Robust Resource Allocation in Weather Data Processing Systems

Mohana Oltikar[1], Jeff Brateman[3], Joe White[4], Jon Martin[5], Keith Knapp[1], Anthony A. Maciejewski[1], H. J. Siegel[1,2]

Colorado State University
[1]Department of Electrical & Computer Engineering
[2]Department of Computer Science
Fort Collins, CO 80523-1373
{*mohana*, *knappkei*, *aam*, *hj*}*@colostate.edu*

[3]*Purdue University*
School of Electrical and Computer Engineering
West Lafayette, IN 47907-2035
brateman@purdue.edu

[4] *Raytheon Company*
Software Engineer I
Aurora, CO 80011-9046
JPWhite1@Raytheon.com

[5] *R. L. Martin & Associates, Inc.*
Fort Collins, CO 80525
jrmartin@jrmartin.com

Abstract

Reliability of weather data processing systems is of prime importance to ensure the efficient operation of space-based weather monitoring systems. This work defines a heterogeneous weather data processing system that is susceptible to uncertainties in data set arrival times. The resource allocation must be robust with respect to these uncertainties. The tasks to be executed by the data processing system are classified into three broad categories: telemetry, tracking and control (high priority); data processing (medium priority); and data research (low priority). The high priority tasks must be completed before considering medium and low priority tasks. The goal of this research is to find a resource allocation that minimizes makespan of the high priority tasks, and to find a mapping that maximizes a function of the completion time and priority of the medium and low priority tasks. Different heuristic techniques to find near optimal solutions are studied, and their performance is evaluated.

1. INTRODUCTION

A space-based weather monitoring system consists of three major components: a satellite scheduling system, the satellite with its data collection sensors, and the data processing system (see Figure 1). The satellite scheduling system is responsible for issuing a request to the satellite about the data that must be collected and sends the same information to the data processing system. The satellite collects information about weather conditions on earth and transmits it back to the data processing system.

The data sent down by the satellite (the data set) must be processed before it can provide any value to the users. The tasks to be executed on the data set can be classified into three broad categories: TT&C - telemetry, tracking and control (high priority); data processing (medium priority); and data research (low priority) [21]. Currently, the computer systems used for processing the data sets are divided into three

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

Proceedings of the 2006 International Conference on Parallel Processing Workshops (ICPPW'06)
0-7695-2637-3/06 $20.00 © 2006 IEEE

distinct sets of processing elements, each dedicated to process one of the three types of tasks. The result of this partitioning is that each set of processing elements must be built for the worst-case demand, leading to resource and cost inefficiency.

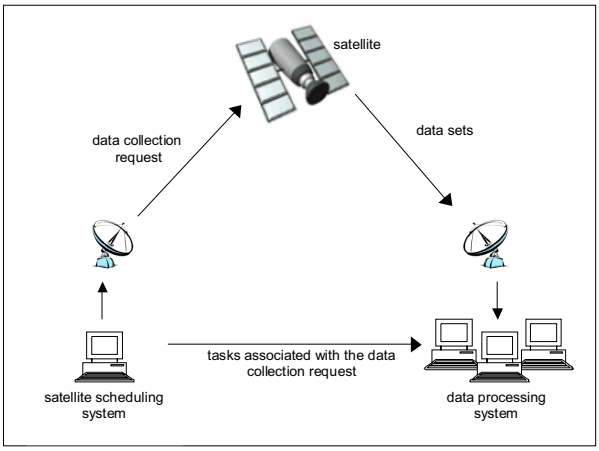

Figure 1: Overview of a space-based weather system.

The goal of this research is to develop a resource manager so that a smaller global bank of resources can replace the three sets of processing elements and increase the ability of the system to respond to a mixture of different task types and reduce the cost of the system. This system is a heterogeneous computing system, where some tasks have greater affinity to certain machines. The act of assigning (matching) each task to a machine and ordering (scheduling) the execution of the tasks in each machine is known as mapping, resource allocation, or resource management. Two different types of mapping are static and dynamic. Static mapping is performed when the tasks are mapped in an off-line planning phase, e.g., planning the schedule for tomorrow. Dynamic mapping is performed when the tasks are mapped in an on-line, real-time fashion, e.g., when tasks arrive at random intervals and are mapped as they arrive. In either case, the mapping problem has been shown, in general, to be NP-complete (e.g., [8, 13, 15]). Hence, the development of heuristic techniques to find near optimal solutions is an active area of research (e.g., [1, 12, 14, and 26]).

