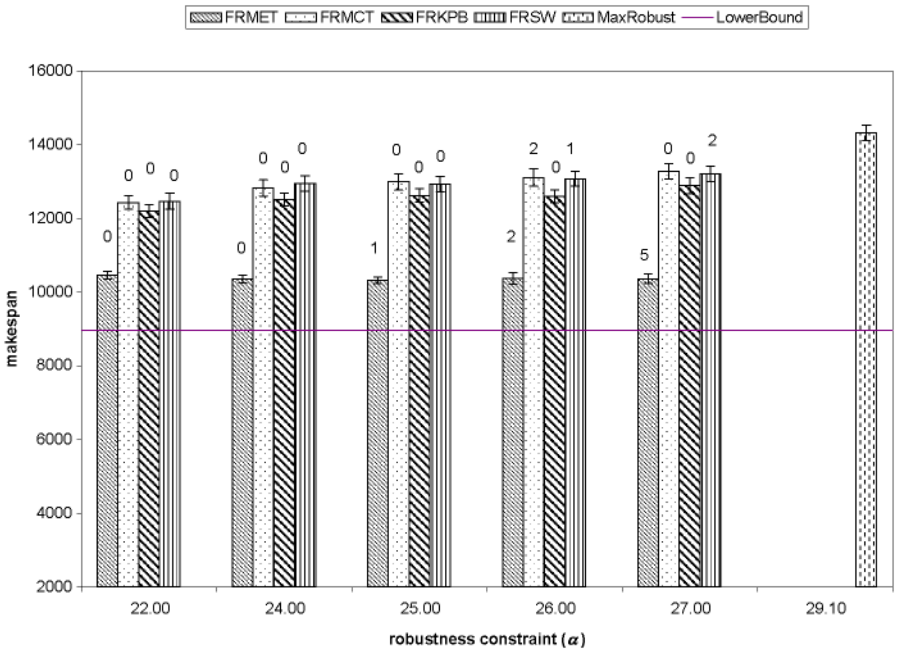

Fig. 1 Simulation results of makespan for different values of robustness constraint (α) for immediate mode heuristics for HIHI heterogeneity

FRMET heuristic indirectly balances the incoming task load across all of the machines. Also, because of the highly inconsistent nature of the data sets coupled with the high mean execution time (100 seconds), FRMET is able to maintain a lower makespan compared to FRMCT.

To illustrate this, consider the following ETC matrix:

	t_0	t_1	t_2	t_3	t_4
M_1	10	150	180	150	100
M_2	100	70	170	100	150
M_3	180	100	60	140	300

If the tasks arrive in the above order and the robustness constraint is $\alpha = 22$, the mapping obtained by FRMET would be:

M_1	$t_0(10)$	$t_4(100)$
M_2	$t_1(70)$	$t_3(100)$
M_3	$t_2(60)$	

whereas using FRMCT results in the following mapping:

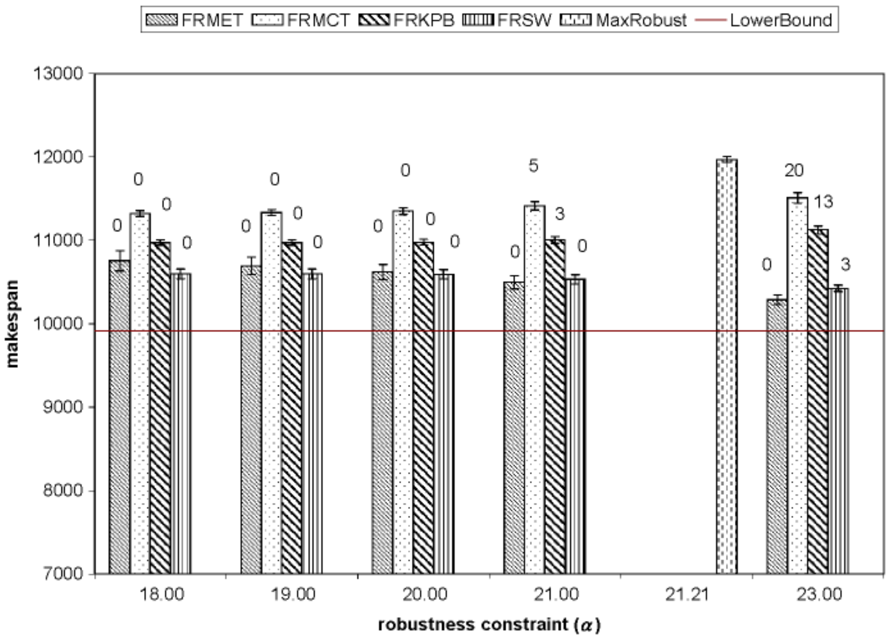

Fig. 2 Simulation results of makespan for different values of robustness constraint (α) for immediate mode heuristics for LOLO heterogeneity

