Time Utility Functions for Modeling and Evaluating Resource Allocations in a Heterogeneous Computing System

Luis Diego Briceño[1], Bhavesh Khemka[1], Howard Jay Siegel[1,2], Anthony A. Maciejewski[1], Christopher Groër[3], Gregory Koenig[3], Gene Okonski[4], and Steve Poole[3,4]

Colorado State University
[1] Department of Electrical and Computer Engineering
[2] Department of Computer Science
Fort Collins, CO 80523

[3]Oak Ridge National Laboratory
Oak Ridge, TN 37830

[4]Department of Defense
Washington, DC 20001

email: {LDBricen, Bhavesh.Khemka, HJ, AAM}@colostate.edu
{GroerCS, Koenig, SPoole}@ornl.gov
okonskitg@verizon.net

***Abstract*—This study considers a heterogeneous computing system and corresponding workload being investigated by the Extreme Scale Systems Center (ESSC) at Oak Ridge National Laboratory (ORNL). The ESSC is part of a collaborative effort between the Department of Energy (DOE) and the Department of Defense (DoD) to deliver research, tools, software, and technologies that can be integrated, deployed, and used in both DOE and DoD environments. The heterogeneous system and workload described here are representative of a prototypical computing environment being studied as part of this collaboration. Each task can exhibit a time-varying *importance* or *utility* to the overall enterprise. In this system, an arriving task has an associated priority and precedence. The priority is used to describe the importance of a task, and precedence is used to describe how soon the task must be executed. These two metrics are combined to create a utility function curve that indicates how valuable it is for the system to complete a task at any given moment. This research focuses on using time-utility functions to generate a metric that can be used to compare the performance of different resource schedulers in a heterogeneous computing system. The contributions of this paper are: (a) a mathematical model of a heterogeneous computing system where tasks arrive dynamically and need to be assigned based on their priority, precedence, utility characteristic class, and task execution type, (b) the use of priority and precedence to generate time-utility functions that describe the value a task has at any given time, (c) the derivation of a metric based on the total utility gained from completing tasks to measure the performance of the computing environment, and (d) a comparison of the performance of resource allocation heuristics in this environment.**

This work was supported by the United States Department of Defense and used resources of the Extreme Scale Systems Center at Oak Ridge National Laboratory. This research also used the CSU ISTeC HPC System supported by NSF Grant CNS-0923386.

I. Introduction

In certain High Performance Computing systems, a common metric for evaluating the performance of the system and the scheduler is observing the system utilization. In a *homogeneous* computing system (where task execution times do not vary across machines), this metric intuitively makes sense because the execution time of a task will be the same on any machine to which it is assigned. However, a high utilization of a *heterogeneous* computing system (where task execution times may vary across machines) does not necessarily imply the system is being used in the most effective way. The question that needs to be answered in some heterogeneous environments becomes: How do we measure system performance?

This study presents a simulation of an oversubscribed computing environment with system characteristics and workload parameters based on the expectations for future environments of DOE and DoD interest. The performance of the target system is dependent on how many tasks are executed, how much useful work is done, and how timely this information is to the customer. In the target system, tasks can have several different levels of *priority*, i.e., how important is it for the task to be computed. Additionally, the *precedence* of a task indicates how quickly it loses utility. The value of higher precedence tasks decay more rapidly with time. The priority and precedence are used to create a time-utility function that calculates the utility a task will earn if it is completed at a specific time (e.g., [17], [33]). In our simulation studies, each dynamically arriving

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.123

task will belong to one of a fixed number of "utility characteristic classes," and to one of a fixed number of "task execution types." The utility characteristic class is used in conjunction with the priority and precedence of a task to generate a time-utility function [17] that specifies the utility a task will achieve based on when it completes. Task execution types are groups of tasks with similar computational characteristics. In our model, the workload consists of serial tasks (i.e., non-parallelizable) that are independent (i.e., there is no inter-task communication). A more detailed description of these task features are discussed in Section II-D.

In this paper, we use time-utility functions to measure and quantify the performance of the resource allocation heuristics in the system. In the target system, tasks arrive dynamically and the resource allocation heuristic does not know ahead of time when the next task will arrive, what utility characteristic class it will have, what the task's priority will be, what task execution type it will have, and what the task's precedence level will be. Therefore, a utility function for a specific task is only known after it arrives. In this study, the goal of the heuristics is to maximize the total system utility, which is the sum over all tasks of each task's accrued utility (described in detail in Section II).