The performance of computing systems is susceptible to degradation due to unpredictable circumstances. Therefore, it is necessary to allocate the resources to tasks such that the robustness of the system in response to unpredictable events is maximized [3]. For this study the data set arrival times are not accurate, and the data set may arrive earlier or later than the estimated arrival time. Hence it is necessary to develop a performance metric to evaluate the quality of a mapping produced by the heuristics. The contributions of this research are: (1) a two-phase approach for scheduling tasks with multiple priorities in an oversubscribed system, (2) design and comparison of the performance of several heuristics for the proposed HC system model, and (3) a method for calculating a bound on the performance of a resource allocation for the proposed HC environment.

The rest of this chapter is organized as follows. A detailed overview of the system model is given in Section 2. Section 3 discusses the related work. Section 4 describes the simulation setup used for the experiments. The heuristics for phase I and phase II are explained in Sections 5 and 6, respectively. The bounds are presented in Section 7. The experimental results are discussed in Section 8.

2. PROBLEM STATEMENT

The current problem has a set of T tasks that must be executed on M machines for a given data set. It is assumed that all the tasks associated with a data set must arrive at a pre-determined time before the expected arrival time of the data set. Therefore, all the tasks associated with a data set are known *a priori*, and the mapping problem is a static mapping problem [1, 7]. A new data set is expected to arrive from the satellite after an interval of τ time units. When a new data set arrives, all tasks associated with the old data set are dropped, and the machine queues are emptied. It is assumed that the system is oversubscribed, i.e., not all tasks (for the current data set) can be completed by the expected arrival time of the next data set.

The expected arrival time of the next data set, τ, is only an estimate and the next data set might arrive earlier or later than the expected time. The high priority tasks ensure the proper functioning of the system; therefore, it is necessary to provide a guarantee that these tasks will be completed, even in the event of early arrival of the next data set. Therefore, the mapping problem is divided into two phases. The first phase deals with minimizing the completion time, t_c, referred to as the makespan, of the set of high priority tasks. For phase I, the behavior that makes the system robust is that all the high priority tasks can be completed before the next data set arrives. The uncertainty is arrival time of the next data set, and robustness for phase I, ρ, is quantified as

$$\rho = \tau - t_c. \qquad (1)$$

The second phase deals with the medium and low priority tasks. The medium priority tasks have a priority (P_i) equal to α, and the low priority tasks have

a priority P_i equal to 1. Analogous to robustness for phase I, the behavior of phase II to be maximized is the "worth" of the medium and low priority tasks that complete before the next data set arrives. The uncertainty is the arrival time of the next data set. The uncertainty is the arrival time of the next data set. The tasks are weighted based on the likelihood that they will complete. The likelihood l_i for a task t_i that is scheduled to complete at F_i is

$$l_i = \frac{\tau - F_i}{\tau}. \quad (2)$$

The worth, w_i, for the task is given by

$$w_i = l_i \cdot P_i \quad (3)$$

The quality of the mapping produced by the heuristics is compared using a worth function. Overall worth (W) of a mapping is the summation of the worth of each of the medium and low priority task. If set of medium and low priority tasks is denoted by T_1, the overall worth is given by

$$W = \sum_{i \in T_1} w_i. \quad (4)$$

The goal of this study is to maximize the robustness of the system by minimizing the makespan of phase I and maximizing the overall worth for phase II.

The mapping heuristics must complete before the arrival of the next data set, and it is required that the high priority tasks must be completed before the next data set arrives. Thus, the heuristic execution time is limited to t_c. It is assumed that t_c is greater than one second, and heuristics must take less than one second to generate a mapping. Therefore, greedy and "Min-Min" type heuristics are considered instead of slower evolutionary heuristics. Some of the notations used in this paper are summarized in Table 1.

3. RELATED WORK

There is a strong body of research related to the robustness of data processing systems. There are many definitions of robustness for various environments (e.g. [5, 9, 10, 11, and 17]).