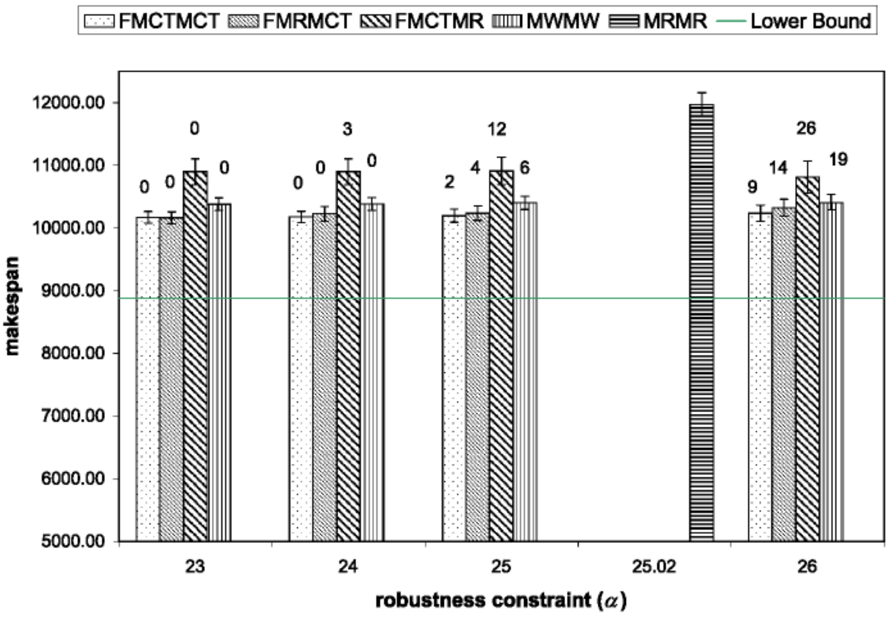

Fig. 3 Simulation results of makespan for different values of robustness constraint (α) for pseudo-batch mode heuristics for HIHI heterogeneity

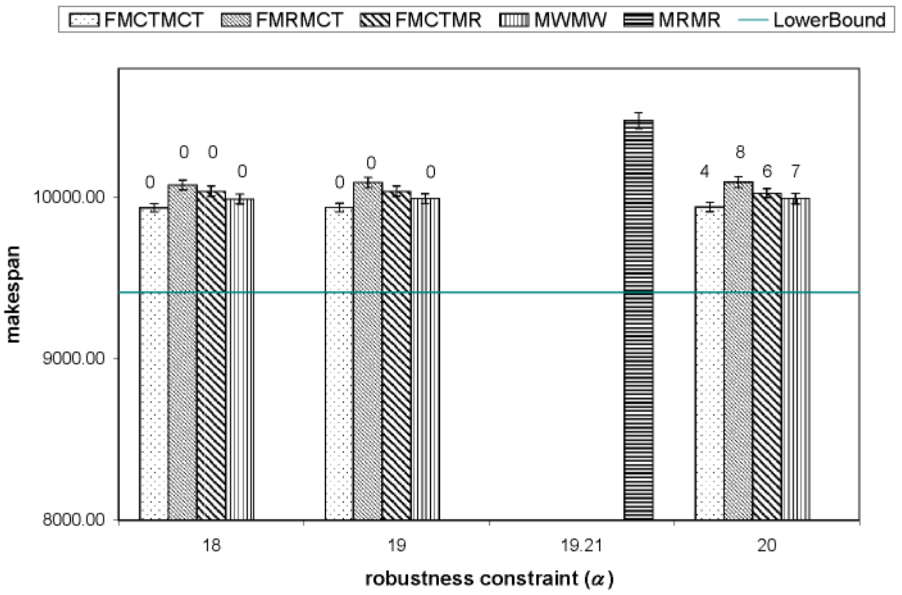

Fig. 4 Simulation results of makespan for different values of robustness constraint (α) for pseudo-batch mode heuristics for LOLO heterogeneity

