Heuristics for Robust Resource Allocation of Satellite Weather Data Processing on a Heterogeneous Parallel System

Luis Diego Briceño, *Member*, *IEEE*, Howard Jay Siegel, *Fellow*, *IEEE*, Anthony A. Maciejewski, *Fellow*, *IEEE*, Mohana Oltikar, Jeff Brateman, Joe White, Jonathan R. Martin, and Keith Knapp

Abstract—This work considers the satellite data processing portion of a space-based weather monitoring system. It uses a heterogeneous distributed processing platform. There is uncertainty in the arrival time of new data sets to be processed, and resource allocation must be robust with respect to this uncertainty. The tasks to be executed by the platform are classified into two broad categories: high priority (e.g., telemetry, tracking, and control), and revenue generating (e.g., data processing and data research). In this environment, the resource allocation of the high-priority tasks must be done before the resource allocation of the revenue generating tasks. A two-part allocation scheme is presented in this research. The goal of first part is to find a resource allocation that minimizes makespan of the high-priority tasks. The robustness for the first part of the mapping is defined as the difference between this time and the expected arrival of the next data set. For the second part, the robustness of the mapping is the difference between the expected arrival time and the time at which the revenue earned is equal to the operating cost. Thus, the heuristics for the second part find a mapping that minimizes the time for the revenue (gained by completing revenue generating tasks) to be equal to the cost. Different resource allocation heuristics are designed and evaluated using simulations, and their performance is compared to a mathematical bound.

Index Terms—Heterogeneous computing, satellite system, robustness, makespan, revenue, and two-part resource allocation.

1 Introduction

The space-based weather monitoring system considered in this work consists of two major components: the satellite with its data collection sensors, and the data processing system (see Fig. 1). The data processing system is responsible for requesting (from the satellite) the data that must be collected, and scheduling the tasks that need to be executed. The tasks to be executed on the data set can be classified into two broad categories: 1) high-priority tasks for positioning and 2) revenue generating data processing and data research tasks [26]. Imaging across a variety of spectral bands is collected by the satellite, and is transmitted back to the data processing system.

The weather imaging data sent down by the satellite (the **data set**) must be processed before it is of any value to the users. A new weather data set is received periodically, and the current data set must be processed before the next data set arrives. A similar requirement is used in the satellite image processing in [19]. Currently, systems used for processing the data sets, at a typical site, are divided into three distinct sets of processing elements (dedicated to satellite positioning, data processing, and data research). As a result of this partitioning of a given data set, one system may be overloaded while another is underloaded.

The goal of this research is to develop a resource manager, so that a smaller heterogeneous global bank of shared common resources can replace the three sets of processing elements and operate efficiently. The global bank will reduce the cost of the system, while being financially viable. This platform is a heterogeneous computing system (**HCS**), because machines are typically added or replaced over time with new machines. Therefore, tasks may have different execution times on different machines, and thus have greater affinity to certain machines.

The allocation of tasks to machines is a static mapping problem [1], because all the tasks that need to be executed are known a priori (before the data set to be processed arrives). However, it has some characteristics of dynamic mapping in that tasks are known a short time in advance, so

- *L.D. Briceño, A.A. Maciejewski, K. Knapp, and H.J. Siegel are with the Department of Electrical and Computer Engineering, Colorado State University, Engineering Room B104, 1373 Campus Delivery, Fort Collins, CO 80523-1373. E-mail: {ldbricen, aam, Keith.Knapp, hj}@colostate.edu.*
- *M. Oltikar is with the Department of Electrical and Computer Engineering, Colorado State University, Engineering Room B104, 1373 Campus Delivery, Fort Collins, CO 80523-1373, and Hughes Network Systems, LLC. E-mail: mohana.oltikar@colostate.edu.*
- *J. Brateman is with the Department of Electrical and Computer Engineering, Colorado State University, Engineering Room B104, 1373 Campus Delivery, Fort Collins, CO 80523-1373, and IBM, Austin, TX. E-mail: Jeff.Brateman@colostate.edu.*
- *J. White is with the Department of Electrical and Computer Engineering, Colorado State University, Engineering Room B104, 1373 Campus Delivery, Fort Collins, CO 80523-1373, and Recondo Technology. E-mail: Joe.White@colostate.edu.*
- *J.R. Martin is with the Department of Electrical and Computer Engineering, Colorado State University, Engineering Room B104, 1373 Campus Delivery, Fort Collins, CO 80523-1373, and R.L. Martin & Associates. E-mail: jrmartin@jrmartin.com.*

Manuscript received 26 Oct. 2009; revised 22 July 2010; accepted 18 Nov. 2010; published online 19 Jan. 2011.
Recommended for acceptance by K. Li.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-10-0529.
Digital Object Identifier no. 10.1109/TPDS.2011.44.

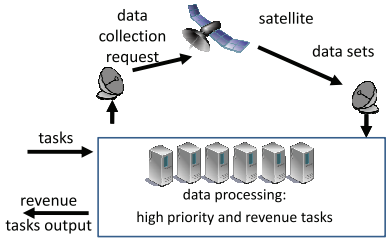


Fig. 1. Overview of a space-based weather system.

the resource allocation must be performed in a short amount of time, e.g., five minutes.

In an HCS, the assignment of tasks to machines to obtain a near optimal resource allocation is an important research problem. 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**. The mapping problem has been shown, in general, to be NP-complete (e.g., [9], [14], [17]). Hence, the development of heuristic techniques to find near optimal solutions is an active area of research (e.g., [1], [13], [15], [31]).

The performance of computing systems is susceptible to degradation due to unpredictable circumstances. Therefore, it is necessary to allocate resources to tasks so that the robustness of the system in response to unpredictable events is maximized [4]. For this study, the times between the arrival of data sets can vary, i.e., it is uncertain. The next data set may arrive earlier than expected. Because the current data set is discarded when the next data set arrives, it is important for the resource allocation to be robust against an early arrival of the next data set. Thus, it is necessary to develop a performance metric to evaluate the robustness of a mapping produced by the heuristics.

In this research, our contributions are:

1. the derivation of a formal mathematical model for a proposed real-world weather image processing system;
2. the definition of a new robustness metric for making resource allocation decisions;
3. the development of a two-part approach for scheduling **high-priority tasks** (**HPTs**) and **revenue generating tasks** (**RGTs**) in an oversubscribed system where each part has a different robustness criterion;
4. the design and simulation-based evaluation of new problem-domain-specific heuristics for developing resource allocations; and
5. the derivation of a bound on the performance of a resource allocation for the proposed HCS.

The remainder of this paper is organized as follows: A detailed overview of the system model is given in Section 2. Section 3 discusses the related work. The heuristics for HPT and RGT are explained in Sections 4 and 5, respectively. Section 6 describes the simulation setup used for the experiments, and the bounds on the performance of a resource allocation are presented in Section 7. The experimental results are discussed in Section 8. In Section 9, the conclusions are presented.

2 Problem Statement

2.1 System Model

In this study, there are a set of T tasks that must be executed on M heterogeneous machines for a *given* data set. We assume that all the tasks associated with a data set must arrive at a predetermined 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 arrives from the satellite after an interval of τ time units. The HPTs executed with this new data set are needed to position the satellite, i.e., to decide where to move the satellite next.

In some scenarios dealing with large-image data sets, it may difficult to keep multiple data sets from multiple time intervals. Furthermore, for the case where the same geographical location is being monitored, it is better to use the latest data. Therefore, when a new data set arrives, all tasks associated with the old data set are dropped, and the machine queues are emptied. In our environment, we will need to stop executing RGTs using the last data set to execute the HPTs on the new data set.

In this system, not all tasks (for the current data set) can be completed by the expected arrival time of the next data set. An oversubscribed system is considered because it makes meeting robustness constraints more difficult to accomplish.