The contributions of this paper are: (a) a mathematical model of a heterogeneous computing system where tasks arrive dynamically and need to be assigned based on their priority, precedence, utility characteristic class, and task execution type, (b) the use of priority and precedence to generate time-utility functions that describe the value a task has at any given time, (c) the derivation of a performance metric based on the total utility gained from completing tasks, and (d) a comparison of the performance of resource allocation heuristics in this environment.

This paper is organized as follows. The next section is a description of the time utility functions. Section III explains the problem statement. Section IV gives a description of the resource allocation heuristics we used. A comparison of our work with other related work on time-utility functions is given in Section V. In Section VI, we describe how the simulations are structured. The results are shown in Section VII, and the conclusions are in Section VIII.

II. Time Utility Functions

A. Overview

Time-utility functions are created to quantify the utility of completing each task at a given time. This time utility function is based on priority, precedence, and utility characteristic class. These utility functions model the importance of tasks to the overall enterprise and are based on discussions with system designers. All the utility functions we consider are monotonically decreasing. However, in general, these utility functions can have any arbitrary shape. We do not use arbitrary shapes in an effort to avoid overwhelming the users by having them specify a utility value at any given point in time. Therefore, the goal is to use a small set of parameters that the users understand and that enables them to obtain the desired utility curve.

We designed the time-utility function framework used in this simulation environment to be flexible. Given that the characteristics of the tasks used in real systems are unknown, it is important to be able to create a large number of possible utility characteristic classes. While the precedence and priority determine the basic information about the utility function, the utility characteristic class allows us to fine tune the basic utility function to obtain the final utility function. This provides flexibility in our model.

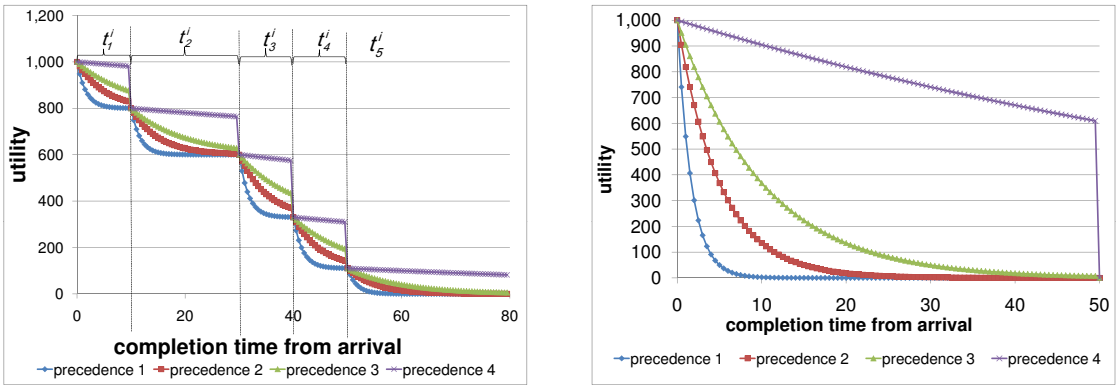
B. Priority

The priority of a task is based on the importance it has to the organization. It is assigned to each task by the system owner in collaboration with the customer submitting the task. Each priority level has an associated maximum "utility" level that is used to compare tasks against others in different priority groups. The maximum "utility" of tasks with the same priority level is the same. We assume there are four priority levels (critical, high, medium, and low). The critical tasks are uncommon (less than 5% of tasks in our simulation study); however, the overall utility of the system is heavily influenced by the timely execution of these tasks. The other priorities are more common.

In our model, π_1 represents the maximum utility of critical priority tasks, π_2 the maximum utility of high priority tasks, π_3 the maximum utility of medium level tasks, and π_4 the maximum utility of low priority tasks. The difference among the maximum utility of priority levels is exponential, e.g., high priority can have a utility of 100 and medium priority a utility of 10.

C. Precedence

The precedence of a task is based on how soon the output of the task is needed. The precedence is represented by the "shape" of the "utility" function. The system uses four levels of precedence, where precedence level i indicates a faster decline in utility over time than level $i+1$. The precedence of a task is determined by the designer/programmer. In this study, we assume exponential decay rates, where $\beta(L)$ represents the exponential decay rate of tasks with a precedence level of L.



(a) Utility function that is composed of five different time intervals is illustrated.

(b) Utility function that consist of one time interval is illustrated.

Fig. 1. Examples of time-utility functions for different precedence levels for a utility characteristic class i are shown.