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

Heuristics	avg. exec. time (sec.)
FRMET	0.001
FRMCT	0.0019
FRKPB	0.0019
FRSW	0.0015
MaxRobust	0.0059

Table 2 Average execution times, in seconds, of a mapping event for the proposed pseudo-batch mode heuristics

Heuristics	avg. exec. time (sec.)
FMCTMCT	0.023
FMRMCT	0.028
FMCTMR	0.028
MWMW	0.0211
MRMR	0.0563

M_1	$t_0(10)$	$t_3(150)$
M_2	$t_1(70)$	$t_4(150)$
M_3	$t_2(60)$	

Thus, the makespan obtained using FRMET is 170 while that obtained using FRMCT is 220.

Table 3 Maximum and average number of mapping events (over successful trials) for which the MET machine was not feasible for HIHI and LOLO heterogeneity

HIHI						
Robustness constraint (α)	22.00	24.00	25.00	26.00	27.00	
max	41	54	73	79	88	
avg	14	22	30	36	42	
LOLO						
Robustness constraint (α)	18.00	19.00	20.00	21.00	21.21	22.00
max	5	10	14	26	26	56
avg	0	1	3	6	6	7

Table 3 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. Thus, even though the vast majority of tasks are mapped to their MET machine, it is important to prevent those rare cases where doing so would make the mapping infeasible.

The FRKPB heuristic performed better than FRMCT (in terms of makespan) for LOLO heterogeneity and comparable to FRMCT for HIHI heterogeneity. FRKPB selects the k-percent feasible machines that have the smallest *execution* time for the task and then assigns the task to the machine in the set with the minimum *completion* time for the task. Thus, rather then trying to map the task to its best completion time machine, it tries to avoid putting the current task onto the machine which might be more suitable for some task that is yet to arrive. This foresight about task heterogeneity is missing in FRMCT, which might assign the task to a poorly matched machine for an immediate marginal improvement in completion time. This might possibly deprive some subsequently arriving better matched tasks of that machine, and eventually leading to a larger makespan than FRKPB.

The FRSW heuristic switches between FRMCT and FRMET depending on the LBR. In the HIHI case T_{low} was set to 0.6 and T_{high} was set to 0.9. With these values of the threshold, FRSW used FRMCT, on average, for 96% of the mapping events (out of total 1024) to map the incoming task. In the LOLO case T_{low} was set to 0.3 and T_{high} was set to 0.6. For these values of the thresholds FRSW used FRMET, on average, for 80% of the mapping events (out of total 1024) to map the incoming task. As stated earlier FRMET performs much better than FRMCT for both the HIHI and LOLO cases. Thus the better performance of FRSW, for LOLO heterogeneity, can be attributed to the fact that it maps a large number of tasks using FRMET as opposed to FRMCT. In contrast, for HIHI heterogeneity, a larger number of tasks are mapped using FRMCT and so the makespan is comparable to that of FRMCT.

An interesting observation was that the FRMCT heuristic was able to mantain a robustness constraint of $\alpha = 27$ for all 50 trials used in this study, but only for 48 trials when $\alpha = 26$ (for HIHI heterogeneity). This could be attributed to the volatile nature of the greedy heuristics. The looser robustness constraint ($\alpha = 26$) allowed for a pairing of task to machine that was disallowed for a tighter robustness constraint ($\alpha = 27$). That is, the early greedy selection proved to be a poor decision because it ultimately led to a mapping failure.

For the HIHI case all of 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.

When considering the performance of the pseudo-batch mode heuristics (Figs. 3 and 4) recall that they were evaluated across a different set of ETC matrices (mean task inter-arrival rate of six seconds as opposed to eight seconds for ETC matrices for immediate mode). The MWMW heuristic used a value of $\eta = 0.6$ for HIHI and $\eta = 0.3$ for LOLO.