The expected arrival time of the next data set, $\tau_{expected}$, is only an estimate and the next data set might arrive earlier than expected. The estimated time to compute task i on machine j ($\boldsymbol{ETC}(i,j)$) is assumed to be known. The assumption of such ETC information (based on historical or experimental data) is a common practice in resource allocation research (e.g., [16], [18], [20], [23], [32]). Let the **machine ready time** be the time at which a machine would be able to start the execution of a currently unassigned task.

Execution of the HPTs ensure the proper functioning of the system; therefore, it is necessary that these tasks are completed. The RGTs are important to ensure the system is financially viable. The resource allocation is separated into two parts. The first part is concerned with minimizing the maximum completion time (**makespan**) of all HPTs, while the second part is concerned with minimizing the time which it takes to reach a "profit" and is different from just minimizing makespan, as different tasks have different revenues.

Let $\boldsymbol{makespan_{HP}}$ be the completion time of the last high-priority task to finish, $\mathbf{Dataset}_i$ the time when data set i arrives, and $\Delta\tau = \tau_{expected} - \tau$. An illustration of this notation is shown in Fig. 2.

2.2 Robustness

Features, Perturbations, Impact, and Analysis (**FePIA**) is a procedure for deriving a robustness metric for an HCS [4]. The FePIA procedure addresses three fundamental questions [3]: 1) What behavior of the system makes it robust? 2) What uncertainties is the system robust against? and 3) Quantitatively, exactly how robust is the system?

Using the FePIA procedure, we define what behavior makes the first part robust. This system needs to be robust against uncertainty in the arrival time of the next data set. For HPTs, a resource allocation is robust if all HPTs finish before

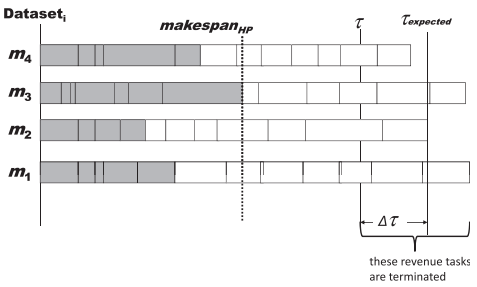


Fig. 2. Illustration of a resource allocation that includes both HPT and RGTs. In this figure, gray rectangles represent high priority and white rectangles represent RGTs.

$\tau_{expected}$, and the robustness is quantified by the difference between $\tau_{expected}$ and the makespan of HPTs ($makespan_{HP}$). By minimizing $makespan_{HP}$, we maximize time difference between the maximum completion time of these tasks and $\tau_{expected}$. The robustness for HPTs (ρ_{HPT}) is quantified as

$$\rho_{HPT} = \tau_{expected} - makespan_{HP}. \quad (1)$$

This measure of robustness for HPTs is similar to a slack (or laxity) (e.g., [11], [29]).

RGTs generate a revenue, but are not critical for positioning the satellite. Each RGT task i (t_i) has an associated revenue of $R(t_i)$. The revenue generated by the satellite should cover the costs associated with the continued operation of the satellite (e.g., labor and facilities).

For the system to be robust, the revenue generated by the satellite for each data set needs to be larger than the cost associated with processing that data set. Let C_{total} be the cost associated with processing a data set, and $CTasks(\tau)$ be the set of RGTs completed before the arrival of the next data set; this set is a function of τ. The system is robust if

$$C_{total} \leq \sum_{\forall t_i \in CTasks(\tau)} R(t_i). \quad (2)$$

The uncertainty considered when assigning both HPTs and RGTs is the actual arrival time τ of the next data set. The robustness metric for RGTs (ρ_{RGT}) is the maximum value of $\Delta\tau$ ($= \tau_{expected} - \tau$) such that the revenue is equal to or exceeds the cost, and is calculated using the following equation:

$$\begin{aligned} &\rho_{RGT} = \max \Delta\tau \\ &\text{Such that} \\ &C_{total} \leq \sum_{\forall t_i \in CTasks(\tau_{expected} - \Delta\tau)} R(t_i). \end{aligned} \quad (3)$$

Finding a resource allocation that maximizes $\Delta\tau$ will optimize robustness because it allows the next data set to arrive sooner and still meet the system cost (C_{total}) associated with processing a data set.

3 Related Work

There are many definitions of robustness for various environments (e.g., [4], [5], [25], [28]). Our study applies the robustness concept to an environment that is required to be profitable, where revenue is earned by executing tasks.

The study in [5] discusses a job shop environment that is susceptible to sudden changes that render an existing schedule infeasible. In our study, the system model and performance metrics are quite different.

In [6], the authors describe an oversubscribed system for scheduling communications for a satellite range scheduling problem. 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 complete before their deadline. This problem is similar to our study, because we schedule based on task priority, i.e., assign HPTs first, and also the money generated by RGTs can be considered as a priority. However, in our study, the profitability of the system is robust against uncertainties in the arrival time of the next data set. This does not necessarily correlate to minimizing the number of tasks that cannot complete before the deadline.

The work in [21] also discusses an oversubscribed environment of tasks with multiple priorities, but 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. Our study is similar because it is divided into a two-part scheduling problem, where the HPTs must be completed before the RGTs can be considered. However, for RGTs there is a trade-off. Another significant difference between our work and the work in [21] is that we study how the system performs when there is uncertainty in the arrival time of the next data set.

4 Heuristics for High-Priority Tasks

Six static heuristics are considered here: five greedy heuristics and a Genitor Algorithm. We implement two types of greedy heuristics: one-phase and two-phase. The one-phase heuristics for HPTs are Minimum Execution Time (MET) [7], [24], Minimum Completion Time (MCT) [7], [17], and K-Percent Best (KPB) [24], and the two-phase heuristics are MinCT-MinCT and MaxCT-MinCT [7], [17], [24]. These heuristics were chosen because they have performed well in similar environments, and were appropriate for the goals and time constraints of this problem domain, where the goal is to minimize makespan. Because the use of these heuristics for minimizing makespan was taken directly from the earlier work cited above, they are not discussed here.

A Genitor algorithm [30] (a steady-state genetic algorithm) was implemented in this study for comparison purposes. Genetic algorithms have been used successfully in the literature for resource allocation (e.g., [8], [12], [22]). This Genitor approach cannot be fielded in a live system because of time constraints on the heuristic runtime. The implementation details of greedy and Genitor heuristics are in Appendix A (see supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.44).

5 Heuristics for Revenue Tasks

5.1 Overview

For the second part of the problem, each machine's ready time is its finishing time found by the best heuristic used for mapping HPTs. Several heuristics were implemented for RGTs. Of these, MET, MCT, KPB, and MinCT-MinCT are the same as used in Section 4 (see Appendix A in supplemental material), except with descending revenue being used to

(1) Generate a list of all the unmapped tasks.
(2) For each task in the list, find the machine j that gives the maximum worth.
(3) For all task-machine pairs found in step (2), select pair that has the maximum worth value.
(4) Assign the selected task, remove it from the list, and update the ready time of the machine.
(5) Repeat steps (2) – (4) until all the tasks have been mapped.

Fig. 3. Procedure for using MaxW-MaxW to generate a resource allocation.

determine the order in which tasks are assigned for one-phase greedy heuristics. The MaxW-MaxW, MaxWPTU-MaxWPTU, and MaxWPTU-MinCT heuristics are greedy heuristics similar in structure to the MinCT-MinCT heuristic but using different objective functions. A Genitor-based heuristic also was implemented for comparison only, due to its long execution time.

5.2 Greedy Heuristics

For the one-phase greedy heuristics (MET, MCT, and KPB), three different orderings of tasks were used: random orderings (**RAND**), maximum revenue per time unit (**MRPTU**), and average revenue per time unit (**ARPTU**).

1. RAND: Random ordering of tasks.
2. MRPTU: For each task i that needs to be mapped, $MRPTU_i$ is calculated using the following equation:

$$MRPTU_i = \frac{R(t_i)}{\min_{1 \le j \le M} ETC(i,j)}. \quad (4)$$

   The tasks that need to be mapped are sorted in descending order based on $MRPTU_i$.
3. ARPTU: For each task i that needs to be mapped, $ARPTU_i$ is calculated using the following equation:

$$ARPTU_i = \frac{R(t_i)}{\left[\sum_{1 \le j \le M} ETC(i,j)\right]/M}. \quad (5)$$

   The tasks that need to be mapped are sorted in descending order based on $ARPTU_i$.