The study in [5] discusses a job shop environment that is susceptible to sudden changes that render an existing schedule infeasible. It tries to increase the robustness of a scheduling system by increasing the flexibility of a schedule. It also tries to achieve a compromise between optimizing the objective function and maximizing the flexibility of a schedule. For this study, we use the definition of robustness used in [3] that focuses on the robustness of a resource allocation in a parallel and distributed computing system. It states: *"A resource allocation is defined to be robust with respect to specified system performance features against perturbations in specified system parameters if degradation in these features is limited when the perturbations occur."*

Table 1: Notations used in the study.

parameter	significance	Value
τ	approximate arrival time of next data set	5 minutes
α	P_i of medium priority tasks	256
$ETC(i,j)$	estimated time to compute task t_i on machine j	computed using COV method
t_c	makespan of phase I tasks	heuristic dependent
ρ	robustness of phase I tasks	heuristic dependent
l_i	likelihood that task t_i will complete before next data set arrives	heuristic dependent
w_i	worth of task t_i	heuristic dependent
W	worth of phase II tasks	heuristic dependent

The work in [23] defines a robustness metric for systems that use a stochastic model for task execution times. However, in our study, the task execution time estimates are deterministic.

Robustness of a system to uncertainties in execution time estimates is widely researched. The works in [19, 20, 22, and 24] describe a mapping environment where the robustness of a mapping for a set of tasks has to be maximized when the task execution times are inaccurate. Other studies such as [9, 10, and 17] explore the robustness of a job shop environment to such uncertainties. The work in [2] discusses a heterogeneous system that deals with uncertainties in load, while the studies in [3 and 10] explore the robustness of a system in the event of machine failures. In our work, the uncertainty is the data set arrival time, not execution times, load or machine failures.

The research in [6] describes an oversubscribed system for scheduling communications using antennas for a satellite range scheduling problem. For this study, each task has a priority and a deadline associated with it, and not all tasks can be scheduled before their deadlines. The goal is to minimize the number of tasks that cannot be completed before their

deadline. Our work differs from [6], as the individual tasks do not have a deadline, however, the execution of all the tasks in a data set is constrained by the estimated arrival of the next data set. Also, the time for the scheduler to generate a schedule in this study is much less. The work in [18] also discusses an oversubscribed environment of tasks with multiple priorities, but the work emphasizes that task priorities must be rigidly respected, i.e., a higher priority task can never be traded for a set of low priority tasks. However, in our study, the problem is divided into a two-phase scheduling problem, where, the higher priority tasks must be completed before the medium and low priority tasks can be considered, but the priorities within the medium and low priority tasks are not rigid.

4. SIMULATION SETUP

The simulation studies used to evaluate and compare the heuristics had 128 TT&C tasks (phase I tasks), 512 medium and low priority tasks (phase II tasks), and $M = 8$. The expected time to compute a task t_i on machine j ($ETC(i,j)$) was calculated using the coefficient of variance (COV) based method, as described in [4]. The *ETC*s generated here were partially consistent. For a consistent *ETC* matrix, if machine x has a lower execution time than machine y for a task t_i then the same is true for any task t_k. In inconsistent *ETC* matrices, the relationships among the task computational requirements and machine capabilities are such that no structure as that in the consistent case is enforced. A combination of these two cases, which may be more realistic in many environments, is the partially consistent *ETC* matrix, which is an inconsistent matrix with a consistent sub-matrix [4]. In this study, the consistent sub-matrix was 25% of the tasks for 25% of the machines.

To simulate the diverse task mixtures in a real system, the COV for task and machine heterogeneity was set to 0.8. To ensure an oversubscribed system, and to provide a sufficient challenge for the mapping heuristics, the mean time to execute the tasks was set to 15 seconds. Also, the estimated arrival time of a new data set was set to five minutes, i.e., $\tau = 5$ minutes. Four different weighting of α (P_i for the medium priority tasks) were used in the experiments (1, 16, 256, and 4096) and 100 different *ETC*s were generated.

5. HEURISTICS FOR PHASE I

5.1 Overview

Six heuristics are discussed here. Genitor, a genetic algorithm approach that cannot be used in practice due to its long run time, was also implemented for comparison purposes.

5.2 Minimum Execution Time (MET)

The MET heuristic considers tasks in an arbitrary order, and maps the task t_i under consideration to the machine that has the smallest value of $ETC(i,j)$ for that task. The makespan using this heuristic remains unaltered if the ordering is changed.

5.3 Minimum Completion Time (MCT)

MCT considers the tasks in a given order. Each task is mapped to the machine that completes the task soonest, where, the completion time of the task t_i on machine j is the machine ready time for machine j plus the $ETC(i,j)$.

