Dynamic Mapping in a Heterogeneous Environment with Tasks Having Priorities and Multiple Deadlines

Jong-Kook Kim[1], Sameer Shivle[2], Howard Jay Siegel[2,3], Anthony A. Maciejewski[2], Tracy D. Braun[4], Myron Schneider[2,5], Sonja Tideman[3], Ramakrishna Chitta[3], Raheleh B. Dilmaghani[2], Rohit Joshi[2], Aditya Kaul[2], Ashish Sharma[2], Siddhartha Sripada[2], Praveen Vangari[2], and Siva Sankar Yellampalli[6]

[1]Purdue University
Electrical and Computer Engineering School
West Lafayette, IN 47907-1285, USA
jongkook@purdue.edu

Colorado State University
[2]Electrical and Computer Engineering Dept.
[3]Computer Science Dept.
Fort Collins, CO 80523-1373, USA
{hj, aam}@colostate.edu
{ssameer, raheleh, rohit, ashish, siddhu, praveen}@engr.colostate.edu
{ramacmr, tideman}@cs.colostate.edu
aditya21@lycos.com

[4]University of Maryland
University College, Asia Division
Kadena Education Center
18 MSS/DPE, Unit 5134, Box 40
APO AP 96368-5134
tdbraun@ad.umuc.edu

[5]Agilent Technologies
Loveland, CO 80537, USA
myron_schneider@agilent.com

[6]Lousiana State University
Electrical and Computer Engineering School
Baton Rouge, LA 70802, USA
syella1@lsu.edu

Abstract

In a distributed heterogeneous computing system, the resources have different capabilities and tasks have different requirements. To maximize the performance of the system, it is essential to assign resources to tasks (match) and order the execution of tasks on each resource (schedule) in a manner that exploits the heterogeneity of the resources and tasks. The mapping (defined as matching and scheduling) of tasks onto machines with varied computational capabilities has been shown, in general, to be an NP-complete problem. Therefore, heuristic techniques to find a near-optimal solution to this mapping problem are required. Dynamic mapping is performed when the arrival of tasks is not known a priori. In the heterogeneous environment considered in this study, tasks arrive randomly, tasks are independent (i.e., no communication among tasks), and tasks have priorities and multiple deadlines. This research proposes, evaluates, and compares eight dynamic heuristics. The performance of the best heuristics is 83% of an upper bound.

This research was supported in part by the Colorado State University George T. Abell Endowment.

1. Introduction and Problem Statement

Heterogeneous computing (HC) is the coordinated use of various resources with different capabilities to satisfy the requirements of varying task mixtures. The heterogeneity of the resources and tasks in an HC system is exploited to maximize the performance or the cost-effectiveness of the system (e.g., [5, 9, 13, 18]). A typical HC system consists of heterogeneous sets of resources and tasks. To exploit the different capabilities of a suite of heterogeneous resources, typically a resource management system (RMS) allocates the resources to the tasks and the tasks are ordered for execution on the resources. In this research, heuristics are proposed that can be used in such an RMS.

An important research problem is how to assign resources to tasks (match) and order the execution of tasks on the resources (schedule) to maximize some performance criterion of an HC system. This procedure of matching and scheduling is called mapping. There are two different types of mapping: static and dynamic. Static mapping is performed when the applications are mapped in an off-line planning phase [6], e.g., planning the schedule for a set of production jobs. Dynamic mapping is performed when the applications are mapped in an on-line fashion [17], e.g., when tasks arrive at unknown intervals and are mapped as they arrive (the workload is not known *a priori*). In both cases, the mapping problem has been shown, in general, to be NP-complete (e.g., [8, 10, 15]). Thus, the development of heuristic techniques to find near-optimal solutions for the mapping problem is an active area of research (e.g., [1, 3, 4, 5, 6, 9, 11, 17, 19, 25]).

In this research, the dynamic mapping of tasks onto machines is studied. Simulation is used for the evaluation and comparison of the dynamic heuristics developed in this research. As described in [17], dynamic mapping heuristics can be grouped into two categories, immediate mode and batch mode. Each time a mapping is performed (mapping event), immediate mode heuristics only consider the new task for mapping, whereas batch mode considers a subset of tasks for mapping, thus having more information about the task mixture before mapping the tasks. As expected, the study in [17] showed that the immediate mode heuristics had shorter running times than those of the batch mode heuristics, but the batch mode heuristics gave higher performance. The heuristics proposed in this research are batch mode schemes.

At any mapping event, the very next task in each machine's job queue waiting for the currently executing task to finish is not considered in any of the heuristics. The reason is that while a mapping event occurs the current task can finish; therefore, to help ensure that the machine will not be idle for the duration of the mapping event, the very next task is not considered for mapping. It is not desirable for a machine to be idle for the duration of the mapping event. While it is still possible that a machine may become idle, it is highly unlikely for the assumptions in this research (the average execution time of a task is 180 seconds while the average execution time of a mapping event is less than 0.5 seconds).

A dynamic mapping approach is designed to compute the new mapping faster than the anticipated average arrival rate of the tasks to avoid being interrupted by an arriving task. Therefore, the heuristics that are developed have a limit on the maximum time each computation of a new mapping can take. When a task arrives while a mapping event is in progress, the current mapping event is not disturbed. As soon as the current mapping event is completed, the next mapping event starts that include any tasks that had arrived during the previous mapping event.

The HC environment considered is oversubscribed, such that not all tasks can complete during the evaluation period. To model such an environment, the arrival rates of tasks are determined so that in the evaluation period, there are enough tasks to simulate an oversubscribed system. An environment with bursty task arrivals during the evaluation period was simulated.

Eight dynamic mapping schemes are studied in this paper: Max-Max, Max-Min, Min-Min, Queueing Table, Relative Cost, Slack Sufferage, Switching Algorithm, and Percent Best. The Max-Max and Max-Min approaches are considered greedy heuristics, the Queueing Table uses a lookup table to map tasks, Relative Cost and Slack Sufferage are based on the sufferage concept used in [17], and Switching Algorithm and Percent Best are an extension of the Switching Algorithm and *k* percent best methods, respectively, given in [17].

In the simulation experiments, the estimated time to complete (ETC) values are used by the mapping heuristics, where the ETC(*i*, *j*) is the estimated execution time of task *i* on machine *j*, where *i* is the task number and *j* is the machine number. These estimated values may differ from actual times, e.g., actual times may depend on input data. Therefore, for the simulation studies, the actual time to complete (ATC) values are calculated using the ETC values as the mean. The ATC values are used for the evaluation of the heuristics. The details of the simulation environment are presented in Section 4.

The tasks considered here are assumed to be independent, i.e., no communication or dependency between tasks. Each task has a priority level, i.e., high, medium, and low. In addition, each task has multiple deadlines, i.e., a 100% deadline, a 50% deadline, and a 25% deadline. The worth of a submitted task will degrade according to a degradation scheme if the task misses a certain deadline.

The performance metric for evaluating the mappings generated by the heuristics has three components. The first component of the evaluation function is the weighted priority of each task, p_j where,

$$p_j = \begin{cases} x^2 & \text{for high priority tasks} \\ x & \text{for medium priority tasks} \\ 1 & \text{for low priority tasks} \end{cases} .$$

For this research, $x = 2$ or 4. The weighted priority of a task is the maximum worth it can contribute to the performance function that specifies the value of a mapping.

The second component of the evaluation function incorporates the deadlines. Let w_i be a deadline factor for task i, where

$$w_i = \begin{cases} 1.00 & \text{if } t_i \text{ finished at or before its primary deadline} \\ 0.50 & \text{if } t_i \text{ finished at or before its 50\% deadline} \\ 0.25 & \text{if } t_i \text{ finished at or before its 25\% deadline} \\ 0.05 & \text{if } t_i \text{ finished after its 25\% deadline} \\ 0 & \text{if } t_i \text{ is never executed} \end{cases}$$

The w_i indicates the degradation scheme of the worth of a task according to when the task finishes.