For the HIHI heterogeneity trials, FMCTMCT and FMRMCT performed comparably, in terms of makespan, though FMRMCT had a higher failure rate than FRMCTMCT for high values of α. The inclusion of the concept of feasible machines helped FMCTMCT and FMRMCT maintain a high level of robustness. The FMCTMR heuristic had a higher makespan as compared to FMRMCT. The reason being the first stage choice of machines for these two-stage greedy heuristics. The FMRMCT heuristic tries to minimize the completion time in the first stage and then selects the task/machine pair that maximizes the robustness radius, as opposed to maximizing the robustness radius in stage one and then selecting the task/machine pair that minimizes the completion time as used by FMCTMR.

For the LOLO heterogeneity trials, FMCTMCT performed the best on average, while MWMW performed comparably (in terms of makespan). The motivation behind using MRMR was to greedily maximize robustness at every mapping event. As can be seen from Figs. 3 and 4, the MRMR heuristic was able to maintain a high level of robustness, however, it had the worst makespan among the heuristics studied.

5 Makespan constrained heuristics

5.1 Heuristics 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.

5.2 Heuristic descriptions

5.2.1 Minimum completion time-minimum completion time (MinCT-MinCT)

The MinCT-MinCT heuristic is similar to the FMCTMCT heuristic studied in the robustness constrained problem variation but with the new definition of the feasible machine.

5.2.2 Maximum robustness-maximum robustness (MaxR-MaxR)

As was seen in the robustness constrained problem variation, the MRMR heuristic was able to maintain a high level of robustness, but had a higher makespan. The goal in this problem variation is to maximize the robustness at each mapping event, and hence a variation of the MRMR heuristic is employed. However, unlike the MRMR heuristic, 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.

5.2.3 Maximum robustness-minimum completion time (MaxR-MinCT)

MaxR-MinCT is similar to the FMRMCT heuristic studied in robustness constrained problem variation, but with the new definition of a feasible machine.

5.2.4 Minimum completion time-maximum robustness (MinCT-MaxR)

The MinCT-MinCT heuristic is similar to the FMCTMR heuristic studied in the robustness constrained problem variation but with the new definition of a feasible machine.

5.2.5 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 Subsect. 5.4. 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 metric.
   (j) From the above task/machine pairs select the pair that maximizes the robustness metric.
   (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).

5.3 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 [35]), 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 Eq. 3, which correspondingly improves the robustness metric.

5.4 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 Eq. 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.

5.5 Results

In Figs. 5 and 6, the average robustness value (over all mapping events) for each heuristic is plotted with their 95% confidence intervals. The average execution time of each heuristic over all mapping events in all 200 trials is shown in Table 4. Recall that the heuristics operate in a pseudo-batch mode, therefore, the times in Table 4 are the average time for each heuristic to map an entire batch of tasks.

As can be seen from Figs. 5 and 6, MxMxMnMn with fine tuning gives the best robustness result for both the HIHI and LOLO cases (although there is one failure). 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.

Although, MinCT-MinCT is able to achieve one of the best makespan (Figs. 7 and 8) 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.

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

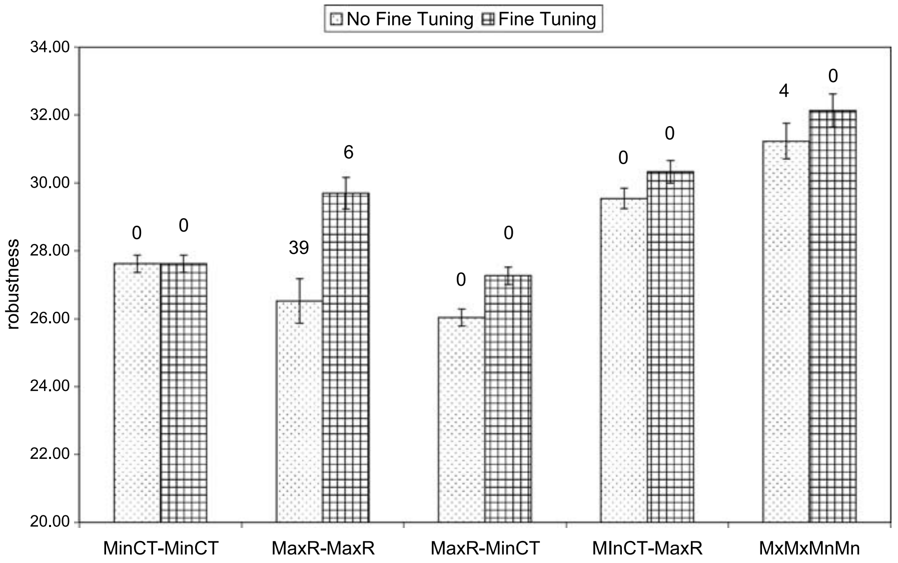

Fig. 5 Average robustness value (over all mapping events) for the HIHI case with $\gamma = 14000$