Because this heuristic considers the ready times of the machines, the order in which the tasks are considered for mapping influences the machine that a task is mapped on, effectively altering the finishing time of the individual machines, and the makespan. The average execution time of a task was calculated as:

$$avg_i = \frac{\sum_{1 \le j \le M} ETC(i,j)}{M}. \qquad (5)$$

Three variations are used to order the tasks: ascending and descending order of their average execution times, and arbitrary order.

5.4 K-Percent Best (KPB)

For this heuristic, a subset of the 'K Percent' fastest machines for a given task is selected, and the task is mapped to the machine in this subset that has the least completion time. A 'K' value of 0% causes this heuristic to coincide with MET, while 100% implies that the heuristic is same as MCT. Different values of K were explored, and it was found that the best results are obtained when K equal to 25%.

Because this heuristic makes the final decision based on MCT, different ordering of the tasks leads to different mappings, and effectively different makespan. Different orderings of tasks were considered as explained for the MCT heuristic.

5.5 Min-Min

Min-Min is a two-phase heuristic based on the completion time of the tasks. It can be described as follows:

1. Generate a task list of all the unmapped tasks.

2. For each task in the task list, find the machine that has the minimum completion time for that task (ignoring the other unmapped tasks).
3. For all the task-machine pairs found in step 2, select the pair that has the smallest minimum completion time.
4. Assign the selected task, remove it from the list of unmapped tasks, and update the ready time of the machine.
5. Repeat steps 2-4 until all the tasks have been mapped.

5.6 Max-Min

The Max-Min heuristic is similar to the Min-Min heuristic. However, instead of selecting the task-machine pair with the smallest minimum completion time, this heuristic selects the task-machine pair that has the largest of the minimum completion times. The intuition behind selecting the tasks with larger execution times is to decrease the penalty that such tasks would incur if they are not mapped on the best machines [7].

5.7 Genitor

The Genitor heuristic implemented here is a variation of Genitor described in [25]. The population size use for this study is 200 chromosomes, where a chromosome represents a valid mapping. The population is seeded with one chromosome generated using KPB "descending" variation, while the rest of the chromosomes are generated randomly. The population is sorted in the ascending order of makespan. For crossover, two parents are randomly selected, where the probability of selecting a chromosome is 10%, and a random cut-off point is generated. The machine assignments for the tasks from the bottom half of the chromosome are exchanged, and two new offspring are generated. For mutation, a single parent is selected (probability of selecting a chromosome is 25%) and the machine assignment of a single task is randomly changed. The offspring are inserted into the population and the worst chromosomes are taken out. The heuristic is stopped after one hour, and the best solution is selected.

6. HEURISTICS FOR PHASE II

6.1 Overview

For second phase of the problem, each machine ready time is the finishing time from phase I heuristic that minimizes t_c. Six heuristics were implemented for this phase. Of these, MET, MCT, KPB, Min-Min are the same as described in Section 4. The Max-Max heuristic uses the same concept as the Min-Min heuristic. A Genitor based heuristic was also implemented for comparison.

6.2 Greedy Heuristics

For the greedy heuristics, (MET, MCT, KPB), several different orderings of tasks were used. Because the finishing time of the individual task governs the likelihood, and hence the value of a task, different orderings can potentially produce different results for worth. The different orderings used for these heuristics are:

- Random ordering of tasks.
- Map the set of the medium priority tasks before the set of low priority tasks. Within each set of this partitioning, the tasks are considered for mapping in a random order.
- Tasks are ordered in descending order of priority per unit time (PT_i). To calculate the priority per unit time, the priority of the task t_i is divided by the smallest $ETC(i,j)$ time for that task.

$$PT_i = \frac{P_i}{\min_{1 \le j \le M} (ETC(i,j))}. \quad (6)$$

For each greedy heuristic, the variation that orders the tasks based on priority per unit time outperforms the other variations.

6.3 Max-Max

This heuristic uses the Min-Min concept, but based on value.

1. Generate a task list of all the unmapped tasks.
2. For each task in the task list, find the machine j that gives the maximum w_i based on the tasks that have been mapped already. Call this w_i value for machine j, $V_{i,j}$
3. For all the task-machine pairs found in step 2, select the pair that has the maximum value $V_{i,j}$.
4. Assign the selected task and remove it from the list of unmapped tasks, and update the finishing time of the machine.
5. Repeat steps 2 - 4 until all the tasks have been mapped.