D. Utility Characteristic Class

The utility function associated with a task is based on the task's precedence, priority, and utility characteristic class. For our study, there are four different utility characteristic classes.

Each utility characteristic class i is composed of three different elements. The first element is the set of τ_i time intervals used to partition the time axis of the utility function. The start time for the kth interval, $t(k,i)$ is relative to the arrival time of that task.

The second element of a utility characteristic class i is the percentage of maximum utility at the beginning of a time interval k, $u(k,i)$, where $0 < u(k,i) \leq 1$. The maximum utility for a utility characteristic class i with priority x in time interval k is

$$u(k,i,x) = u(k,i) \cdot \pi(x) \tag{1}$$

The last element is a modifier $\Gamma(k,i)$ for the exponential decay in each time interval k ($1 \leq k \leq \tau_i$); in our studies, $0.8 \leq \Gamma(k,i) \leq 1.2$. The precedence decay modifier for a utility characteristic class i with a precedence level of ρ at an interval k is therefore

$$\alpha(k,i,\rho) = \beta(\rho) \cdot \Gamma(k,i). \tag{2}$$

For task h, let $x \in \{critical, high, medium, low\}$ represent the priority level, $\rho \in \{1,2,3,4\}$ represent the precedence level, and i represents the utility characteristic class. The utility of this task at time t (which is relative to the arrival time of task h), where $t(k,i) \leq t < t(k+1,i)$ is

$$U_{i,\rho,x}(t) = \frac{u(k,i,x) - u(k+1,i,x)}{e^{\alpha(k,i,\rho)\cdot(t-t(k,i))}} + u(k+1,i,x). \tag{3}$$

An example of the utility functions for different precedence levels is shown in Figure 1. The utility function shown in Figure 1(a) has five intervals, while Figure 1(b) only has one interval.

E. Model of Environment

The target system simulated here has M heterogeneous machines (M = 100) and 10,000 tasks. In our simulation study, we consider two types of machines: general purpose and special purpose machines. The general purpose machines are capable of running most tasks but do not offer significant advantages in execution time. In contrast, the special purpose machines can execute a few tasks with a significant improvement in execution time. In this simulation environment, we assume that tasks *cannot* be preempted.

For each task execution type i, we assume that an Estimated Time to Compute on each machine j has been provided, denoted $\boldsymbol{ETC(i,j)}$. The generation of ETC values for our simulation is explained in Section VI-B. The relative values of the entries in the ETC matrix represent how "heterogeneous" the environment is. This assumption is common in the literature (e.g., [3], [10], [12], [19], [22], [37], [42]).

In this environment, the workload consists of a small set of task execution types. Therefore, it is reasonable to collect information about the running times of these task execution types. We assume that the initial ETC values can be determined by doing experiments, and then refined as the scheduler accumulates historical data.

Due to the nature of special purpose machines in our study, certain tasks cannot execute on some machines. In our simulation model, if task h cannot execute on machine j then $ETC(h,j) = \infty$.

F. Performance Metric

Let $t_{finished}(h)$ represent the time when task h finishes executing, $U_{h\rightarrow\{i,\rho,x\}}(t_{finished}(h))$ represent the utility that task h obtained, and $\Omega(t)$ represent the set of tasks that have finished execution by time t. The measure of performance (*system utility*) in this heterogeneous computing system is defined as

$$u_{system}(t) = \sum_{h\in\Omega(t)} U_{h\rightarrow\{i,\rho,x\}}(t_{finished}(h)). \quad (4)$$

III. Problem Statement

In this study, we evaluate the performance of different resource allocation heuristics in the computing system environment described in Sections I and II. The goal of each heuristic is to maximize the performance metric described in Equation 4. We evaluate the heuristics considering two types of heterogeneous environments: inconsistent and partially-consistent (defined in Section VI-B), and two different arrival rates.

IV. Heuristics

A. Types of Dynamic Mode Heuristics

There are several types of dynamic mode (also known as online [25]) heuristics in the literature. The two types we consider in this study are the immediate mode and batch mode heuristics [30]. The immediate mode heuristics assign incoming tasks to machines as soon as they arrive, and their assignment is never changed.

For batch mode heuristics, there is a *virtual queue* of tasks that is composed of tasks that have arrived but are not pending or currently executing (as shown in Figure 2). This virtual queue is stored at the scheduler. Tasks that are in the virtual queue can be reassigned in subsequent mapping events. This contrasts with tasks that are pending or executing that cannot be reassigned in subsequent mapping events. These batch mode heuristics execute periodically after a given time interval (i.e., one minute in this simulation study). The batch mode heuristics gather incoming tasks before allocating these tasks and any tasks in the virtual queue to machines.