For the third component, let B denote the beginning of the evaluation interval and let E denote the end of the evaluation interval. The simulated actual execution time for task i on machine j is ATC(i, j). The start time of task i on machine j is $st(i, j)$ and the finish time of task i on machine j is $ft(i, j)$. Then

$$b_i = \begin{cases} \left(ct(i,j) - B\right) / \mathrm{ATC}(i,j) & \text{if } st(i,j) < B \text{ and } B < ft(i,j) \le E \\ 1.00 & \text{if } st(i,j) \ge B \text{ and } ft(i,j) \le E \\ \left(E - st(i,j)\right) / \mathrm{ATC}(i,j) & \text{if } B \le st(i,j) < E \text{ and } ft(i,j) > E \\ \left(E - B\right) / \mathrm{ATC}(i,j) & \text{if } st(i,j) \le B \text{ and } ft(i,j) \ge E \\ 0 & \text{if } ft(i,j) \le B \text{ or } st(i,j) \ge E \end{cases}$$

gives the boundary weighting for each task i, where j is determined by the mapping done by the heuristic for each task.

Let T be the total number of tasks that are mapped (i.e., the total number of tasks in the ETC matrix). Then the value function, V, used to evaluate each mapping is defined as

$$V = \sum_{i=0}^{T-1} p_i \times w_i \times b_i.$$

The next section provides discussions of the literature related to this work. In Section 3, the heuristics studied in this research are presented. Section 4 describes the simulation environment and results, and the last section gives a brief summary of this research.

2. Related Work

In the literature, the mapping of tasks onto machines is also often referred to as scheduling. Researchers have worked on the dynamic mapping problem for distributed computer systems.

Many of the heuristics in the literature (e.g., [15, 16]) use the minimum completion time of a task to decide where the task should be mapped. The heuristics presented in [15] are concerned with mapping independent tasks onto heterogeneous machines such that the completion time of the last finishing task is minimized. The difference is that our research has a task model with priorities and multiple deadlines and that an overloaded environment is simulated. Our research considers a different performance metric that is the collective value of tasks completed during an interval of time. The minimum completion time is used in the decision process of some of the heuristics presented in our research.

The Min-Min heuristic in [15], which is also one of the heuristics implemented in SmartNet [12], has proven to be a good heuristic for dynamic and static mapping problems in earlier studies (e.g., [1, 6, 17]). The Max-Min approach in [15], which is a variation of the Min-Min heuristic, also performed well in certain HC environments. These two heuristics are expanded for the environment in this research.

The two heuristics, *k*-percent best and the Sufferage heuristic, that were the idea behind some of the approaches presented in this research, performed comparably to the Min-Min and the Max-Min schemes in the studies in [17]. In our preliminary experimentation using the variations of the *k*-percent best and the Sufferage heuristics, these heuristics performed as well as or better than the Min-Min and Max-Min type heuristics. Therefore, these two were chosen for this research. The environment in [17] is similar in that the tasks are independent and randomly arriving. The difference is that our study considers tasks with priorities and multiple deadlines, and the value of the tasks completed is used as the performance metric.

The environment in [20] is similar to the environment considered in our research in that [20] has randomly arriving tasks with a hard deadline. The idea in [20] to move a task if its deadline may not be satisfied is used in one of the heuristics in our research. However, the environment in our research includes task with priorities and multiple deadlines, and heterogeneous machines, all of which complicate the scheduling problem. Furthermore, our performance metric is different.

The DeSiDeRaTa project (e.g., [7, 14, 21, 22, 23, 24]) focuses on dynamically reallocating resources for applications, but the system model is very different. The system model in DeSiDeRaTa includes sets of heterogeneous machines, sensors, applications, and actuators. The application model in the DeSiDeRaTa project is different from the task model here in that the applications in the DeSiDeRaTa project are continuously running ones where data inputs to an application are processed and output to another application or an actuator. In contrast, the tasks in this research are independent, are randomly arriving, have priorities, and have multiple deadlines.

The work in [26] focuses on the dynamic mapping of independent tasks onto machines in an environment that is similar to the one in this study (i.e., randomly arriving tasks, heterogeneous machines, and heterogeneous tasks). Some of the algorithms in [26], such as Min-Min and Max-Min, are also used in our research. The idea of "fine-tuning" in [26] is used in our research as "rescheduling" after all tasks are mapped. The difference is that the tasks in our study are assigned a weighted priority, each task has multiple deadlines, and the performance metric is the value of tasks completed in an interval of time instead of completion rate, defined as the number of tasks completed in an interval of time.

3. Heuristics

3.1. Max-Max

The Max-Max heuristic is based on the Min-Min (greedy) concept in [15]. The fitness value for the task on a given machine is the worth of the task divided by the estimated execution time of the task, where the worth of the task is the priority weighting of the task multiplied by the deadline factor of the task. The fitness value in this heuristic calculates the worth per unit time and it is used in the selection of a task to be mapped.

The Max-Max heuristic can be summarized by the following six-step procedure. The procedure starts when a new task arrives and generates a mapping event. The mappable tasks are tasks that are waiting to be executed in the machine queue (except the very next task) and the new task. When the mapping event begins, it is assumed that none of the mappable tasks are mapped, i.e., they are not in any machine queue.

(1) A task list is generated that includes all the mappable tasks.
(2) For each task in the task list, find the machine that gives the task its maximum fitness value (the first "Max"), ignoring other tasks in the mappable task list.
(3) Among all the task/machine pairs found from above, find the pair that gives the maximum fitness value (the second "Max").
(4) Remove the above task from the mappable task list and map the task to its paired machine.
(5) Update the machine availability status.
(6) Repeat steps (2) to (5) until all tasks are mapped.

The availability status of all machines is updated in step (5) to be used in calculating the deadline factor. The deadline factor for a given task/machine pair is determined using the machine availability time of the machine plus the estimated execution time of the task. The worth for all tasks on all machines is recalculated every time a task is mapped.

3.2. Max-Min and Min-Min

The Max-Min heuristic is also based on the greedy concept in [15]. The completion time for task *i* on machine *j* is the time machine *j* is available to execute task *i*, the machine availability time (mat(*j*)), plus ETC(*i*, *j*). This heuristic finds the machine with the minimum completion time machine for each task. Then, from these task/machine pairs the heuristic selects the pair that has the maximum completion time. This method maps tasks that take more time first because these tasks typically have a higher probability of not completing before their deadline if not mapped as soon as possible.

The Max-Min heuristic can be summarized by the following eight-step procedure. The procedure starts when a new task arrives and generates a mapping event. When the mapping event begins, it is assumed that none of the mappable tasks are mapped, i.e., they are not in any machine queue.

(1) A task list is generated that includes all the mappable tasks.
(2) For each task in the mappable task list, find the minimum completion time machine (the "Min"), ignoring other tasks in the mappable task list.
(3) Among all the task/machine pairs found from above, select the pair that gives the maximum completion time (the "Max").
(4) The task identified above is removed from the mappable task list and assigned to its paired machine.
(5) Update the machine availability status.
(6) Repeat steps (2) to (5) until all the tasks are mapped.
(7) For each machine, if there are tasks in the machine queue, reschedule the tasks in the machine queue according to their worth.
(8) Stop when all machine queues are rescheduled.

The availability status of all machines is updated in step (5) to calculate the minimum completion time over all machines for each task in step (2).

The rescheduling of tasks in step (7) can be summarized by the following procedure.

(a) Initialize the machine availability time to the completion time of the very next task that is waiting to be executed (i.e., assume that none of the mappable tasks are mapped).
(b) Group the tasks using the priority levels of the tasks.
(c) For the tasks in the high priority level group, keeping their relative ordering from the machine queue, one by one, in order, insert the tasks that can

finish by their primary deadline into the machine queue. Once scheduled, a task is removed from the group and the machine availability status is updated.

(d) Repeat step (c) for 50% deadline and 25% deadline.