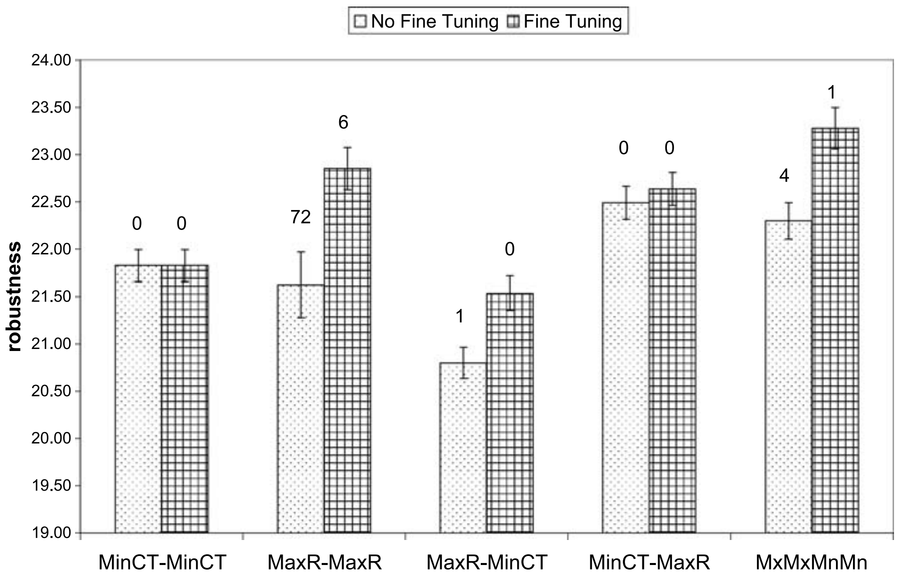

Fig. 6 Average robustness value (over all mapping events) for the LOLO case with $\gamma = 12500$