5.3 MaxW-MaxW

This heuristic is similar in structure to MinCT-MinCT, but instead of minimizing completion time it maximizes a "worth" value. Let F_{ij} be the completion (finishing) time of task i on machine j. For a task i on a machine j, the likelihood (L_{ij}) of task i completing on machine j before the deadline is defined as

$$L_{ij} = \frac{\tau_{expected} - F_{ij}}{\tau_{expected}}. \quad (6)$$

The worth value (w_{ij}) is based on L_{ij} and is calculated using the following equation:

$$w_{ij} = L_{ij} \cdot R(t_i). \quad (7)$$

The procedure used to implement MaxW-MaxW is shown in Fig. 3.

5.4 MaxWPTU-MaxWPTU

The MaxWPTU-MaxWPTU heuristic is a Max-Max heuristic similar in structure to MaxW-MaxW. However, this heuristic is based on worth per time unit (**WPTU**), calculated as follows:

$$wptu_{ij} = \frac{w_{ij}}{ETC(i,j)}. \quad (8)$$

The procedure for MaxWPTU-MaxWPTU is the same as the procedure shown in Fig. 3; however, "worth per time unit" is substituted for "worth."

5.5 MaxWPTU-MinCT

The MaxWPTU-MinCT heuristic is similar to the MaxW-MaxW heuristic previously described. MinCT finds for each unmapped task the task/machine pair with the smallest completion time, then MaxWPTU selects the task/machine pair with the maximum WPTU.

5.6 RGT Genitor

Genetic algorithms (**GAs**) have been shown to work well for numerous problem domains, including resource allocation and job shop scheduling, e.g., [7], [8], [10]. The RGT Genitor requires information about the assignment of tasks to machines, and the ordering of these tasks in the machine queue. This information is represented with two chromosome strings that are illustrated in Fig. 4a. The top string represents the assignment of t_i to machine j. The bottom string is a real number from 0 to 1 that represents the relative ordering of a task in a machine queue. In Fig. 4b, the string from Fig. 4a is converted to a mapping. To understand how the chromosome string is converted into a mapping, we can observe the tasks assigned to machine 1 (m_1). The task-real number pairs assigned to m_1 are: $t_4 - 0.74$, $t_7 - 0.23$, $t_{11} - 0.34$, and $t_{12} - 0.99$. The real number is used to arrange the tasks in ascending order within the machine queue (e.g., t_{11} executes before t_4 because $0.34 < 0.74$).

The best solution generated among the greedy heuristics was used as a seed in the RGT Genitor. The rest of the population is created by generating a random assignment of

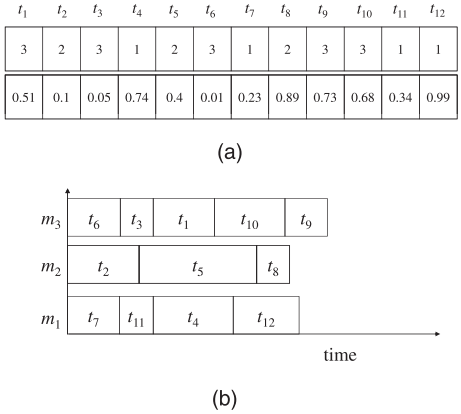


Fig. 4. Chromosome representation for RGT Genitor: (a) chromosome strings, (b) machine queues.

tasks to machines, and a uniform random variable ($U(0,1)$) for the relative ordering for each task. The chromosomes are sorted in a ranked list based on descending order of ρ_{RGT}.

The crossover for the RGT Genitor is done by selecting two parents using linear bias [30]. For the two selected parents, two crossover points are randomly generated. Between these two crossover points, the machine assignments and the real numbers are exchanged among parents. This crossover procedure generates two new offspring.

The mutation is done on both offspring. For each entry i in the machine assignment string of the offspring, with X percent probability of mutation (determined experimentally) reassign task i to a randomly selected machine. After the machine assignment string is mutated, the random number string is mutated. For each entry i of the random number string of the offspring, with X percent probability of mutation (determined experimentally), a new random number ($U(0,1)$) is generated and it replaces the entry for task i.

After the crossover and mutation operations are done, the offsprings are evaluated and inserted into the sorted population; the two worst chromosomes are discarded from the population (i.e., the size of the population remains constant). This process is repeated until the stopping criteria is met (i.e., heuristic execution time reaches one hour). Based on experimentation: the population size used for this study was 100 chromosomes, the probability of mutating a task-machine assignment was set to 3 percent, the probability of crossover was 100 percent, and the linear bias parameter is set to 1.5.

6 Simulation Setup

The simulation environment was intended to represent a typical satellite data processing system. The environment we used to evaluate and compare the heuristics had eight machines ($M = 8$), and 2,048 total tasks (subdivided into 512 HPTs and 1,536 RGTs). In this environment, users submit requests to process a provided data set by one of a collection of well-known tasks. For this simulation, the ETC matrix is generated using the coefficient of variance (**COV**)-based method, described in [2].

To simulate the diverse task mixtures in a real system, the COV for task heterogeneity was 0.1 and machine heterogeneity was set to 0.4, i.e., low task-high machine heterogeneity. The simulation parameters were configured to ensure an oversubscribed system, and provide a sufficient challenge for the mapping heuristics. The mean time to execute the tasks was set to 7.5 seconds, and $\tau_{expected} = 900$ seconds. The execution ratio of a task (ERT_i) is the average execution time of a task i over all machines divided by the average execution time of all tasks across all machines.

In an actual system, revenue for each task is negotiated between the system provider and the user. For our simulation studies, the $R(t_i)$ values for revenue task i are computed by multiplying ERT_i and a sample from a Gamma distribution (mean of 200 and standard deviation of 50).

The simulations were run on an Intel Core 2 Duo T8100 (2.1 GHz), with 4 GB RAM running a Windows Vista OS. The code was written in C++ and run on cygwin.

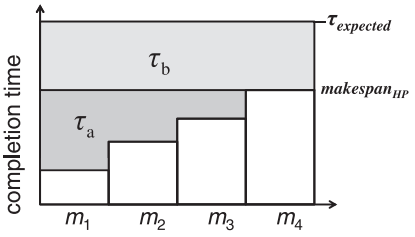


Fig. 5. Illustration of τ_a and τ_b.

7 Bound

In this section, we present an upper bound on ρ_{RGT}, assuming an oversubscribed system that can have a revenue that exceeds cost. An upper bound on ρ_{HPT} is from [7] and is summarized in Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.44. The upper bound on ρ_{RGT} (UB_{RGT}) uses the best result found from among the heuristics used to assign HPTs to set the initial machine ready times for the RGTs. This method is used (instead of using the result of UB_{HPT}) to make the bound tighter.