Two variations of this heuristic are implemented. The first variation selects the task-machine pairs in step 2 by selecting the minimum completion time machine for each unmapped task. The second variation calculates value per unit time ($VT_{i,j}$).

$$VT_{i,j} = \frac{V_{i,j}}{ETC(i,j)}. \quad (7)$$

$VT_{i,j}$ is then used in steps 2 and 3 to map the tasks that complete before τ, while, for tasks that complete after τ, $V_{i,j}$ is used. The results for both the variations are comparable to result of Max-Max heuristic.

6.4 Genitor

For phase II, the order in which tasks are executed on a machine is important. Each chromosome can be viewed as a two-dimensional array, where each column of the array represents a machine, and the tasks to be executed on a machine are listed in the order in which they are executed. The population constitutes of a seed generated by the Max-Max heuristic, while 199 chromosomes are generated randomly. The population is sorted based on the worth of the mapping represented by the chromosomes. For a crossover operation, two parents are selected using a linear bias function [25]. The linear bias was varied between 1 and 2 in steps on 0.1 and finally a bias of 1.6 was used because it gave best results. For the selected parents a random cut-off point is generated, and the machine assignments and the positions of the tasks (in the machine queues) are exchanged. Consider a task t_i that has to be moved from machine j to machine k. If the new position of the task is beyond the last task on machine k, the task is placed at the end of the queue for machine k. However, if there if already a task scheduled at the new position, all the tasks on machine k are moved later in the queue, and the task is inserted at the desired position. All tasks on machine j that were scheduled to be executed after t_i are moved earlier in the machine queue for j. For mutation, a single parent is selected (probability of selecting a chromosome is 25%) and the machine assignment of a single task is randomly changed. The offspring have to compete for inclusion in the population as described in phase I, and the heuristic is stopped after one hour.

7. BOUND

7.1 Overview

A mathematical bound was calculated for both of the phases to compare against the performance of the heuristics. The method for calculating the bounds assumes a homogeneous MET system [24] in which the execution time for each task on all machines is the same and is equal to the minimum time that the task would take to execute across the original set of machines. The minimum execution time of task t_i, MET_i, is given by the following equation

$$MET_i = \min_{1 \le j \le M} (ETC(i, j)). \quad (8)$$

7.2 Lower bound on Makespan

For a lower bound on the makespan of the phase I tasks, we assume a homogeneous MET system. The lower bound (LB) on makespan for the high priority tasks is given by

$$LB = \frac{\sum_{\forall i} MET_i}{M}. \quad (9)$$

The calculation for the bound makes an assumption that each task can be executed on its MET machine, and that a single task can be split across multiple machines [7]. These assumptions are unrealistic and the bound is fairly loose.

7.3 Upper Bound on Worth

The method to find an upper bound on the worth of the medium and low priority tasks uses the following four steps:

Step 1: Assume a homogeneous MET system.

Let δ_j be the initial ready time of machine j, and β_j be the summation of the execution time for tasks mapped on machine j before task t_i is mapped. The likelihood for t_i is given by

$$l_i = \frac{\tau - ((\delta_j + \beta_j) + MET_i)}{\tau}. \quad (10)$$

The value of β_j for the homogeneous MET system is lesser than or equal to β_j for the original system (because of equation (8)), and therefore, the likelihood for the homogeneous MET system is greater than or equal to the likelihood of the original system.

Step 2: Assume that all machines are available as soon as the first machine to finish all its phase 1 tasks and that every task can be split into M equal parts that are executed in parallel across the M machines.

For phase II tasks, the initial machine ready time for each machine is given by the finishing time of that machine from phase I. Let machine k have the earliest ready time, i.e., $\delta_k = \min_{j \in M} \delta_j$. Hence, if one assumes that each machine is available at time δ_k, the worth of all the tasks mapped on a machine will be greater than or equal to the worth if the ready time of that machine is equal to δ_j. Likewise, if a task t_i is split across M machines, its effective execution time becomes MET_i / M so that its likelihood, l_i, is greater than the likelihood for that task if it is executed on a single machine.

Step 3: Assume that each medium priority task t_i is composed of α low priority tasks, each having execution time equal to MET_i / α.