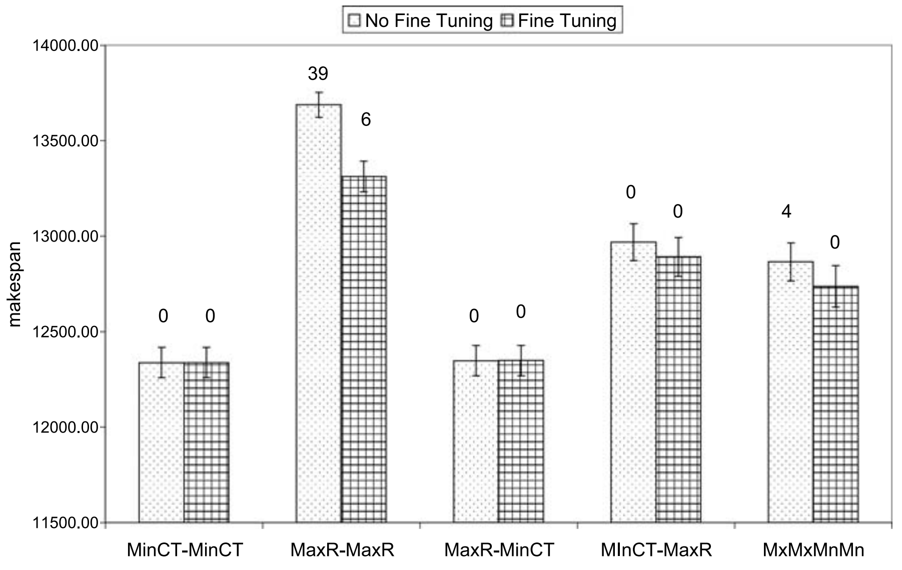

Fig. 7 Average makespan for the HIHI case with $\gamma = 14000$

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:	$t_2(50)$	

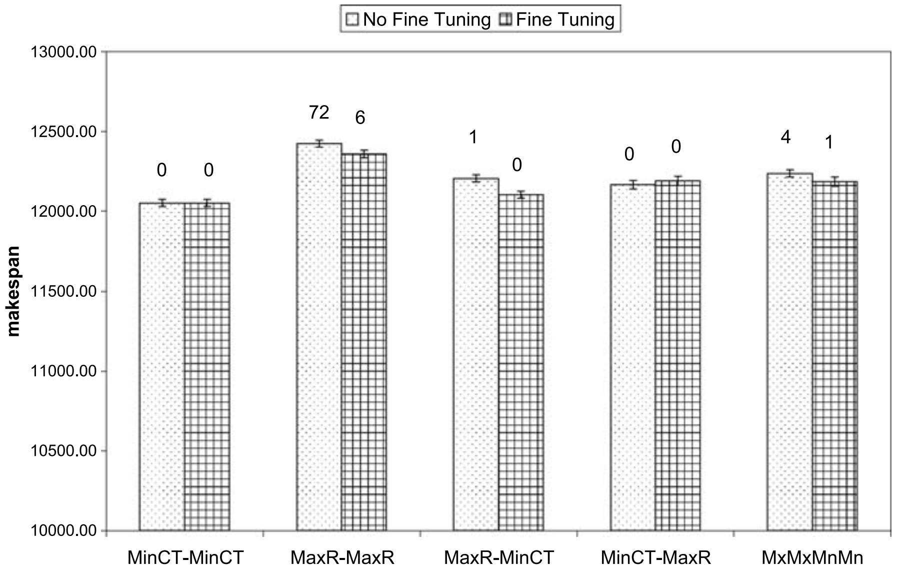

Fig. 8 Average makespan for the LOLO case with $\gamma = 12500$

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.

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:	$t_2(30)$	$t_3(80)$

Table 4 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

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.

6 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., [7, 11, 28, 31]).

The research in [7] 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 [11] 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 [11] 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 [28] 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 [31], the robustness is derived using the same FePIA procedure used here. However the environment considered is static (off-line), as opposed to the dynamic (online) environment in this research. The robustness metric and heuristics employed in a dynamic environment are substantially different from those employed in [31].

7 Conclusion

This research presented a model for quantifying robustness in a dynamic environment. It also involved the characterization and modeling of two dynamic heterogeneous computing problem environments, and examined and compared various heuristic techniques for each of the two problem variations. This work also presented the bounds on the highest attainable value of the system performance feature for both problem variations.

The robustness constrained problem variation presented five immediate and five pseudo-batch mode heuristics. For the immediate mode heuristics, FRMET gave the lowest makespan for both the heterogeneity trials, but it also had a high number of failed trials (for HIHI heterogeneity). The FRKPB heuristic had the lowest number of failed trials for HIHI heterogeneity. For the pseudo-batch mode, FMCTMCT performed the best in terms of both makespan and failed number of trials. 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.

For the makespan constrained problem variation five pseudo-batch heuristics were designed and evaluated. A process of fine tuning was also adapted to maximize the robustness level. Of the proposed heuristics, MxMxMnMn with fine tuning performed the best for the proposed simulation environment.

Acknowledgement 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.

Preliminary portions of this material were accepted for presentation at the 2006 International Conference on Parallel & Distributed Processing Techniques & Applications (PDPTA 2006) and the 12th International Conference on Parallel & Distributed Systems (ICPADS 2006).

References

1. Ali S, Braun TD, Siegel HJ, Maciejewski AA, Beck N, Boloni L, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B (2005) Characterizing resource allocation heuristics for heterogeneous computing systems. In: Hurson AR (ed), Advances in computers vol 63: parallel, distributed, and pervasive computing. Elsevier, Amsterdam, Netherlands, pp 91–128