(e) Repeat steps (c) to (d) for the medium priority tasks and then repeat for the low priority tasks.

(f) High priority tasks that cannot finish by the 25% deadline are added to end of the machine queue. Medium priority tasks that cannot finish by the 25% deadline are added next and then the low priority tasks are added to the machine queue.

The Min-Min heuristic, which is a variation of the Max-Min heuristic, was implemented. The difference is in step (3), where instead of selecting the pair that gives the maximum completion time, the pair that gives the minimum completion time is selected. This means that tasks with shorter execution times will be selected. This variation is attempted to greedily complete as many tasks as possible.

3.3. Percent Best

The Percent Best heuristic is a variation of the *k* percent best heuristic found in [17]. This heuristic tries to map the tasks onto the minimum execution time machine while considering the completion times on the machines. The idea behind the heuristic is to pick the top *m* machines with the best execution time for a task, so that the task can be mapped onto one of its best execution time machines. However, limiting the number of machines to which a task can be mapped, may cause the system to become unbalanced, therefore the completion times are also considered in selecting the machine to map the task.

The Percent Best heuristic can be summarized by the nine-step procedure below. The procedure starts when a new task arrives and generates a mapping event. When the mapping event begins, it is assumed that none of the mappable tasks are mapped, i.e., they are not in any machine queue.

(1) A task list is generated that includes all the mappable tasks.

(2) Tasks are grouped according to their priority levels.

(3) For each task in the high priority level group, find the top $m = 3$ machines that give the best *execution* time for that task (the total number of machines used in the simulation studies in this research is eight).

(4) For each task, find the minimum *completion* time machine from the machines found in step (3) and the machines that are idle.

(5) Map the tasks with no contention (i.e., when there are no other tasks with the same minimum completion time machine) and remove them from the priority level group.

(6) For tasks with contention (tasks having the same minimum completion time machine), map the task with the earliest primary deadline and remove it from the priority level group.

(7) Update the machine availability status.

(8) Repeat steps (3) to (7) until all tasks in the group are mapped.

(9) Repeat steps (3) to (8) for tasks in the medium and low priority level groups, using $m = 4$ and $m = 8$, respectively.

3.4. Queueing Table

The Queueing Table heuristic uses a lookup table constructed using the priority, the relative speed of execution, and the nearness of deadline (see Table 1) in the mapping process. The relative speed of execution (RSE) is the ratio of the average execution time of a task across all machines to the overall average task execution time for all tasks across all machines in the HC system. The Queueing Table heuristic uses the above definition and a heuristic constant (RSE cutoff) to classify tasks into one of two categories: "slow" and "fast." If a task's RSE > RSE cutoff, then it is considered to be slow and if a task's RSE ≤ RSE cutoff, then it is considered to be fast.

queuing order	priority level	relative speed of execution	nearness of deadline
1	high	slow	sooner
2	high	fast	sooner
3	high	slow	later
4	high	fast	later
5	med	fast	sooner
6	low	fast	sooner
7	med	fast	later
8	low	fast	later
9	med	slow	sooner
10	med	slow	later
11	low	slow	sooner
12	low	slow	later

Table 1: The lookup table constructed using the priority, relative speed of execution, and the nearness of deadline for the Queueing Table heuristic.

Let δ be the primary deadline of a given task *i* minus the current time. Then the nearness of deadline (NOD) of a given task *i* is

$$(\delta - 2 \times (\text{average ETC}(i, j) \text{ over all } j))/\delta.$$

This ratio measures the urgency of a given task *i*; the smaller the NOD the more urgent. When the current time passes the primary deadline of a task, the task's NOD is set to infinity. The arbitrary multiplier of "2" in the above definition is to provide some padding to help take into account that the task may be mapped onto a machine where the simulated actual execution time will be worse than the average. The heuristic uses the above definition of NOD and a heuristic constant (NOD cutoff) to classify task urgency into two categories. If a task's NOD ≤ NOD cutoff this indicates that the task needs to be started "sooner" and if a task's NOD > NOD cutoff, then the task can be started "later."

The Queueing Table heuristic can be summarized by the following ten-step procedure. The procedure starts when a new task arrives and generates a mapping event. In contrast to other heuristics, this method does not generate a task list that includes all the mappable tasks and initially maps only the new task to a machine.

(1) For all mappable tasks, the NOD is calculated.
(2) For the new task, calculate the RSE.
(3) For each of the machines, compare the new task with the tasks on that machine's queue, starting from the front of the queue. If there are no tasks with the same queueing order as the new task, then the new task's position is in front of the first task with the higher numbered queueing order. If there are tasks with the same queueing order as the new task, then the new task's position is in front of the first task that has a higher NOD value than that of the new task.
(4) Using the position on each machine queue found from above, the completion time on all machines is calculated and the new task is mapped to its minimum completion time machine.
(5) For each machine, check if there are any tasks that will miss their primary deadline.
(6) Among the tasks that miss their primary deadline, find the first task that misses its deadline.
(7) For the task found in step (6), find machines where (a) the priority of the task is equal to or greater than the highest priority of any task on that machine and (b) moving the task to the front of that machine queue does not cause any task to miss its primary deadline (tasks already missing their primary deadline are not checked).
(8) Among the machines identified above, find the machine that gives the minimum completion time for the task and move the task to the head of the machine queue. (If no machines are found in step (7), the task is not moved.)
(9) Update the machine availability status.
(10) Repeat steps (5) to (9) until all machines are checked (the order in which the machines are checked is from machine 1 to machine M, where M is the total number of machines).

As indicated in step (5), the search of tasks missing their primary deadline is done on all machines. This is because at the next mapping event, there may be tasks in machine queues other than the one the new task is mapped to that miss their primary deadline. At any mapping event, even if there are multiple tasks in a machine queue that need to be moved, only one task from the machine is allowed to be moved (the maximum number of tasks that can be moved at any mapping event is equal to the total number of machines).

3.5. Relative Cost

The Relative Cost heuristic uses the worth and the sufferage idea in [17] to map tasks. For each mappable task considered, the relative cost (RC) is calculated by computing the minimum completion time of that task over all machines divided by the average completion time of that task on all machines. When the RC is high, the minimum completion time is similar to the average and most of the completion times on all machines are similar. When the RC is low, the minimum completion time is very different from the average. Assume tasks *a* and *b* prefer the same machine (best machine) for mapping. Task *a* is considered to suffer more than Task *b*, when there is a larger difference between the completion times of the best and the second best machines. The RC is an approximation of this difference. If the RC for a task is high then there is a low probability that the task will suffer more than a task that has a low RC.

The Relative Cost heuristic can be summarized by the following seven-step procedure. The procedure starts when a new task arrives and generates a mapping event. When the mapping event begins, it is assumed that none of the mappable tasks are mapped, i.e., they are not in any machine queue.

(1) A task list is generated that includes all the mappable tasks.
(2) For each task in the mappable task list, calculate the RC.
(3) For all the tasks in the mappable task list, their worths are calculated (as described below) and the tasks are sorted according to their worths (highest first).
(4) The task having the highest worth is chosen. Ties among the highest worth tasks are broken by selecting the lowest RC value task.
(5) Map the chosen task on its minimum completion time machine and remove it from the mappable task list.
(6) Update the machine availability status.

(7) Repeat steps (2) to (6) until all tasks are mapped.

In step (3), the deadline factor for each task is calculated using the minimum completion time of that task over all machines, the current time, and the deadline for the tasks, ignoring other tasks in the mappable task list. Using the deadline factor found for each task, the worth is recalculated every time a task is mapped.

The availability status of all machines is updated in step (6) to calculate the completion time of all tasks on all machines and the deadline factor. Among the same worth tasks, the task with the smallest relative cost is considered to suffer most if it is not mapped first.