Consider a medium priority task, t_i, whose priority, by definition, is equal to α. Let δ_j be the ready time of machine j. The worth of t_i on machine j is

$$w_i = \frac{\tau - (\delta_j + MET_i)}{\tau} \cdot \alpha . \quad (11)$$

If a medium priority task is broken into α low priority tasks and executed sequentially on a single machine, its worth $\overline{w_i}$ is given by

$$\overline{w_i} = \sum_{k=1}^{\alpha} \frac{\tau - (\delta_j + k \cdot MET_i / \alpha)}{\tau} = \frac{\tau - (\delta_j + \frac{(\alpha+1)}{2\alpha} MET_i)}{\tau} \cdot \alpha$$

i.e., that $\overline{w_i} > w_i$ for $\alpha \geq 1$.

Step 4: Let S be the sorted list of the medium priority tasks (each divided into α low priority tasks) and low priority tasks, arranged in ascending order of *MET* time. The tasks from this list are executed on the machines in the sorted order, and the worth is calculated.

From step 3, all the tasks have an equal priority, i.e., P_i =1. Let t_0, t_1, …, t_{n-1} represent the tasks in the sorted list S, such that, $MET_0 < MET_1 < ... < MET_{n-1}$. Because of the impact of the finishing time of the task on its likelihood, a task that is completed sooner has a higher likelihood. Therefore, $l_0 > l_1 > ... > l_{n-1}$, where, the likelihood, l_x, of a task t_x is given by

$$l_x = \frac{\tau - (\delta_j + MET_0 + MET_1 + ... + MET_x)}{\tau}. \quad (12)$$

Therefore, there cannot be a higher worth if the tasks are sorted in any other order.

If E_x is equal to the execution time of task t_x in the list S, find the largest y such that $\sum_{x=0}^{y-1} E_x \leq M \cdot (\tau - \delta_k)$. These y elements of S define the upper bound (*UB*). The other elements of S correspond to tasks that do not "fit" before the expected arrival time of the next data set, and are ignored. The likelihood, l'_x, of each task in S is calculated as

$$l'_x = \frac{\tau - (\delta_k + \beta_k + (E_x / M))}{\tau}. \quad (13)$$

Recall that all tasks are now low priority tasks (using step 3), and $w'_x = l'_x$. The upper bound on the worth of the medium and low priority tasks is

$$UB = \sum_{0 \leq x < y} w'_x. \quad (14)$$

8. EXPERIMENTAL RESULTS

All heuristics were run for 100 different trials (*ETC*s). The average values and 95% confidence intervals [16] were plotted. Heuristic execution times are shown in Tables 2 and 3.

Table 2: Average heuristic execution time per *ETC* for phase I.

heuristic	time (milliseconds)
MET	17
MCT	16
KPB	19
Min-Min	21
Max-Min	20
Genitor	3,600,000

Four different weighting of α (1, 16, 256, and 4096) were studied. All the results presented here are for α = 256. The relative performance of all the heuristics for all other values of α is similar.

Table 3: Average heuristic execution time per *ETC* for phase II.

heuristic	time (milliseconds)
MET	19
MCT	19
KPB	19
Min-Min	57
Max-Max	75
Genitor	3,600,000

The phase I results are shown in Figure 2. Among all the heuristics, the "descending" variation for KPB gave the best results. KPB "random," Min-Min, and MCT "descending" performed comparably to each other. The high makespan for MET can be accounted for by the partial consistency in the *ETC* matrices.

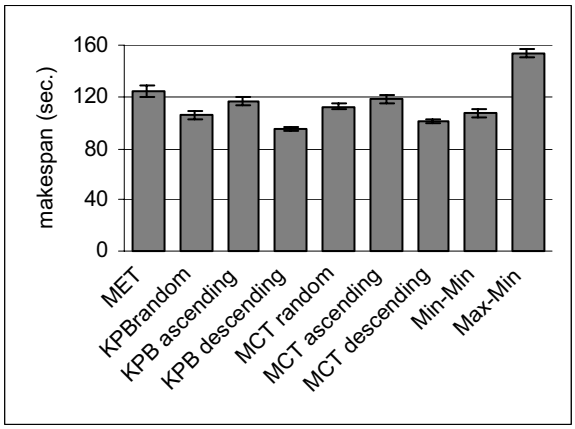

Figure 2: Makespan of the phase I tasks.