Let $\boldsymbol{Set_{revenue}}$ be the set of RGTs. For each revenue task ($t_i \in Set_{revenue}$), $MRPTU_i$ is calculated using (4). A list (UB_{list}) of the RGTs sorted based on $MRPTU_i$ in descending order is created. To calculate UB_{RGT}, we define CT_j as the completion time of machine j, and task i (t_i) as the ith task from UB_{list}. Let

$$MET(t_i) = \min_{1 \le j \le M} ETC(i,j). \tag{9}$$

The N_{min} is the minimum number of RGTs from UB_{list} needed for the revenue to be larger than the cost, and τ_{total} is the minimum time that the minimum execution time machines would need to achieve exactly C_{total}. That is,

$$N_{min} = \min Y : C_{total} \le \sum_{i=1}^{Y} R(t_i), \tag{10}$$

$$\tau_{total} = \sum_{i=1}^{N_{min}-1} [MET(t_i)] + MET(t_{N_{min}}) \cdot \frac{C_{total} - \sum_{i=1}^{(N_{min}-1)} R(t_i)}{R(t_{N_{min}})}. \tag{11}$$

As illustrated in Fig. 5,

$$\tau_a = \sum_{j=1}^{M} \left(makespan_{HP} - CT_j\right), \tag{12}$$

$$\tau_b = (\tau_{expected} - makespan_{HP}) \cdot M. \tag{13}$$

$$\text{If } \tau_{total} \le \tau_a, \text{then } UB_{RGT} = \rho_{HPT}. \tag{14}$$

$$\text{If } \tau_a < \tau_{total} \le \tau_a + \tau_b, \text{then } UB_{RGT} = \tau_{expected} - \frac{\tau_a + \tau_b - \tau_{total}}{M}. \tag{15}$$

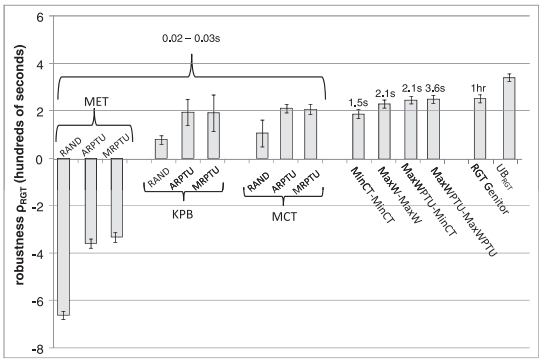


Fig. 6. Robustness for RGT, averaged over 100 trails, for inconsistent heterogeneity. The 95 percent confidence intervals are shown. The heuristic execution times are shown above the results.

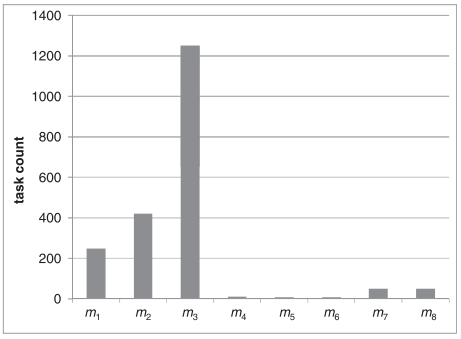


Fig. 7. Sample histogram of number of tasks that have machine j as its minimum execution time machine for inconsistent heterogeneity.

If the system is not robust (i.e., $\tau_{total} > \tau_a + \tau_b$), then

$$UB_{RGT} = \frac{\tau_a + \tau_b - \tau_{total}}{M}. \quad (16)$$

This value can be used to determine the minimum time past $\tau_{expected}$ needed to make revenue equal to cost.

8 Results

Different types of heterogeneity (consistencies [2]) are defined in Appendix C, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.44. The results from HPTs are given in Appendix D. The results of RGT for the inconsistent matrices are shown in Fig. 6 and discussed below (additional RGT results are in Appendix E of the supplemental material).

The MaxWPTU-MaxWPTU was the best performing heuristic. This heuristic uses the main characteristics of the MaxW-MaxW and improves upon it by calculating the worth per time unit for each specific task-machine pair. Note that both MaxWPTU-MaxWPTU and upper bound for RGTs use a revenue over execution-time-based function. The equations for $MRPTU_i$ (4) and $wptu_{ij}$ (8) are similar for the MET machine of task i. The difference comes when we incorporate the likelihood of completing the task before $\tau_{expected}$ in (8). This inclusion allows the heuristic to incorporate information about the current state of the machine that is not present in (4).

The result of the MaxWPTU-MaxWPTU was over 98.7 percent of the robustness value generated by the Genitor heuristic even though Genitor is seeded with the best of the heuristics for each trial and has a runtime of 60 minutes.

We believe that the solution found by MaxWPTU-MaxWPTU is near optimal; because, the Genitor heuristic (in general) did not find a significantly better solution and the UB is loose. The UB is loose because, in the low task-high machine heterogeneity case, the number of tasks that have a given machine j as its minimum execution time machine is not evenly distributed across machines (as shown in Fig. 7). As a result, not all tasks execute on their MET machine, as assumed in the upper bound, resulting in a somewhat loose bound.

It was interesting to observe that the MET heuristic did not perform well in environments with low task-high machine inconsistent heterogeneity. In previous studies (e.g., [7]), MET performed similarly to MCT with inconsistent heterogeneity. However, in this experiment, its performance was significantly inferior to MCT due to the low task-high machine heterogeneity.

One-phase greedy heuristics were significantly faster than the two-phase heuristics. The average execution time of the one-phase heuristics was less than 0.03 s, while the quickest two-phase greedy heuristic had a runtime of 1.4 s. Both MaxW-MaxW and MaxWPTU-MinCT had a runtime of approximately 2.11 s; this shows that calculating the worth per time unit, as opposed to just the worth, does not increase the runtime significantly. The results of our experiments in these heterogeneities are presented in Appendices D (HPTs) and E (RGTs), which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.44.

9 Conclusions

The environment considered for this study is a computer system that processes weather data. Additionally, this research also could apply to different sensor data processing, e.g., battleship radar and homeland surveillance security. The satellite data system has two parts that need to be robust against uncertainty. The first part is the execution of high-priority tasks to ensure the operation of the satellite, and the second part ensures financial viability of the system. The uncertainty considered for this system is the variability in the time interval between the arrival of successive data sets.

Several heuristics and their variations were implemented for each of the parts. For all the evaluated scenarios, for HPT, the best heuristic was MinCT-MinCT, and for RGTs MaxWPTU-MaxWPTU was either the best or had comparable performance.

An example of possible extensions could be: tasks execution times could vary based on the input data sets, possible machine failures, representing the entries in an ETC as a probability mass function, use of different revenue models, computing platforms with different heterogeneities, variation in the number of tasks that need to be executed, and applications to other important sensor problems. Another extension to this work is to incorporate good will or fairness when assigning resources. This fairness could become part of the robustness metric or a Quality-of-Service constraint. Additionally, the same model could be used to simulate a multicore or multithreaded environment. However, to do this accurately, we would need to incorporate

memory hierarchy and sharing. Another possible direction for future work would be the design of fast heuristics that can combine the makespan measure of HPTs and the revenue of RGTs into the creation of composite HPT and RGT resource allocations, so that an allocation for the HPTs that is "close" to the best makespan can be considered if it gives "much more" revenue for the RGTs; defining "close" and "much more" may need to be user-specified.

APPENDIX

For the appendices, please refer to the online supplemental material, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2011.44.

ACKNOWLEDGMENTS

A preliminary version of portions of the "high priority" material was presented at Eighth Workshop on High Performance Scientific and Engineering Computing [27]. The authors would like to thank Paul Maxwell, Abdulla Al-Qawasmeh, Jerry Potter, Ricky Kwok, and the anonymous reviewers for all their help. This research was supported by the US National Science Foundation (NSF) under grants CNS-0615170 and CNS-0905399, and by the Colorado State University George T. Abell Endowment.

REFERENCES

[1] S. Ali, T.D. Braun, H.J. Siegel, A.A. Maciejewski, N. Beck, L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, and B. Yao, "Characterizing Resource Allocation Heuristics for Heterogeneous Computing Systems," *Advances in Computers: Parallel, Distributed, and Pervasive Computing,* vol. 63, pp. 91-128, Academic Press, 2005.

[2] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems," *Tamkang J. Science and Eng.,* special 50th anniversary issue, vol. 3, no. 3, pp. 195-207, Nov. 2000.

[3] S. Ali, A.A. Maciejewski, and H.J. Siegel, "Perspectives on Robust Resource Allocation for Heterogeneous Parallel Systems," *Handbook of Parallel Computing: Models, Algorithms, and Applications,* S. Rajasekaran and J. Reif, eds., pp. 1-30, Chapman & Hall/CRC Press, 2008.

[4] S. Ali, A.A. Maciejewski, H.J. Siegel, and J.-K. Kim, "Measuring the Robustness of a Resource Allocation," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 7, pp. 630-641, July 2004.