3.6. Slack Sufferage

The Slack Sufferage heuristic uses the sufferage concept in [17]. The Slack Sufferage heuristic can be summarized by the following nine-step procedure. The procedure starts when a new task arrives and generates a mapping event. When the mapping event begins, it is assumed that none of the mappable tasks are mapped, i.e., they are not in any machine queue. In this heuristic, the percentage slack for task i on machine j using a given deadline d is defined as

$$\underline{PS(i, j, d)} = 1 - (ETC(i, j) / (d - mat(j))).$$

(1) A task list is generated that includes all the mappable tasks.
(2) For each task in the mappable task list, for each machine calculate the PS(i, j, d), where d is task i's primary deadline. PS(i, j, d) = -1 for a machine if the task misses its deadline on that machine.
```
For a given task $i$, if PS($i$, $j$, $d$) < 0 for all machines
    recalculate the PS($i$, $j$, $d$) for each machine using $d$ = 50% deadline.
    if PS($i$, $j$, $d$) < 0 for all machines
        recalculate the PS($i$, $j$, $d$) for each machine using $d$ = 25% deadline
        if PS($i$, $j$, $d$) < 0 for all machines
            recalculate the PS($i$, $j$, $d$) for each machine using $d$ = end of the evaluation period
```
(3) For each task, determine the maximum percentage slack machine.
(4) Sort tasks by their worth (worth is calculated using the deadline factor associated with d).
(5) If there is more than one task with the current highest worth, check for contention (i.e., whether tasks have the same maximum percentage slack machine).
(6) If there is no contention, select the highest worth task.
If there is contention among the highest worth tasks, select the most critical task (the task with the largest difference of percentage slack between the best percentage slack and the second best percentage slack machines).
(7) Map the selected task and remove it from the task list.
(8) Update the machine availability status.
(9) Repeat steps (2) to (8) until all tasks are mapped.

The availability status of all machines is updated in step (8) to determine the deadline factor and the PS(i, j, d).

3.7. Switching Algorithm

The Switching Algorithm heuristic is an extended version of the switching algorithm in [17]. The Switching Algorithm heuristic can be summarized by the following three-step procedure. The procedure starts when a new task arrives and generates a mapping event. The load balance ratio for the system in the heuristic is the ratio of the earliest machine availability time over all the machines in the suite to the latest machine availability time. A high threshold and a low threshold are determined arbitrarily for this ratio (high threshold > low threshold). Initially, new tasks are mapped onto their minimum completion time machine and they are always inserted at the end of the chosen machine queue.

(1) Calculate the load balance ratio for the system.
(2) If the load balance ratio > high threshold, switch method to use the minimum *execution* time machine to map the new task.
If the load balance ratio < low threshold, switch method to use the minimum *completion* time machine to map the new task.
If low threshold ≤ load balance ratio ≤ high threshold, use method from previous mapping event to map the new task.
(3) All the tasks in the machine queue where the new task is mapped are reordered using their priority. If tasks have the same priority then order the tasks with earlier primary deadline first.

If the load balance ratio becomes higher than a high threshold, this means that the system has become balanced in load. Then the algorithm switches to use the minimum execution time machine for mapping tasks. If the load balance ratio becomes lower than a low threshold, this means that the system has become unbalanced in load. Then the heuristic switches back to use the minimum completion time machine.

After the new task is assigned to a machine and inserted at the very end of that machine queue, the tasks on that machine are reordered using step (3). The reordering allows the new task to move closer to the front of the queue if it has a higher priority weighting than some of the other tasks in the queue. After the tasks are reordered using the priority levels, the tasks within the same priority level are ordered again using their primary deadline. The more urgent tasks are moved closer to the front of the queue, e.g., a task with the earliest primary deadline is ordered first.

3.8. Tight upper bound (TUB)

The procedure for the tight upper bound (TUB) starts by sorting all tasks in descending order based on (priority of task *i*)/(minimum ATC(*i*, *j*) over all *j*). (Recall that an oversubscribed system is assumed.) Ties are broken arbitrarily. Using the ordering, each task is considered. The full priority weighting of each task is summed until the added minimum ATC values of the tasks exceed the total evaluation time of all machines (i.e., (evaluation period) × (number of machines)).

4. Simulation Setup and Results

An HC system with eight machines and an average of 1250 tasks was simulated for a period of 250 minutes. A trial is defined as one such simulation of the HC system. For each of the scenarios that will be discussed later in this section, 50 trials are run. The period from 0 to 10 minutes is the system start-up period. The period between 10 to 250 minutes is considered the evaluation period (i.e., the period where the heuristics' performance is measured). Within the simulation period (i.e., the system start up period and the evaluation period), the arrival times of the tasks are randomly generated using a Poisson distribution. To better simulate an overloaded system, the mean task inter-arrival time is faster (3.5 seconds) during the system start-up period than during the evaluation period (14 seconds). In addition, random bursty arrival rate periods are introduced during the evaluation period, where the arrival rate is increased. These periods do not overlap with each other and have a mean task inter-arrival time of 7 seconds. The duration of a bursty period is 10 minutes.

The estimated execution times of all tasks taking heterogeneity into consideration are generated using the gamma distribution method described in [2]. Two different cases of ETC heterogeneities are used in this research, the high task and high machine heterogeneity (high heterogeneity) case and the low task and low machine heterogeneity (low heterogeneity) case. For both heterogeneity cases, a task mean and coefficient of variation (COV) are used. (The COV is defined as the standard deviation divided by the mean.) The high heterogeneity cases use a mean task execution time of three minutes and a COV of 0.9 (task heterogeneity) to calculate the values for all of the elements in a task vector (where the number of elements equal the total number of tasks). Then using the *i*-th element of the vector as the mean and a COV of 0.9 (machine heterogeneity), the ETC values for task *i* on all the machines are calculated. The low heterogeneity cases use a mean task execution time of three minutes and a COV of 0.3 for task heterogeneity and 0.3 for machine heterogeneity.

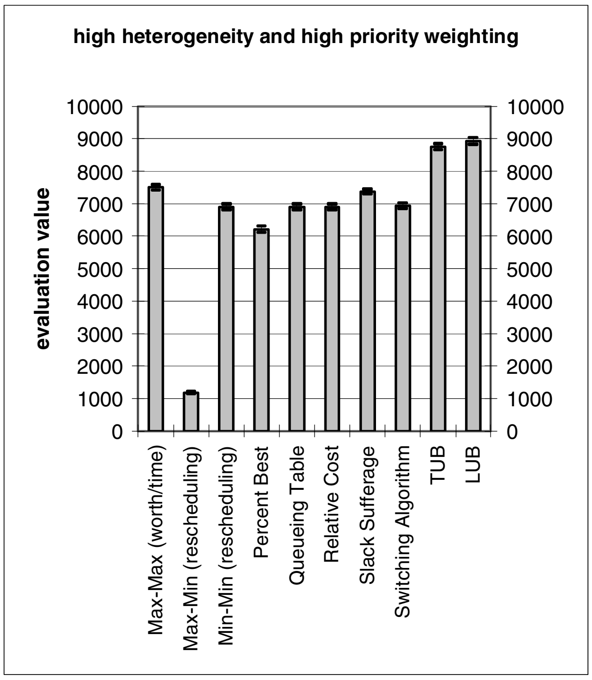
The ATC values are generated for the purpose of determining how well the heuristics perform when the actual task execution times on the machines vary from the estimated times in the ETC matrix. The ATC values are not known to the heuristics. For a given ETC matrix, ATC(*i*, *j*) is computed using ETC(*i*, *j*) as the mean and a COV of 0.1.

There are two types of priority weightings that are assigned to high, medium, and low priority level tasks, namely, sixteen, four, and one for the high priority weighting and four, two, and one for the low priority weighting. Of all the tasks that arrive, approximately one third will be of each priority level.