Figure 3 shows the worth of the phase II tasks for the greedy heuristics using different orderings and Figure 4 the compares makespan. For each heuristic, the variation that orders tasks based on priority per unit time outperformed the other two variations in terms of worth. The makespan for all three variations of MET was the same because the different orderings do not change the finishing time of the machines. The worth (Figure 3) is the performance metric. The makespan (Figure 4) is shown to contrast the heuristic makespan and worth properties.

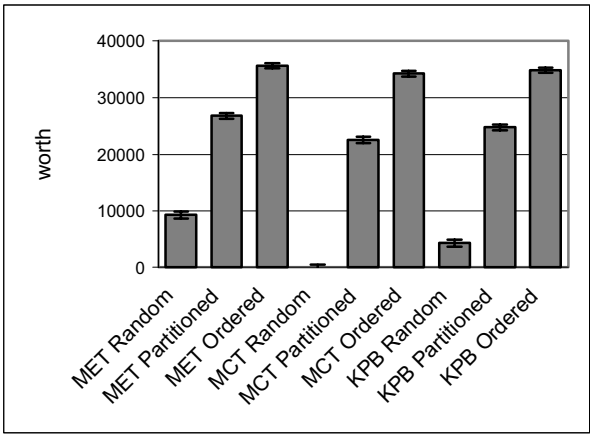

Figure 3: Variation in worth of greedy heuristics for phase II tasks by using different orderings (α = 256).

The worth results of the best greedy variations and the three other heuristics, and the upper bound are shown in Figure 5. Relative performance of all the heuristics (in terms of worth) for only the tasks that complete before τ was also studied, and was found to be similar to that using equation 4. Based on averages, Max-Max, MET "ordered," and KPB "ordered" performed the best. It can be noted that the Genitor heuristic performs comparably to Max-Max (which is used to seed Genitor), and does not significantly increase the worth of the seed. We hypothesize that this is because the Max-Max heuristic produces a near optimal schedule. As an example of Max-Max, for one of the trials, we measured what percentage of the tasks are assigned to their *MET* machines, and what was the load balance index for the mapping, defined as the ratio of the finishing times for the machine that finishes first to the machine that finishes last. The result was that 89% of the tasks were mapped to their *MET* machines, and the load balance index had a high value of 0.91.

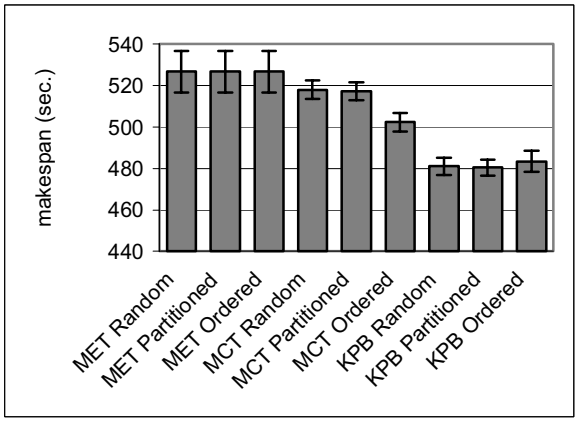

Figure 4: Variation in makespan of greedy heuristics for phase II by using different orderings (α = 256).

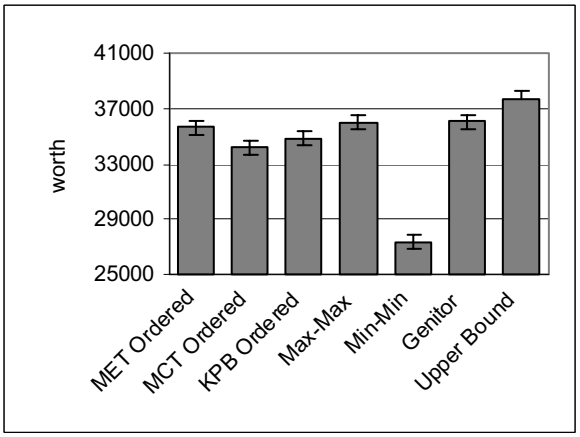

Figure 5: Worth of the phase II tasks (α = 256).

The makespan for different heuristics is shown in Figure 6. The worth (Figure 5) is the performance metric. The makespan (Figure 6) is shown to contrast the heuristic makespan and worth properties. Min-Min and Max-Max have comparable makespan; however, the worth of the mapping generated by Min-Min is lower than that of any other heuristics. This can be explained by the fact that it ignores the value of the individual tasks while mapping. Thus, it can be seen from the results that low makespan does not imply high worth and vice-versa.