[5] C. Artigues, J. Billaut, and C. Esswein, "Maximization of Solution Flexibility for Robust Shop Scheduling," *European J. Operational Research,* vol. 165, no. 2, pp. 314-328, 2005.

[6] L. Barbulescu, A.E. Howe, L.D. Whitley, and M. Roberts, "Trading Places: How to Schedule More in a Multi-Resource Oversubscribed Scheduling Problem System," *Proc. Int'l Conf. 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 B. Yao, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing,* vol. 61, no. 6, pp. 810-837, June 2001.

[8] R. Cheng, M. Gen, and Y. Tsujimura, "A Tutorial Survey of Job-Shop Scheduling Problems Using Genetic Algorithms—I. Representation," *Computers and Industrial Eng.,* vol. 30, no. 4, pp. 983-997, 1996.

[9] E.G. Coffman, *Computer and Job-Shop Scheduling Theory.* John Wiley and Sons, 1976.

[10] F.D. Croce, R. Tadei, and G. Volta, "A Genetic Algorithm for the Job Shop Problem," *Computers and Operations Research,* vol. 22, no. 1, pp. 15-24, 1995.

[11] A.J. Davenport, C. Gefflot, and J.C. Beck, "Slack-Based Techniques for Robust Schedules," *Proc. Sixth European Conf. Planning,* pp. 7-18, Sept. 2001.

[12] M.K. Dhodi, I. Ahmad, and I. Ahmad, "A Multiprocessor Scheduling Scheme Using Problem-Space Genetic Algorithms," *Proc. IEEE Int'l Conf. Evolutionary Computation,* pp. 214-219, 1995.

[13] M.M. Eshaghian, *Heterogeneous Computing.* Artech House, 1996.

[14] D. Fernandez-Baca, "Allocating Modules to Processors in a Distributed System," *IEEE Trans. Software Eng.,* vol. SE-15, no. 11, pp. 1427-1436, Nov. 1989.

[15] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1999.

[16] A. Ghafoor and J. Yang, "A Distributed Heterogeneous Supercomputing Management System," *Computer,* vol. 26, no. 6, pp. 78-86, June 1993.

[17] O.H. Ibarra and C.E. Kim, "Heuristic Algorithms for Scheduling Independent Tasks on Non-Identical Processors," *J. ACM,* vol. 24, no. 2, pp. 280-289, Apr. 1977.

[18] M. Kafil and I. Ahmad, "Optimal Task Assignment in Heterogeneous Distributed Computing Systems," *IEEE Concurrency,* vol. 6, no. 3, pp. 42-51, July 1998.

[19] S. Khan and I. Ahmad, "A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids," *IEEE Trans. Parallel and Distributed Systems,* vol. 20, no. 3, pp. 346-360, Mar. 2009.

[20] A. Khokhar, V.K. Prasanna, M.E. Shaaban, and C. Wang, "Heterogeneous Computing: Challenges and Opportunities," *Computer,* vol. 26, no. 6, pp. 18-27, June 1993.

[21] L.A. Kramer and S.L. Smith, "Maximizing Flexibility: A Retraction Heuristic for Oversubscribed Scheduling Problems," *Proc. 18th Int'l Joint Conf. Artificial Intelligence,* Aug. 2003.

[22] Y.-K. Kwok and I. Ahmad, "Efficient Scheduling of Arbitrary Task Graphs to Multi-Processors Using a Parallel Genetic Algorithm," *J. Parallel and Distributed Computing,* vol. 47, no. 1, pp. 58-77, Nov. 1997.

[23] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic Task Mapping Algorithms for a Distributed Heterogeneous Computing Environment," *Proc. Fourth IEEE Heterogeneous Computing Workshop (HCW '95),* pp. 30-34, Apr. 1995.

[24] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, and R.F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *J. Parallel and Distributed Computing,* vol. 59, no. 2, pp. 107-121, Nov. 1999.

[25] A.M. Mehta, J. Smith, H.J. Siegel, A.A. Maciejewski, A. Jayaseelan, and B. Ye, "Dynamic Resource Allocation Heuristics that Manage Tradeoff between Makespan and Robustness," *J. Supercomputing,* Special Issue on Grid Technology, vol. 42, no. 1, pp. 33-58, Jan. 2007.

[26] NESDIS, Nat'l Environmental Satellite Data Information Service (Nesdis), http://www.nesdis.noaa.gov/About/about.html, Mar. 2006.

[27] M. Oltikar, J. Brateman, J. White, J. Martin, K. Knapp, A.A. Maciejewski, and H.J. Siegel, "Robust Resource Allocation in Weather Data Processing Systems," *Proc. Eighth Workshop High Performance Scientific and Eng. Computing,* pp. 445-454, 2006.

[28] V. Shestak, J. Smith, H.J. Siegel, and A. Maciejewski, "Stochastic Robustness Metric and Its Use for Static Resource Allocations," *J. Parallel and Distributed Computing,* vol. 68, no. 8, pp. 1157-1173, Aug. 2008.

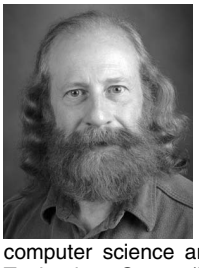
[29] S. Smith and C.-C. Cheng, "Slack-Based Heuristics for Constraint Satisfaction Scheduling," *Proc. 11th Nat'l Conf. Artificial Intelligence,* pp. 139-144, 1993.

[30] D. Whitley, "The GENITOR Algorithm and Selective Pressure: Why Rank Based Allocation of Reproductive Trials is Best," *Proc. Third Int'l Conf. Genetic Algorithms,* pp. 116-121, June 1989.

[31] M. Wu and W. Shu, "Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems," *Proc. Ninth Heterogeneous Computing Workshop (HCW '00),* pp. 375-385, Mar. 2000.

[32] D. Xu, K. Nahrstedt, and D. Wichadakul, "QoS and Contention-Aware Multi-Resource Reservation," *Cluster Computing,* vol. 4, no. 2, pp. 95-107, Apr. 2001.

Luis Diego Briceño received the BS degree in electrical and electronic engineering from the University of Costa Rica and the PhD degree in electrical and computer engineering from Colorado State University. He is currently a postdoctoral scholar at Colorado State University. His research interests include heterogeneous parallel and distributed computing. He is a member of the IEEE.



Howard Jay Siegel received the BS degrees in electrical engineering and management from Massachusetts Institute of Technology (MIT) and the MA, MSE, and PhD degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He was appointed the Abell Endowed Chair distinguished professor of electrical and computer engineering at Colorado State University (CSU) in 2001, where he is also a professor of computer science and director of the CSU Information Science and Technology Center (ISTeC). From 1976 to 2001, he was a professor at Purdue University. He has coauthored more than 380 technical papers. His research interests include robust computing systems, resource allocation in computing systems, heterogeneous parallel and distributed computing and communications, parallel algorithms, and parallel machine interconnection networks. He was a coeditor in chief of the *Journal of Parallel and Distributed Computing*, and has been on the Editorial Boards of both the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government. He is a fellow of the IEEE and the ACM. For more information, please see www.engr.colostate.edu/~hj.

Anthony A. Maciejewski received the BS, MS, and PhD degrees in electrical engineering from The Ohio State University in 1982, 1984, and 1987, respectively. From 1988 to 2001, he was a professor of electrical and computer engineering at Purdue University. In 2001, he joined Colorado State University, where he is currently the head of the Department of Electrical and Computer Engineering. He is a fellow of the IEEE. For more details, see www.engr.colostate.edu/~aam.

Mohana Oltikar received the bachelor's degree in engineering from the University of Mumbai, India, and the MS degree in electrical and computer engineering from Colorado State University. She is currently employed at Hughes Network Systems. Her research interests include heterogeneous parallel and distributed computing, baseband hardware design and verification, and FPGA design and verification.

Jeff Brateman received the BS degree in computer engineering from Colorado State University and the MS degree in electrical and computer engineering from Purdue University. Currently, he is a software engineer for PayPal in Austin, Texas.