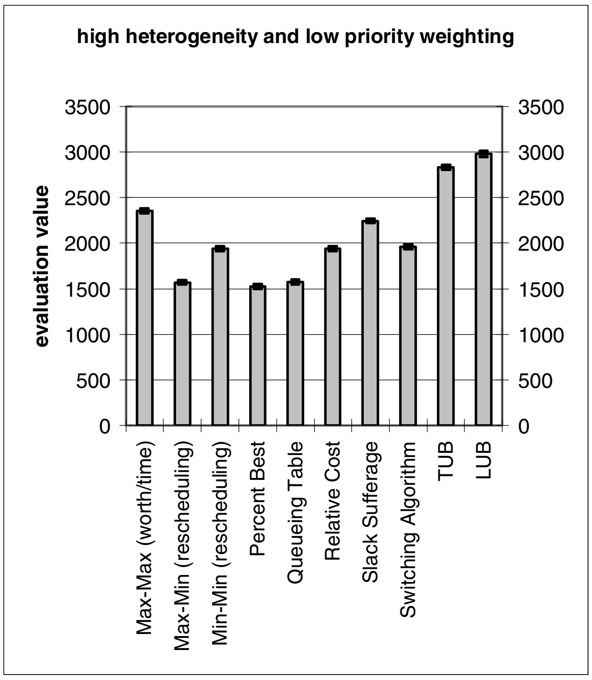
The deadline of each task is calculated using the following process. A deadline for each task is the arrival time of the task, plus the median execution time of the task (across all machines), plus a multiplier times the median execution time of all tasks (i.e., 2.4 minutes in this study). Two types of deadlines, i.e., loose and tight, are used in the simulation. The multiplier is changed to make the deadlines (i.e., the 100%, 50%, and 25% deadline) for the two types of deadlines. For the loose deadline, the multiplier is four, eight, and twelve for the primary (100%), 50%, and 25% deadline, respectively. For the tight deadline, the multiplier is one, two, and four for the primary (100%), 50%, and 25% deadline, respectively.

The simulation results are shown in Figures 1 and 2 for the two different types of deadlines. Each figure consists of four scenarios (all combinations of high/low heterogeneity and high/low priority weighting). The loose upper bound (LUB) shown is the simple bound that is the sum of the priority weightings of all tasks. All heuristics are run for 50 ETC and ATC matrices for each scenario (a total of 200 trials). The averages over 50 trials and the 95% confidence intervals are shown (most of the intervals are very close to the mean). The running time per mapping event of each heuristic is averaged over 200 trials, mapping 1250 tasks per trial on average. The average execution times of a mapping event for the heuristics are shown in Table 2.

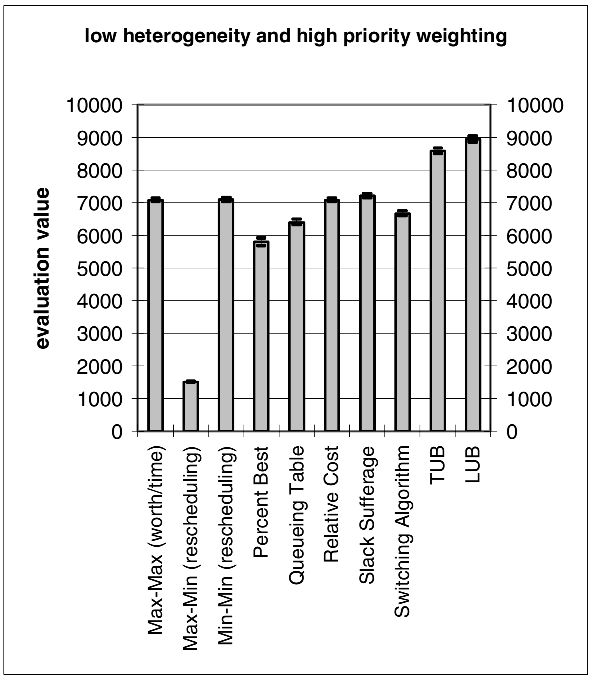
In Figure 1, simulation results using loose deadlines are shown. For the high heterogeneity cases, the Max-Max heuristic did the best (86% and 83% of the TUB for high and low priority weighting, respectively), while the Slack Sufferage heuristic was the best in the low heterogeneity cases (84% and 81% of the TUB for high and low priority weighting, respectively). The relative performance among the rest of the heuristics was similar in all the scenarios, with Max-Min performing the worst.



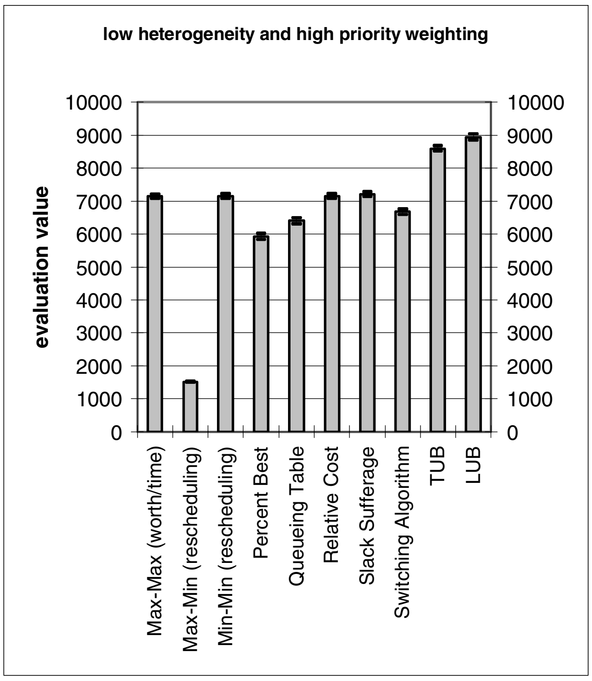
(a)



(b)

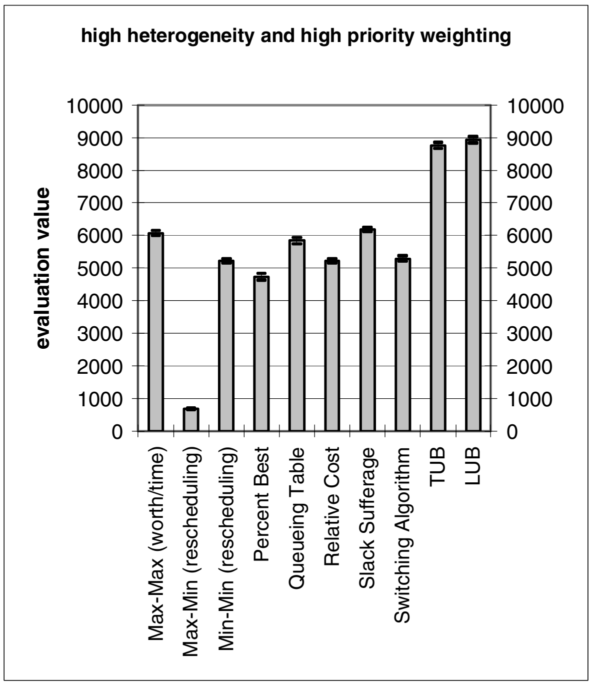


(c)

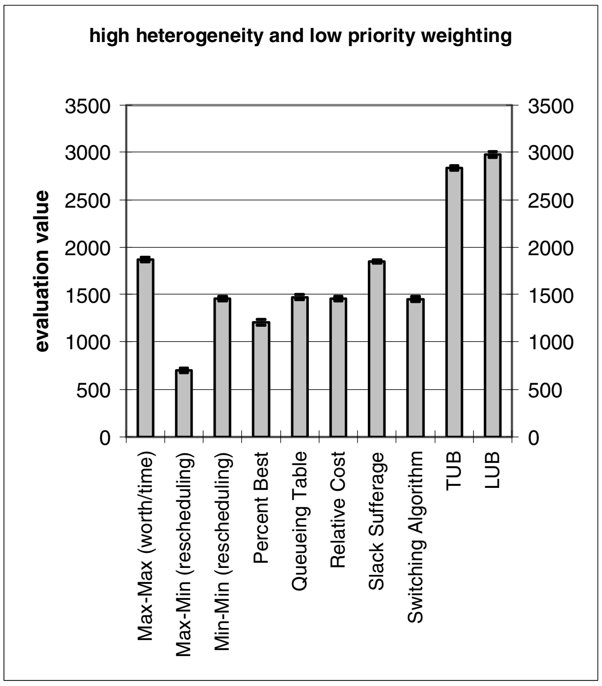


(d)