9. SUMMARY

Several heuristics and their variations were implemented for each of the phases. The variation of KPB that orders tasks in descending order of their average execution time performs the best for phase I,

and KPB "ordered," MET "ordered," and Max-Max performed best (in terms of average) for phase II. The Genitor variations give a slight average improvement over the seed, but run time is significantly longer. Because the time for the scheduler is limited in this study, KPB "descending" variation and Max-Max heuristic are recommended for the given problem.

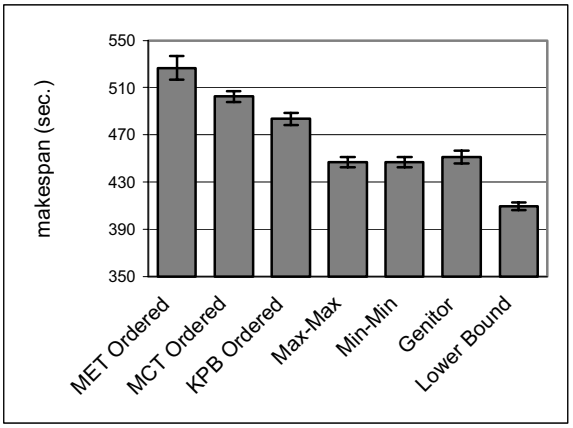

Figure 6: Makespan of the phase II tasks (α = 256).

ACKNOWLEDGMENT

The authors thank Mr. Ashish Mehta and Mr. Vladimir Shestak for their valuable comments.

REFERENCES

[1] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "Characterizing resource allocation heuristics for hetserogeneous computing systems," in *Advances in Computers Volume 63: Parallel, Distributed, and Pervasive Computing*, A. R. Hurson, ed., Elsevier, Amsterdam, 2005, pp. 91-128.

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

[3] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 7, July 2004, pp. 630–641.

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

[5] C. Artigues, J. Billaut, C. Esswein, "Maximization of solution flexibility for robust shop scheduling," *European Journal of Operational Research*, Vol. 165, No. 2, 2005 , pp. 314–328.

[6] L. Barbulescu, A.E. Howe, L.D. Whitley, and, M. Roberts, "Trading place: How to schedule more in a multi-resource oversubscribed scheduling problem system," *International Conference on Automated Planning and Scheduling (ICAPS-04)*, June, 2004.

[7] 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.

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

[9] 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.

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

[11] 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.

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

[13] 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.

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

[15] 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.

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

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

[18] L.A. Kramer and S.L. Smith, "Maximizing flexibility: A retraction heuristic for oversubscribed scheduling problems," *Eighteenth International Joint Conference on Artificial Intelligence*, Aug., 2003.

[19] A. M. Mehta, J. Smith, H. J. Siegel, A.A. Maciejewski, A. Jayaseelan, and B. Ye, "Dynamic resource allocation heuristics for maximizing robustness with an overall makespan constraint in an uncertain environment," *2006 International*

Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA 2006), June 2006, accepted to appear.

[20] A. M. Mehta, J. Smith, H. J. Siegel, A.A. Maciejewski, A. Jayaseelan, and B. Ye, "Dynamic resource allocation heuristics for minimizing makespan while maintaining an acceptable level of robustness in an uncertain environment," *12th International Conference on Parallel and Distributed Systems*, July 2006, accepted to appear.

[21] National Environmental Satellite Data Information Service (NESDIS), http://www.nesdis.noaa.gov/About/about.html, accessed March 2, 2006.

[22] V. Shestak, J. Smith, R. Umland, J. Hale, P. Morranville, A. A. Maciejewski, and H.J. Siegel, "Greedy approaches to stochastic robust resource allocation for periodic sensor driven distributed systems," *2006 International Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA 2006)*, June 2006, accepted to appear.

[23] V. Shestak, J. Smith, H. J. Siegel, and A.A. Maciejewski, "A stochastic approach to measuring the robustness of resource allocations in distributed systems," *2006 International Conference on Parallel Processing (ICPP 2006)*, Aug. 2006, accepted to appear.

[24] 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.

[25] D. Whitley, "The GENITOR algorithm and selective pressure: Why rank based allocation of reproductive trials is best," *3rd International Conference on Genetic Algorithms*, June 1989, pp. 116–121.

[26] 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.