2. Ali S, Kim J-K, Yu Y, Gundala SB, Gertphol S, Siegel HJ, Maciejewski AA, Prasanna V (2002) Utilization-based techniques for statically mapping heterogeneous applications onto the HiPer-D heterogeneous computing system. Parallel Distrib Comput Pract, Special issue on parallel numerical algorithms on faster computers 5(4)
3. Ali S, Maciejewski AA, Siegel HJ, Kim J-K (2004) Measuring the robustness of a resource allocation. Trans Parallel Distrib Syst 15(7):630–641
4. Ali S, Siegel HJ, Maheswaran M, Hensgen D, Ali S (2000) Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang J Sci Eng, Special 50th anniversary issue (invited), 3(3):195–207
5. Barada H, Sait SM, Baig N (2001) Task matching and scheduling in heterogeneous systems using simulated evolution. In: 10th IEEE heterogeneous computing workshop (HCW 2001), 15th international parallel and distributed processing symposium (IPDPS 2001), Apr 2001
6. Banicescu I, Velusamy V (2001) Performance of scheduling scientific applications with adaptive weighted factoring. In: 10th IEEE heterogeneous computing workshop (HCW 2001), 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), Apr 2001
7. Bean J, Birge J, Mittenthal J, Noon C (1991) Matchup scheduling with multiple resources, release dates and disruptions. J Oper Res Soc Am 39(3):470–483
8. Braun TD, Siegel HJ, Beck N, Boloni L, Freund RF, Hensgen D, Maheswaran M, Reuther AI, Robertson JP, Theys MD, Yao B (2001) A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. J Parallel Distrib Comput 61(6):810–837
9. Castain R, Saylor WW, Siegel HJ (2004) Application of lagrangian receding horizon techniques to resource management in ad-hoc grid environments. In: 13th heterogeneous computing workshop (HCW 2004), in the proceedings of the 18th international parallel and distributed processing symposium (IPDPS 2004), Apr 2004
10. Coffman EG, Jr (ed), (1976) Computer and job-shop scheduling theory. Wiley, New York
11. Daniels RL, Carrilo JE (1997) β-Robust scheduling for single-machine systems with uncertain processing times. IIE Trans 29(11):977–985
12. Eshaghian MM (ed) (1996) Heterogeneous computing. Artech House, Norwood
13. Fernandez-Baca D (1989) Allocating modules to processors in a distributed system. IEEE Trans Softw Eng SE-15(11):1427–1436
14. Foster I, Kesselman C (eds) (1999) The grid: Blueprint for a new computing infrastructure. Morgan Kaufmann, San Fransisco
15. Freund RF, Siegel HJ (1993) Heterogeneous processing. IEEE Comput 26(6):13–17
16. Ghafoor A, Yang J (1993) A distributed heterogeneous supercomputing management system. IEEE Comput 26(6):78–86
17. Ibarra OH, Kim CE (1977) Heuristic algorithms for scheduling independent tasks on non-identical processors. J ACM 24(2):280–289
18. Kafil M, Ahmad I (1998) Optimal task assignment in heterogeneous distributed computing systems. IEEE Concur 6(3):42–51
19. Khokhar A, Prasanna VK, Shaaban ME, Wang C (1993) Heterogeneous computing: challenges and opportunities. IEEE Comput 26(6):18–27
20. Kim J-K, Shivle S, Siegel HJ, Maciejewski AA, Braun T, Schneider M, Tideman S, Chitta R, Dilmaghani RB, Joshi R, Kaul A, Sharma A, Sripada S, Vangari P, Yellampalli SS (2003) Dynamic mapping in a heterogeneous environment with tasks having priorities and multiple deadlines. In: 12th Heterogeneous computing workshop (HCW 2003), in the proceedings of the 17th international parallel and distributed processing symposium (IPDPS 2003), Apr 2003
21. Leangsuksun C, Potter J, Scott S (1995) Dynamic task mapping algorithms for a distributed heterogeneous computing environment. In: 4th IEEE heterogeneous computing workshop (HCW '95), 1995, pp 30–34
22. Leon VJ, Wu SD, Storer RH (1994) Robustness measures and robust scheduling for job shops. IIE Trans 26(5):32–43
23. Luh P, Zhao X, Wang Y, Thakur L (2000) Lagrangian relaxation neural networks for job shop scheduling. IEEE Trans Rob Autom 16(1):78–88
24. Maheswaran M, Ali S, Siegel HJ, Hensgen D, Freund RF (1999) Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. J Parallel Distrib Comput 59(2):107–121
25. Maheswaran M, Braun TD, Siegel HJ (1999) Heterogeneous distributed computing. In: Webster JG (ed) Encyclopedia of electrical and electronics engineering, vol 8, Wiley, New York, pp 679–690
26. Michalewicz Z, Fogel DB (2000) How to solve it: modern heuristics. Springer, New York