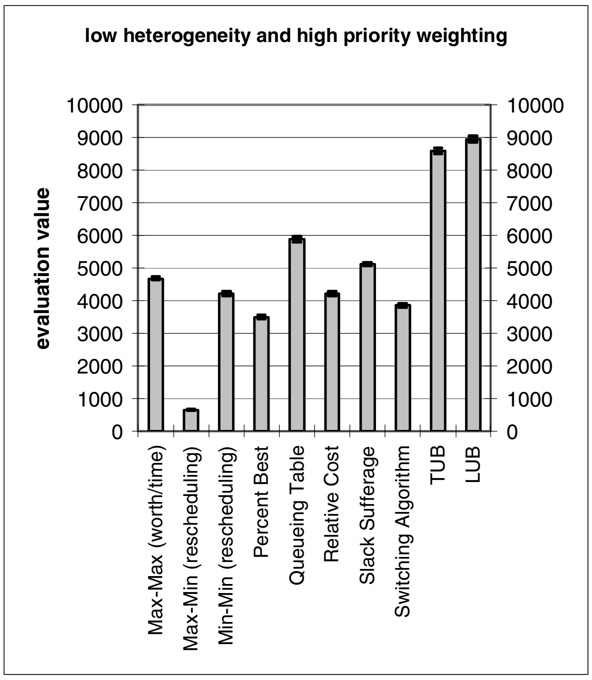
Figure 1: The simulation results using loose deadlines for (a) high heterogeneity with the high priority weighting of sixteen, four, and one for high, medium, and low priority levels, (b) high heterogeneity with the low priority weighting of four, two, and one for high, medium, and low priority levels, (c) low heterogeneity with the high priority weighting of sixteen, four, and one for high, medium, and low priority levels, and (d) low heterogeneity with the low priority weighting of four, two, and one for high, medium, and low priority levels.



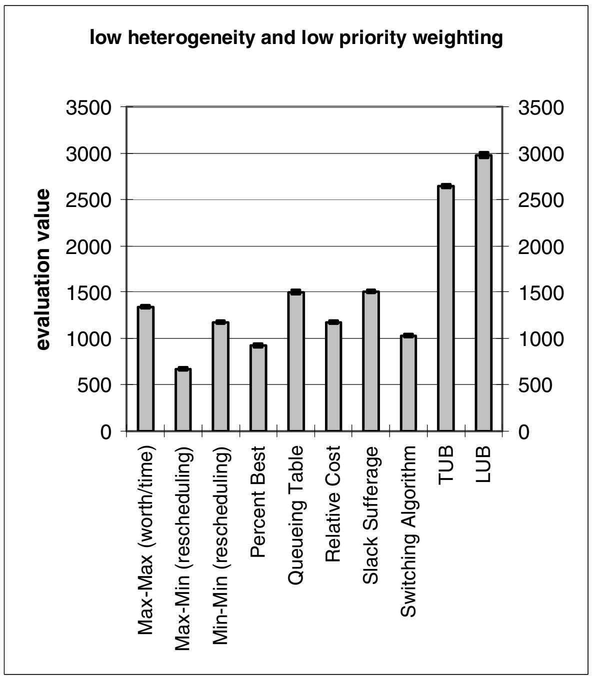
(a)



(b)



(c)



(d)

Figure 2: The simulation results using tight deadlines for (a) high heterogeneity with the high priority weighting of sixteen, four, and one for high, medium, and low priority levels, (b) high heterogeneity with the low priority weighting of four, two, and one for high, medium, and low priority levels, (c) low heterogeneity with the high priority weighting of sixteen, four, and one for high, medium, and low priority levels, and (d) low heterogeneity with the low priority weighting of four, two, and one for high, medium, and low priority levels.

In the high priority cases, there is less performance difference among all heuristics (excluding the Max-Min heuristic) than in the low priority cases. This is because all heuristics map the high priority tasks that can meet their primary deadline first and if the weighting of the high priority task is dominant, then there is less difference in performance.

heuristic	average execution time of a mapping event (in seconds)
Max-Max	0.11
Max-Min	0.45
Min-Min	0.35
Percent Best	0.44
Queueing Table	0.0004
Relative Cost	0.36
Slack Sufferage	0.18
Switching Algorithm	0.0002

Table 2: The average execution times of a mapping event for the eight heuristics.

In Figure 2, as can be expected, the performance of all heuristics degraded as tasks are more likely to miss their deadlines because of the tight deadlines. The relative performance among the heuristics remained the same (i.e., Max-Max and Slack Sufferage performed well) except for the Queueing Table method. The Queueing Table heuristic was the best in the low heterogeneity cases and the performance of the Queueing Table heuristic degraded the least from Figure 1 to Figure 2 for each of the scenarios. The Queueing Table method is one of the heuristics that explicitly uses urgency (i.e., nearness of deadline (NOD)) to order the execution of tasks in a machine queue and this accounts for the limited degradation. Percent Best and Switching Algorithm also use urgency to order the execution of tasks in a machine queue (ties are broken using the method of earlier primary deadline first, see Subsections 3.3 and 3.7), but they do not determine whether a task can finish before its primary deadline or not. In their mapping process, assuming tasks have the same priority weighting, tasks that cannot finish by their 25% deadline may be scheduled to execute in front of a task that can meet its primary deadline. This has a higher probability of occurring in the scenarios that use the tight deadline than in those that use the loose deadline, because tasks with the tight deadline have a higher probability of violating their deadlines. However, in the Queueing Table heuristic this will not happen because, if a task misses its primary deadline, then the NOD is set to infinity.

It is interesting to note that the relative performance of the Max-Max and Slack Sufferage heuristics changes according to the heterogeneity. In the high heterogeneity cases for both figures, Max-Max performs better than Slack Sufferage. However, in the low heterogeneity cases for both figures, Slack Sufferage performs better than Max-Max.

The following is an example of a high heterogeneity case where Max-Max will do better than Slack Sufferage. Assume that there are two tasks (t_1 and t_2) with the same priority and two machines (m_1 and m_2), where the machine availability times are 5 and 155 seconds respectively, and that the estimated execution times and deadlines are as shown in Table 3. Assume that when the primary deadline is not met, the 50% deadline will be met.

tasks	machines		primary deadline
	m_1	m_2	
t_1	38	20	160
t_2	3	10	10

Table 3: An example of tasks with high heterogeneity estimated execution times in seconds.

Using the information from the previous paragraph, the two tasks will miss their primary deadline on m_2. This means that the deadline factor will be 0.5 for both tasks on m_2 and the worth (priority weighting multiplied by deadline factor) will be half of that on m_1. Therefore, after calculating the fitness value, the Max-Max heuristic will determine the two task/machine pairs t_1/m_1 and t_2/m_1 in the first phase and pick the t_2/m_1 pair first to map and then pick the t_1/m_2 pair to map. The Slack Sufferage heuristic will first calculate the percentage slack for all task/machine pairs as shown in Table 4. Because the best machine is the same for both tasks, the task that is more critical is picked first. In this case, t_1 is picked and mapped to m_1. Another calculation of the percentage slack value for t_2 after t_1 is mapped indicates that t_2 will miss its primary deadline on both m_1 and m_2. The task t_2 in the mapping from the Slack Sufferage heuristic will violate its primary deadline, while the Max-Max heuristic completes both tasks before their primary deadline.

tasks	machines		primary deadline
	m_1	m_2	
t_1	0.75	−1	160
t_2	0.4	−1	10

Table 4: The calculation of the percentage slack values using Table 3.