Joe White received the BS degree in computer engineering from Colorado State University and the MS degree in computer science from the University of Colorado Denver. Currently, he is the director of Development for SurePayHealth at Recondo Technology in Denver, Colorado.

Jonathan R. Martin received the BS degree in computer engineering and computer science from Colorado State University. He is currently a vice president of web development firm R.L. Martin & Associates, Inc., specializing in assisting energy efficiency, and renewable energy organizations.

Keith Knapp received the BS degree in electrical and computer engineering from Colorado State University. He is currently working for GLI in Golden, Colorado.

Supplemental Material for Heuristics for Robust Resource Allocation of Satellite Weather Data Processing on a Heterogeneous Parallel System

Luis Diego Briceño[1], *Student Member, IEEE*, Howard Jay Siegel[1,2], *Fellow, IEEE*, Anthony A. Maciejewski[1], *Fellow, IEEE*, Mohana Oltikar[3], Jeff Brateman[4], Joe White[5], Jon Martin[6], and Keith Knapp[1].
Submitted to the IEEE Transactions on Parallel and Distributed Systems.

APPENDIX A: HEURISTICS FOR HIGH PRIORITY TASKS

1. Minimum Execution Time (MET)

The Minimum Execution Time (**MET**) [2], [6] heuristic considers tasks in an arbitrary order, and maps a task t_i to the machine j that has the smallest $ETC(i,j)$ for that task. The assignment obtained by the MET is independent of the task ordering [3]. Therefore, a random task ordering was used. The procedure for the MET heuristic is shown in Figure 1.

2. Minimum Completion Time (MCT)

The Minimum Completion Time (**MCT**) [2], [6] heuristic considers the tasks in a given random order. Each task is mapped to the machine that completes the task soonest, where the completion time (**CT**) of t_i on machine j is the ready time for machine j plus $ETC(i,j)$. The procedure for the MCT heuristic is shown in Figure 2.

3. K-Percent Best (KPB)

For the K-Percent Best (**KPB**) [7] heuristic, the "K-percent" of the machines with the smallest execution time for a given task are identified. The task is mapped to the machine in this subset that has the minimum completion time. A "K" value of $1/M\%$ causes this heuristic to be the same as with MET, while 100% implies that the heuristic is the same as MCT. Different values of K were explored, and it was found that the best average results across all ETCs were obtained when K was equal to 50%. A random ordering of tasks was used for the KPB heuristic. The procedure for the KPB heuristic is shown in Figure 3.

4. MinCT-MinCT

MinCT-MinCT [2], [6], [7] is a two-phase greedy heuristic based on the minimum completion time of the tasks. The procedure used for the MinCT-MinCT heuristic is shown in Figure 4.

5. MaxCT-MinCT

The MaxCT-MinCT [2], [6], [7] heuristic is similar to the MinCT-MinCT heuristic. However, instead of selecting the task-machine pair with the smallest completion time, this heuristic selects the task-machine pair that has the largest completion time. The intuition behind selecting the tasks with larger execution times first is to prevent the delayed mapping of a long task from disrupting a balanced loading near the end of the allocation [2]. The procedure used for the MaxCT-MinCT heuristic is shown in Figure 5.

This research was supported by the NSF under grants CNS-0615170 and CNS-0905399, and by the Colorado State University George T. Abell Endowment.
[1] Department of Electrical and Computer Engineering at Colorado State University. [2] Department of Computer Science at Colorado State University. [3] Hughes Network Systems, LLC. [4] IBM, Austin, TX. [5] Recondo Technology. [6] R. L. Martin & Associates.

(1) A list is generated that includes all unmapped tasks in a given arbitrary order.
(2) The first task in the list is assigned to its minimum execution time machine (ties are broken randomly).
(3) The task selected in step (2) is removed from the list.
(4) The ready time of the machine where the task is assigned is updated.
(5) Steps (2)–(4) are repeated until all the tasks have been mapped.

Fig. 1. Procedure for using MET to generate a resource allocation.

(1) A list is generated that includes all unmapped tasks in a given arbitrary order.
(2) The first task in the list is assigned to its minimum completion time machine (machine ready time plus estimated computation time of the task on that machine with ties broken randomly).
(3) The task selected in step (2) is removed from the list.
(4) The ready time of the machine where the task is assigned is updated.
(5) Steps (2)–(4) are repeated until all the tasks have been mapped.

Fig. 2. Procedure for using MCT to generate a resource allocation.

6. HPT Genitor

Genetic algorithms (**GA**) have been shown to work well for numerous problem domains, including resource allocation and job shop scheduling, e.g., [2], [4], [5]. The Genitor algorithm is a steady-state GA [8]. It uses a ranked population and only does one crossover operation per generation. Genitor (like other GAs) uses **chromosomes** to represent possible solutions, e.g., all tasks and the machines to which they are assigned. The Genitor heuristic implemented here is a variation of Genitor described in [8].

The chromosome is a vector of length T (number of tasks), and the i^{th} element of the vector is the machine to which t_i is assigned. The population is seeded with the best solution generated among the greedy heuristics for each specific simulation trial. The remaining 199 chromosomes are generated randomly. Multiple copies of a chromosome are not allowed in the initial population to reduce the probability of premature convergence.

The population is sorted in descending order of robustness radius (ρ_{HPT}). **Elitism**, the property that guarantees the best solution remains in the population, is implicitly implemented in Genitor by always maintaining a sorted list.

For crossover, two parents are selected using the linear bias approach [8], and two point crossover is used with the selected parents. The machine assignments from the part of the chromosome between the crossover points is exchanged between parents, and two new offspring are generated.

For mutation, each entry in an offspring has a probability of being randomly selected. For each mutated entry, a random machine assignment (from 1 to *M*) is chosen to replace the old machine assignment.

Each offspring is then evaluated and must compete for inclusion in the population. If the new offspring has a larger robustness radius than the worst member in the population, then the offspring is inserted in sorted order into the population, and the worst chromosome is removed. Otherwise, the new offspring is discarded. The heuristic is stopped after one hour (recall the Genitor is used just for comparison), and the best solution is selected.

The procedure for the HPTs Genitor heuristic is shown in Figure 6. Based on experimentation, the population size used for this study was 200 chromosomes, the probability of mutating a task-machine assignment was set to 5%, the probability of crossover was 100%, the linear bias parameter is set to 1.5, and the stopping criteria was set to one hour of execution time.

APPENDIX B: BOUND FOR HIGH PRIORITY TASKS

1. HPT Upper Bound (UB_{HPT})

To compute an upper bound ($\mathbf{UB}_{HPT}$) on the robustness of the HPTs, let $\boldsymbol{Set_{HP}}$ be the set of HPTs. The calculation for the bound assumes that each task can be executed using its minimum execution time, and that a single task can be split across multiple machines [2]. These assumptions are unrealistic and

(1) A list is generated that includes all unmapped tasks in a given arbitrary order.
(2) For the first task, a subset is formed by identifying the $\lfloor M \cdot (\frac{K}{100}) \rfloor$ machines with the smallest execution times for the task.
(3) The task is assigned to the machine that provides the minimum completion time in the subset (ties are broken randomly).
(4) The task is removed from the list.
(5) The ready time of the machine where the task is assigned is updated.
(6) Steps (2)–(5) are repeated until all tasks have been assigned.

Fig. 3. Procedure for using K-Percent Best to generate a resource allocation.

(1) A list is generated that includes all the unmapped tasks.
(2) For each task in the list, the machine that gives the task its minimum completion time (first "Min") is determined (ignoring other unmapped tasks).
(3) Among all task-machine pairs found in (2), the pair that has the minimum completion time (second "Min") is determined (ties are broken randomly).
(4) The task selected in (3) is removed from the list and is assigned to the paired machine.
(5) The ready time of the machine where the task is mapped is updated.
(6) Steps (2)–(5) are repeated until all tasks have been mapped.