27. Naik VK, Sivasubramanian S, Bantz D, Krishnan S (2003) Harmony: a desktop grid for delivering enterprise computations. In: Fourth international workshop on grid computing (GRID 03), Nov 2003
28. Policella N (2005) 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
29. Shivle S, Siegel HJ, Maciejewski AA, Sugavanam P, Banka T, Castain R, Chindam K, Dussinger S, Pichumani P, Satyasekaran P, Saylor W, Sendek D, Sousa J, Sridharan J, Velazco J (2006) Static allocation of resources to communicating subtasks in a heterogeneous ad hoc grid environment. J Parallel Distribut Comput, Special Issue on Algorithms for Wireless and Ad-hoc Networks 66(4):600–611
30. Singh H, Youssef A (1996) Mapping and scheduling heterogeneous task graphs using genetic algorithms. In: 5th IEEE heterogeneous computing workshop (HCW '96), pp 86–97
31. Sugavanam P, Siegel HJ, Maciejewski AA, Oltikar M, Mehta A, Pichel R, Horiuchi A, Shestak V, Al-Otaibi M, Krishnamurthy Y, Ali S, Zhang J, Aydin M, Lee P, Guru K, Raskey M, Pippin A, Robust static allocation of resources for independent tasks under makespan and dollar cost constraints, J Parallel Distrib Comput accepted, to appear
32. Wu M-Y, Shu W, Zhang H, (2000) Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In: 9th IEEE Heterogeneous Computing Workshop (HCW 2000), May 2000, pp 375–385
33. Xu D, Nahrstedt K, Wichadakul D (2001) QoS and contention-aware multi-resource reservation. Clust Comput 4(2):95–107
34. Yang J, Ahmad I, Ghafoor A (1993) Estimation of execution times on heterogeneous supercomputer architectures. In: International conference on parallel processing, Aug 1993, pp I-219–I-226
35. Yarmolenko V, Duato J, Panda DK, Sadayappan P, (2000) Characterization and enhancement of dynamic mapping heuristics for heterogeneous systems. In: International conference on parallel processing workshops (ICPPW 00), Aug 2000, pp 437–444

Ashish M. Mehta is pursuing his M.S. degree in Electrical and Computer Engineering at Colorado State University, where he is currently a Graduate Teaching Assistant. He received his Bachelor of Engineering in Electronics from University of Mumbai, India. His research interests include resource management in distributed computing systems, computer architecture, computer networks, and embedded systems.

Jay Smith is a software Engineer in the Integrated Technology Delivery center of the IBM corporation. Prior to joining IBM, he studied at the University of Colorado at Boulder, where he earned his Bachelors degree in Mathematics. During his tenure at IBM, Jay has earned an Outstanding Technical Achievement ward for his work on Scanning Technology. He is also the co-inventor on over 21 patents filed in that and other areas. In addition to his duties within ITD, Jay is currently pursuing his Ph.D. in Electrical and Computer Engineering at Colorado State University. He is a member of the IEEE and the ACM.

H.J. Siegel was appointed the George T. Abell Endowed Chair Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU) in August 2001, where he is also a Professor of Computer Science. In December 2002, he became the first Director of the university-wide CSU Information Science and Technology Center (ISTeC). From 1976 to 2001, he was a professor at Purdue University. He received two B.S. degrees from MIT, and the MA, M.S.E., and Ph.D. degrees from Princeton University. Prof. Siegel has co-authored over 300 published papers on parallel and distributed computing and communication. He is a Fellow of the IEEE Fellow and a Fellow of the ACM. He was a Coeditor-in-Chief of the Journal of Parallel and Distributed Computing, and was on the Editorial Boards of both the IEEE Transactions on Parallel and Distributed Systems and the IEEE Transactions on Computers. He was Program Chair/Co-Chair of three major international conferences, General Chair/Co-Chair of six international conferences, and Chair/Co-Chair of five workshops. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. For more information, please see www.engr.colostate.edu/~hj.

Anthony A. Maciejewski received the BSEE, M.S., and Ph.D. degrees from Ohio State University in 1982, 1984, and 1987. From 1988 to 2001, he was a professor of Electrical and Computer Engineering at Purdue University, West Lafayette. He is currently the Department Head of Electrical and Computer Engineering at Colorado State University. He is a Fellow of the IEEE. A complete vita is available at: http://www.engr.colostate.edu/~aam.