The following is a low heterogeneity case where Slack Sufferage will do better than Max-Max. The ETC values of tasks have a higher probability of being similar in low versus high heterogeneity cases. The fitness value of a task on all machines calculated for the Max-Max heuristic may be similar. In the first phase of the Max-Max heuristic, the scheme determines the task/machine pair that has the maximum fitness value. Assume that the worth is the same on all machines (i.e., the deadline factor is the same for all machines). This means that when selecting the task/machine pair in the first phase of Max-Max only the ETC values will determine the pair. The calculation of the percentage slack value in the Slack Sufferage heuristic includes the machine availability time. Therefore, the best machine chosen by Slack Sufferage for a task can be different from that of the Max-Max heuristic. In this example, selecting the machine that gives a higher percentage slack for the task to map onto may give the task a higher probability of not violating its deadline rather than picking the most worth per unit time machine.

As an example of a low heterogeneity case where Slack Sufferage does better than Max-Max, assume that there are two tasks (t_1 and t_2) with the same priority and two machines (m_1 and m_2), where the machine availability times are 4 and 8 seconds respectively, and that estimated execution times and deadlines are as shown in Table 5. Assume that when the primary deadline is not met, the 50% deadline will be met.

tasks	machines		primary deadline
	m_1	m_2	
t_1	9	4.4	16
t_2	5	4	13

Table 5: An example of tasks with low heterogeneity estimated execution times in seconds.

The Max-Max heuristic will determine the two task/machine pairs t_1/m_2 and t_2/m_2 in the first phase (the worth of both tasks on both machines are the same) and pick the t_2/m_2 pair first to map. After mapping t_2 and the machine availability time is updated, if t_1 is mapped on m_1, it does not violate its primary deadline and if t_1 is mapped on m_2, it misses its primary deadline. However, the fitness value (calculated using the deadline factor of 0.5 for m_2) is higher for t_1 on m_2. Therefore, t_1 is mapped on m_2. Slack Sufferage will first calculate the percentage slack for all task/machine pairs as shown in Table 6. In this case, t_1 is mapped onto m_2 and t_2 is mapped onto m_1. Slack Sufferage finishes both tasks by their primary deadline and completes the later task by time 12.4. Max-Max completes the later task by time 16.4 and t_1 misses its primary deadline.

The Max-Min heuristic is the worst in all four scenarios and with both types of deadlines. This is because Max-Min tries to map the maximum in the second phase of the heuristic (see Subsection 3.2). When trying to map the difficult tasks (tasks with longer completion times) earlier, the tasks with short execution times are waiting for the longer running tasks to finish. In the time one difficult task finishes, multiple short tasks could have completed. Assuming the tasks have the same priority weightings and same deadline factors, completing multiple short tasks in an interval of time would indicate higher performance than completing one large task according to the performance metric in this research. The Min-Min heuristic, which is a variation of the Max-Min heuristic, maps the shorter task first and performs better than Max-Min.

tasks	machines		primary deadline
	m_1	m_2	
t_1	0.25	0.45	16
t_2	0.44	0.2	13

Table 6: The calculation of the percentage slack values using Table 5.

5. Summary

Eight heuristics were designed, developed, and simulated using the HC environment presented. Dynamically arriving tasks with priorities and multiple deadlines were mapped using the heuristics proposed in this research.

When loose deadlines were used, the Max-Max heuristic did the best in the high heterogeneity case and the Slack Sufferage heuristic was the best in the low heterogeneity case. When tight deadlines were used, the performance of all heuristics is degraded. In the high heterogeneity cases, Max-Max and Slack Sufferage are still the heuristics of choice, however in the low heterogeneity cases, Queueing Table (that uses urgency in its mapping process) performed the best. However, if the time to produce a solution is a crucial constraint, the Queueing Table (when using tight deadline) and the Switching Algorithm (when using loose deadline) are recommended.

Acknowledgements: The authors thank Shoukat Ali for his valuable comments.

References

[1] S. Ali, J.-K. Kim, H. J. Siegel, A. A. Maciejewski, Y. Yu, S. B. Gundala, S. Gertphol, and V. Prasanna, "Greedy heuristics for resource allocation in dynamic distributed real-time heterogeneous computing systems," *2002 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA 2002*), June 2002, pp. 519-530.

[2] 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).

[3] H. Barada, S. M. Sait, and N. Baig, "Task matching and scheduling in heterogeneous systems using simulated evolution," 10th IEEE Heterogeneous Computing Workshop (HCW 2001), in the CD-ROM "*Proceedings of the 15th International Parallel and Distributed Processing Symposium* (*IPDPS 2001*)," paper HCW 15, Apr. 2001.

[4] I. Banicescu and V. Velusamy, "Performance of scheduling scientific applications with adaptive weighted factoring," 10th IEEE Heterogeneous Computing Workshop (HCW 2001), in the CD-ROM "*Proceedings of the 15th International Parallel and Distributed Processing Symposium* (*IPDPS 2001*)," paper HCW 06, Apr. 2001.

[5] T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Heterogeneous computing: Goals, methods, and open problems," *2001 International Conference on Parallel and Distributed Processing Techniques and Applications* (*PDPTA 2001*), June 2001, pp. 1–12 (invited keynote paper).

[6] T. D. Braun, H. J. Siegel, N. Beck, L. Boloni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and Bin Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, Vol. 61, No. 6, June 2001, pp. 810-837.

[7] C. D. Cavanaugh, L. R. Welch, B. A. Shirazi, E. Huh, and S. Anwar, "Quality of service negotiation for distributed, dynamic real-time systems," *Parallel and Distributed Processing*, J. Rolim et al. eds., Lecture Notes in Computer Science, Vol. 1800, pp. 757–765, Springer-Verlag, New York, NY, 2000.

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

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

[10] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transaction on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

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

[12] R. F. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel, "Scheduling resources in multiuser, heterogeneous, computing environments with SmartNet," *7th IEEE Heterogeneous Computing Workshop* (*HCW 1998*), Mar. 1998, pp. 184–199.

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

[14] E. Huh, L. R. Welch, B. A. Shirazi, B. Tjaden, and C. D. Cavanaugh, "Accommodating QoS prediction in an adaptive resource management framework," *Parallel and Distributed Processing*, J. Rolim et al. eds., Lecture Notes in Computer Science, Vol. 1800, pp. 792-799, Springer-Verlag, New York, NY, 2000.

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

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

[18] M. Maheswaran, T. D. Braun, and H. J. Siegel, "Heterogeneous distributed computing," *Encyclopedia of Electrical and Electronics Engineering*, *Vol. 8*, J. G. Webster, ed., pp. 679-690, John Wiley, New York, NY, 1999.

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

[20] K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Transactions on Computers*, Vol. 38, No. 8, Aug. 1989, pp. 1110-1123.

[21] L. R. Welch, B. Ravindran, B. A. Shirazi, and C. Bruggeman, "Specification and modeling of dynamic, distributed real-time systems," *19th IEEE Real-Time Systems Symposium* (*RTSS '98*), Dec. 1998, pp. 72-81.

[22] L. R. Welch, B. A. Shirazi, B. Ravindran, and C. Bruggeman, "DeSiDeRaTa: QoS management technology for dynamic, scalable, dependable, real-time systems," *Distributed Computer Control Systems 1998*, F. De Paoli and I. M. MacLeod, eds., pp. 7-12, Kidlington, UK (Proceedings volume from the 15th International Federation of Automatic Control (IFAC) Workshop, Sep. 1998), Elsevier Science, 1999.

[23] L. R. Welch and B. A. Shirazi, "A dynamic real-time benchmark for assessment of QoS and resource management technology," *5th IEEE Real-Time Technology and Applications Symposium* (*RTAS '99*), June 1999, pp. 36-45.

[24] L. R. Welch, P. V. Werme, B. Ravindran, L. A. Fontenot, M. W.Masters, D. W. Mills, and B. A. Shirazi, "Adaptive QoS and resource management using a-posteriori workload characterizations," *5th IEEE Real-Time Technology and Applications Symposium* (*RTAS '99*), June 1999, pp. 266-275.

[25] 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.