Fig. 4. Procedure for using MinCT-MinCT to generate a resource allocation.

depending on the heterogeneity of an HCS the bound can be fairly loose. The equation used to calculate the bound is

$$UB_{HPT} = \tau_{expected} - \left[\sum_{t_i \in Set_{HP}} \min_{\forall j} ETC(i,j) \right] / M \ . \tag{1}$$

APPENDIX C: CONSISTENCY OF AN ETC MATRIX

ETC matrices were generated for this simulation using [1] to represent three different types of actual heterogeneity: consistent, inconsistent, and partially-consistent. For this study, 100 different ETCs were generated for each type of consistency. The results presented are averaged for each consistency over the 100 runs. For a consistent ETC matrix, if t_i has a lower execution time on machine x than machine y, then the same is true for any t_k. For an inconsistent ETC matrix, there is no such requirement. A combination of these two cases is the partially-consistent ETC matrix, which is an inconsistent matrix with a consistent sub-matrix [1]. For the partially-consistent matrices simulated here, the consistent sub-matrix was 50% of the tasks for 50% of the machines.

APPENDIX D: RESULTS FOR HIGH PRIORITY TASKS

1. Inconsistent Heterogeneity

The results of HPT for the inconsistent matrices are shown in Figure 7, and the execution time of the heuristics is shown in Table I. The MinCT-MinCT heuristic was the best performing greedy heuristic with an average robustness of 552.35. This result was over 99.9% of robustness value generated by the Genitor heuristic that was used for comparison.

2. Consistent Heterogeneity

The robustness results for the HPT consistent ETC matrices are shown in Figure 8, and the execution time of the heuristics is shown in Table I. The MinCT-MinCT heuristic was the best among the greedy heuristics; its performance was within 80.9% of UB_{HPT}, and within 99.9% of the best solution found by the Genitor heuristic. It is important to remember that the Genitor was implemented only for comparison, because of its long execution time. For the parameters used in this study, it is possible to obtain a good result in a short amount of time with a greedy heuristic (0.2 seconds versus 1 hour of runtime for Genitor).

(1) A list is generated that includes all unmapped tasks.
(2) For each task in the list, the machine that gives the task its minimum completion time is determined (ignoring other unmapped tasks).
(3) Among all task-machine pairs found in (2), the pair that has the maximum completion time is determined (ties are broken randomly).
(4) The task selected in (3) is removed from the list and is assigned to the paired machine.
(5) The ready time of the machine where the task is mapped is updated.
(6) Steps (2)–(5) are repeated until all tasks have been mapped.

Fig. 5. Procedure for using MaxCT-MinCT to generate a resource allocation.

(1) An initial population is generated.
(2) While the stopping criteria is not met (i.e., heuristic execution time is less than 1 hour):
  (a) Two chromosomes in the population are probabilistically selected as parents for crossover using the linear bias function.
    (i) Two random cut-off points are generated.
    (ii) The machine assignments of the tasks between the cut-off points are exchanged.
  (b) Each entry in an offspring chromosome has a chance of being mutated.
  (c) The resultant offspring of the crossover and mutation are inserted in the ranked population, and the two worst chromosomes are discarded.
(3) The best solution is output.

Fig. 6. Procedure for using HPT Genitor to generate a resource allocation.

The MET had the worst results; the average robustness generated by the MET heuristic had -1070.18 time units of robustness. The reason behind the poor performance of the MET heuristic is that all tasks are assigned to one machine. The KPB heuristic outperformed the MET heuristic, but did not perform as well as the MCT heuristic. Because the KPB heuristic shares characteristics with MET and MCT, it is not surprising to see that it shares the negative aspects of the MET heuristic when doing resource allocations with consistent ETC matrices.

3. Partially Consistent Heterogeneity

The results of HPT for the partially-consistent matrices are shown in Figure 9, and the execution time of the heuristics is shown in Table I. The best performing greedy heuristic was MinCT-MinCT, and it was over 99.9% on average of the solution Genitor was able to produce. Both the MinCT-MinCT and Genitor are very close to UB_{HPT} at about 85.8%. The MCT and KPB heuristic both had good performance and overlapping confidence intervals; both were approximately 79.7% of UB_{HPT}.

It is interesting to observe that, in general, the results are closer to UB_{HPT} in the partially consistent case than in the inconsistent case. This can be explained by considering the method used to calculate UB_{HPT}, i.e., it assumes all machines to be the MET machine. The upper bound is slightly tighter for the partially consistent case, because the ETCs we generated using the method in [1] had a better distribution of the minimum execution time machines than the inconsistent case.

APPENDIX E: RESULTS FOR REVENUE GENERATING TASKS

1. Results of Consistent ETCs

The results for the RGT consistent ETCs are shown in Figure 10, and the execution times are shown in Table II. The best greedy heuristic was the MaxWPTU-MaxWPTU with a robustness of 201.7 time units; this robustness was within 98% of robustness value generated by the Genitor heuristic.

The ARPTU variation of MCT outperformed all the other one-phase greedy heuristics. In terms of robustness, the robustness obtained by MCT-ARPTU was about 84.2% of the robustness obtained by the MaxWPTU-MaxWPTU heuristic. In terms of execution, the one-phase greedy mappings had an average of execution time between 0.02-0.03s; compared to the fastest two-phase heuristic (MaxWPTU-MaxWPTU) it was 178 times faster.

heuristic	consistent	partially consistent	inconsistent
MET, MCT, KPB	[0.02-0.03]	[0.02-0.03]	[0.02-0.03]
MinCT-MinCT	0.20	0.19	0.20
MaxCT-MinCT	0.26	0.26	0.26

TABLE I
EXECUTION TIMES (IN SECONDS) FOR HPT.

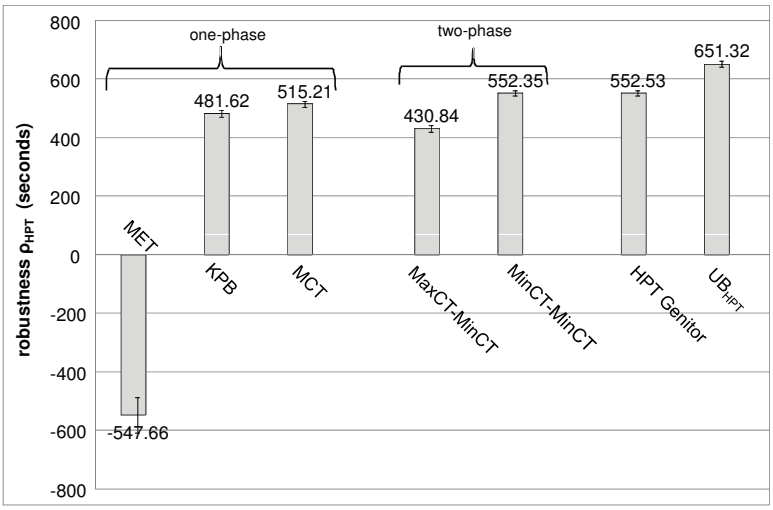


Fig. 7. Inconsistent heterogeneity robustness for HPT, averaged over 100 trails. The 95% confidence intervals are shown.

The best variation for the KPB was the ARPTU; both the ARPTU and MRPTU produced mappings that were robust. However, the RAND variation did not produce robust results. All variations of the MET heuristic have a negative robustness. The variation MET-MRPTU was the best performing among the tested orderings for MET. This is an obvious result as the minimum execution time machine is the only machine that matters when considering the assignment of the MET heuristic.

The MinCT-MinCT heuristic performed better than the MET and KPB heuristics that used revenue to determine the task assignment order. However, the MinCT-MinCT did not perform better than two of the MCT variations. Because the MCT heuristic distributes the load across all machines, the added revenue consideration makes it perform better than the MinCT-MinCT heuristic.

It is better to use a greedy two-phase heuristic instead of the KPB and MET because the two-phase heuristic has a broader view of the unmapped tasks when mapping a task to a machine. The RAND variation of all one-phase greedy heuristics (MET, KPB, and MCT) performed the worst, because it does not consider the revenue when ordering tasks. However, the results from the RAND illustrate how much improvement is obtained by using the revenue-based orderings.

2. *Results of Partially-Consistent ETCs*

The results of RGT for the partially-consistent matrices are shown in Figure 11, and the execution time of the heuristics is shown in Table II. The best performing greedy heuristic for RGT was the MaxWPTU-MaxWPTU with a robustness of 246.00 time units, which was 73.5% of UB_{RGT}. The RGT Genitor gave an average improvement of 1.6% time units, and had overlapping confidence intervals with MaxWPTU-MaxWPTU. Among the one-phase heuristics MCT-ARPTU had the best performance with a robustness of 208.8, which was about 84.9% of the performance obtained by the best performing greedy heuristic (MaxWPTU-MaxWPTU).

REFERENCES

[1] 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, pp. 195–207, Nov. 2000.

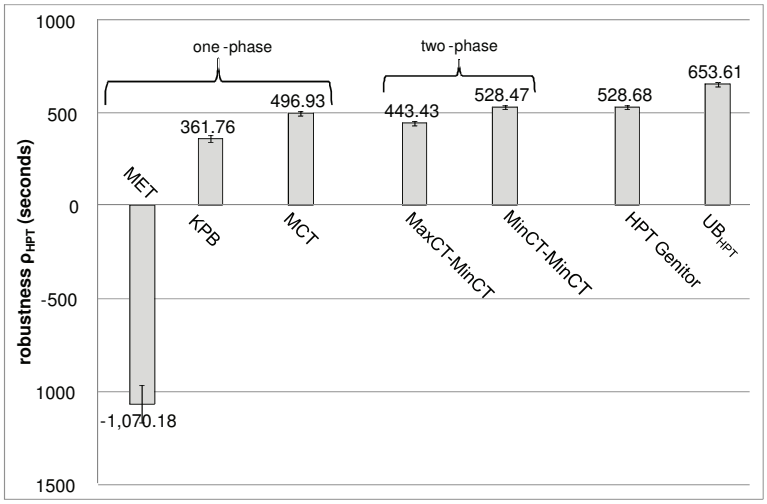


Fig. 8. Consistent heterogeneity robustness for HPT, averaged over 100 trails. The 95% confidence intervals are shown.

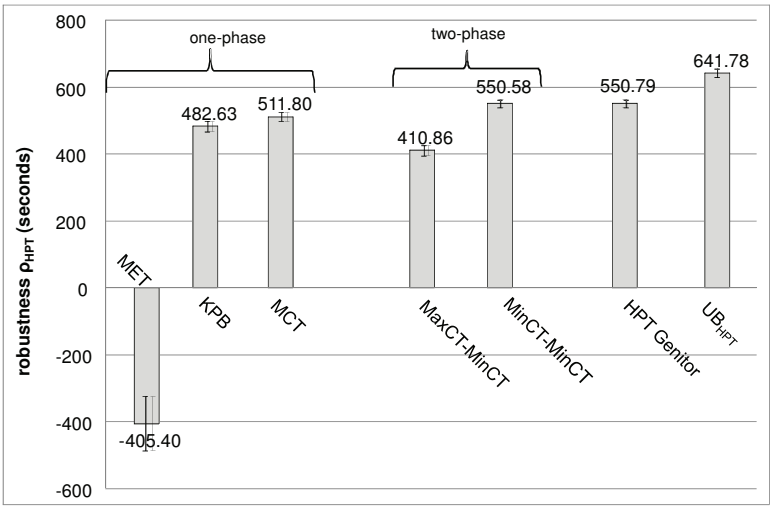


Fig. 9. Partially-consistent heterogeneity robustness for HPT, averaged over 100 trails. The 95% confidence intervals are shown.

[2] 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 B. 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, pp. 810–837, Jun. 2001.

[3] L. D. Briceno, M. Oltikar, H. J. Siegel, and A. A. Maciejewski, "Study of an iterative technique to minimize completion times on non-makespan machines," in *Heterogeneity in Computing Workshop (HCW '07)*, Mar. 2007, p. 138.

[4] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms–i. representation," *Computers & Industrial Engineering*, vol. 30, no. 4, pp. 983 – 997, 1996.

[5] F. D. Croce, R. Tadei, and G. Volta, "A genetic algorithm for the job shop problem," *Computers & Operations Research*, vol. 22, no. 1, pp. 15–24, 1995.

[6] 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, pp. 280–289, Apr. 1977.

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

[8] D. Whitley, "The GENITOR algorithm and selective pressure: Why rank based allocation of reproductive trials is best," in *3rd Int'l Conf. on Genetic Algorithms*, Jun. 1989, pp. 116–121.

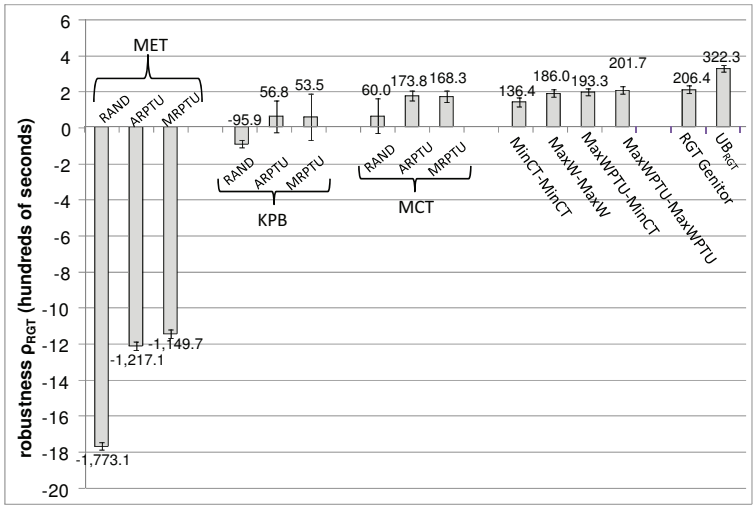


Fig. 10. Consistent heterogeneity robustness for RGT, averaged over 100 trails. The 95% confidence intervals are shown.

heuristic	consistent	partially consistent	inconsistent
MET, MCT, KPB	[0.02-0.03]	[0.02-0.03]	[0.02-0.03]
MinCT-MinCT	1.46	1.42	1.48
MaxW-MaxW	2.10	2.11	2.00
MaxWPTU-MinCT	2.11	2.01	2.11
MaxWPTU-MaxWPTU	3.55	3.54	3.56

TABLE II
EXECUTION TIMES (IN SECONDS) FOR RGT.

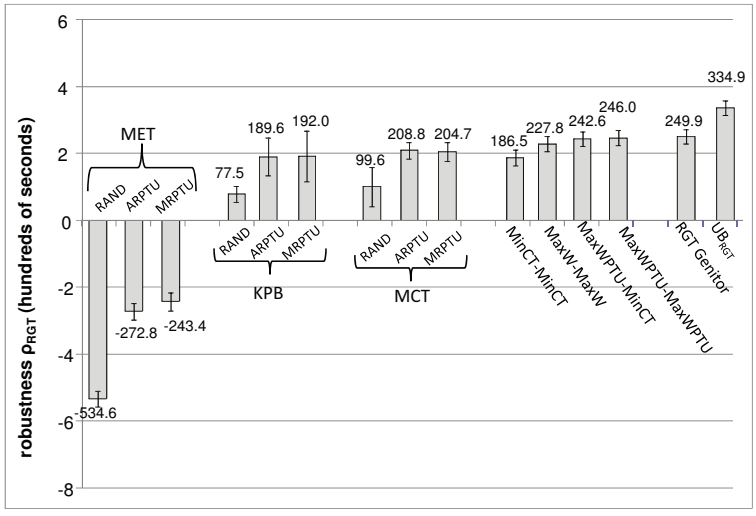


Fig. 11. Partially-consistent heterogeneity robustness for RGT, averaged over 100 trails. The 95% confidence intervals are shown.