[26] V. Yarmolenko, J. Duato, D. K. Panda, and P. Sadayappan, "Characterization and enhancement of dynamic mapping heuristics for heterogeneous systems," *International Workshop on Parallel Processing*, Aug. 2000, pp. 437-444.

Biographies

Jong-Kook Kim is pursuing a Ph.D. degree from the School of Electrical and Computer Engineering at Purdue University, where he has been a Research Assistant since

August 1998. Jong-Kook received his M.S. degree in electrical engineering from Purdue University in May 2000. His Master's thesis was "A Multi-Dimensional Performance Measure for Distributed Computing and Communication Systems." He received his B.S. degree in electronic engineering from Korea University, Seoul, Korea in 1998. His research interests include heterogeneous distributed computing, parallel computing, computer architecture, performance measures, resource management, evolutionary heuristics, and grid computing. He is a student member of the IEEE, IEEE Computer Society, and ACM.

Sameer Shivle is a graduate student of Colorado State University pursuing his M.S. degree in Electrical and Computer Engineering. He received a B.E. degree in electrical engineering from the Government College of Engineering, Pune, India. His fields of interest are heterogeneous computing, computer architecture and digital system design.

H. J. Siegel holds the endowed chair position of Abell Distinguished Professor of Electrical and Computer Engineering at Colorado State University (CSU), where he is also a Professor of Computer Science. He is the Director of the CSU Information Science and Technology Center (ISTeC). ISTeC a university-wide organization for promoting, facilitating, and enhancing CSU's research, education, and outreach activities pertaining to the design and innovative application of computer, communication, and information systems. Prof. Siegel is a Fellow of the IEEE and a Fellow of the ACM. From 1976 to 2001, he was a professor in the School of Electrical and Computer Engineering at Purdue University. He received a B.S. degree in electrical engineering and a B.S. degree in management from the Massachusetts Institute of Technology (MIT), and the M.A., M.S.E., and Ph.D. degrees from the Department of Electrical Engineering and Computer Science at Princeton University. He has co-authored over 300 technical papers. His research interests include heterogeneous parallel and distributed computing, communication networks, parallel algorithms, parallel machine interconnection networks, and reconfigurable parallel computer systems. 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 was Program Chair/Co-Chair of three major international conferences, General Chair/Co-Chair of four international conferences, and Chair/Co-Chair of five workshops. He is currently on the Steering Committees of five continuing conferences/workshops. He is a member of the Eta Kappa Nu electrical engineering honor society, the Sigma Xi science honor society, and the Upsilon Pi Epsilon computing sciences honor society.

Anthony A. Maciejewski received the B.S.E.E, M.S., and Ph.D. degrees in Electrical Engineering in 1982, 1984, and 1987, respectively, all from The Ohio State University under the support of an NSF graduate fellowship. From 1985 to 1986 he was an American Electronics Association Japan Research Fellow at the Hitachi Central Research Laboratory in Tokyo, Japan where he performed work on the development of parallel processing algorithms for computer graphic imaging. From 1988 to 2001, he was a Professor of Electrical and Computer Engineering at Purdue University. In 2001, he joined Colorado State University as a Professor of Electrical and Computer Engineering. Prof. Maciejewski's primary research interests relate to the analysis, simulation, and control of robotic systems and he has co-authored over 100 published technical articles in these areas. He is an Associate Editor for the *IEEE Transactions on Robotics and Automation*, a Regional Editor for the journal *Intelligent Automation and Soft Computing*, and was co-guest editor for a special issue on "Kinematically Redundant Robots" for the *Journal of Intelligent and Robotic Systems*. He serves on the IEEE Administrative Committee for the Robotics and Automation Society and was the Program Co-Chair (1997) and Chair (2002) for the International Conference on Robotics and Automation, as well as serving as the Chair and on the Program Committee for numerous other conferences.

Tracy D. Braun received his Ph.D. in Electrical and Computer Engineering from the School of Electrical and Computer Engineering at Purdue University in 2001. In 1997, he received his M.S.E.E. from the School of Electrical and Computer Engineering at Purdue University. He received a B.S. in Electrical and Computer Engineering with Honors and High Distinction from the University of Iowa in 1995. He is a member of the IEEE, IEEE Computer Society, and Eta Kappa Nu honorary society. He has worked in industry at Norand Data Systems, Silicon Graphics/Cray Research, Noemix, and GridIQ. Dr. Braun has published more than 20 technical papers, has presented his work at several international conferences, and has been a reviewer for numerous conferences and journals. His research interests include scheduling, distributed computing, information assurance, and computer security. He is currently an adjunct faculty member teaching for the University of Maryland University College - Asia Division.

Myron Schneider currently works as a hardware design engineer for Agilent Technologies Manufacturing Test Business Unit in Loveland, Colorado. He received a B.S. in Electrical Engineering from the University of Illinois at

Urbana-Champaign in August 1996 and a Masters of Electrical Engineering from Colorado State University in December 2002. His technical interests and work experience include FPGA/CPLD design, hardware description languages, high-speed digital system design, re-configurable computing, computer architecture, heterogeneous computing, adaptive algorithms, and automated manufacturing test systems.

Sonja Tideman received the B.S. in Computer Science from the University of New Mexico. She is currently pursuing a M.S. degree in Computer Science at Colorado State University. She is employed by Sandia National Laboratory in Albuquerque, NM. Her research interests include computer security, networking, and operating systems.

Ramakrishna Chitta is a Computer Science major pursuing his M.S. at Colorado State University, where he is currently a Graduate Teaching Assistant. He received his B.Tech degree in Computer Science and Engineering from Jawaharlal Nehru Technological University, Hyderabad, India in 2001. His fields of specialization are compilers and computer architecture. He is a member of ACM.

Raheleh B. Dilmaghani received her B.S. in Electrical Engineering from University of Tehran, Iran in 1996 (graduated Supra Cum Laude). Since graduation, she has acted as a technical lead for the Moshanir Engineering Consulting Firm. She is currently a Master's degree candidate with the Electrical and Computer Engineering Department at Colorado State University. Her current interests and areas of research are in computer networking, security, and heterogeneous computing environments.

Rohit S. Joshi is pursuing a M.S. degree in Electrical and Computer Engineering at Colorado State University. He received his B.S. in Electrical Engineering from University of Pune, India in May 2000. His research interests include VLSI system design, microprocessor based systems, and power systems.

Aditya Kaul is currently pursuing his Ph.D. in Industrial Engineering and Operations Research at the Harold Inge Marcus Department of Industrial and Manufacturing Engineering at The Pennsylvania State University. He received his Master's degree in Electrical Engineering from Colorado State University in August 2002 and B.S. in Electrical Engineering from Regional Engineering College, Surat, India in August 2000.

Ashish Sharma is pursuing an M.S.E.E. degree in the Department of Electrical and Computer Engineering at Colorado State University. He received a B.E. degree in Electronics and Power Engineering from Nagpur University, India in 2000. His research interests include heterogeneous computing, fault tolerant computing, and VLSI design.

Siddhartha Sripada is a graduate student in the Department of Electrical and Computer Engineering at Colorado State University. He received a B.Tech degree in Electrical and Electronics Engineering from Nagarjuna University, India. His research interests include computer architecture, digital design, and heterogeneous computing. He has done projects in the fields of digital system design, heterogeneous computing, and fault tolerant computing. He is a student member of IEEE and an active member of the Nagarjuna University alumni.

Praveen Vangari is a graduate student of Colorado State University pursuing an M.S. degree in Electrical and Computer Engineering. He received his B.E. degree in the Vasavi College of Engineering (affiliated to the Osmania University, Hyderabad) in the field of Electronics and Communication Engineering. His research interests include computer architecture, system level hardware design, and VLSI.

Siva S. Yellampalli received his B.Tech in Electrical and Electronics Engineering from Jawaharlal Nehru Technological University in 2001. He is currently pursuing an M.S. degree in VLSI design at Louisiana State University. His research interests include IDDQ testing in nanometer technology, mixed signal design, and computer architecture.