In this study, we do not allow a task to be preempted. The optimal use of preemption adds to the complexity of the scheduling system. Additionally, the use of batch mode heuristics will minimize the advantage gained by preempting tasks. In future studies, it may be possible to incorporate preemption to maximize the total system utility.

B. Immediate Mode Heuristics

1) Round Robin

The Round Robin assigns the first task that arrives for a mapping event to the first machine, the second task to the second machine, and so on. The algorithm for this heuristic is given in Figure 3.

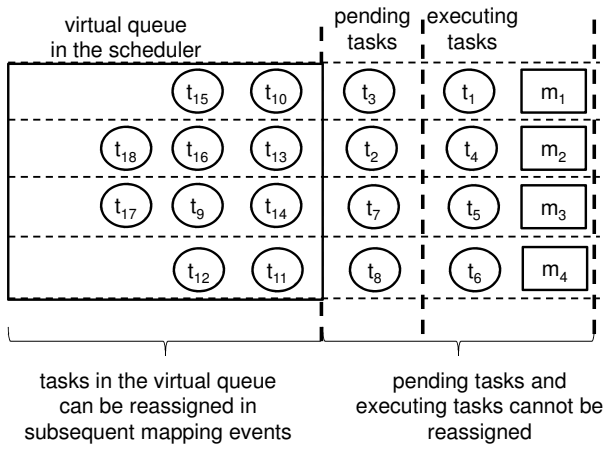


Fig. 2. Illustration of the virtual queue used in the batch mode heuristics. Row i corresponds to tasks on machine i

2) Random

As the name suggests, the Random heuristic randomly assigns an incoming task to some machine. The procedure used for the Random heuristic is shown in Figure 4. The Round Robin and the Random heuristic are used as baseline heuristics for comparison purposes.

3) Maximum Utility

The Maximum Utility heuristic is a greedy heuristic that picks the machine that will obtain the highest utility for each task. For this task, the machine that obtains the maximum utility will also have the earliest completion time. Therefore, this heuristic is similar to the Minimum Completion Time (MCT) heuristic [5]–[7], [16], [30], [31], [35], [38]. The procedure used to assign tasks with the Maximum Utility heuristic is shown in Figure 5.

The Maximum Utility heuristic takes into consideration not only the execution times of the task h on the machines but also the ready times of the machines (when a machine will complete all tasks already assigned to it). Once an assignment has been made, the machine ready time of that machine is updated. The tasks are considered for mapping in the order of their arrival. The drawback of this heuristic is that there is no guarantee that tasks will be assigned to the machines with their fastest execution times (versus earliest completion time).

4) Minimum Execution Time (MET)

The Minimum Execution Time heuristic [5]–[7], [16], [30], [31] is a greedy heuristic that assigns the current task to the machine that executes it the fastest. The tasks are considered for assignment in the order of their arrival, and each task is assigned to the machine that can execute it the quickest. This heuristic is sometimes referred to as "user directed assignment" because the users will want their program to run on the fastest

(1) Let $count$ be a counter using for assigning tasks to machines, and set its initial value to one.
(2) Consider next task h that just arrived.
(3) If task h can execute on machine ($count$ mod M) + 1, then assign task h to ($count$ mod M) + 1.
(4) If task h cannot execute on machine ($count$ mod M) + 1, then increase $count$ until task h is able to execute. Thus, machines that are skipped in this step must wait until the next pass through the list of machines to have a task assigned to it.
(5) $count = count + 1$, for the next mapping event.
(6) Go to step (2).

Fig. 3. Procedure for using the Round Robin heuristic to generate a resource allocation.

(1) Consider task h that just arrived and the set of machines that can execute h.
(2) Assign task h to a randomly selected machine from the set of machines that can execute it.

Fig. 4. Procedure for using the Random heuristic to generate a resource allocation.

(1) Consider task h that just arrived.
(2) Assign task h to the machine that can obtain the highest utility.
(3) Update the machine ready time.

Fig. 5. Procedure for using Maximum Utility heuristic to generate a resource allocation.

machine.

The procedure for the Minimum Execution Time heuristic is shown in Figure 6. The Minimum Execution Time heuristic only considers the execution times for the task on the machines and ignores the ready times of the machines (earliest time when that task can begin execution on that machine). The drawback of Minimum Execution Time is that it can lead to only a few machines being used, and this can lead to long queues on fast machines.

5) K-percent Best

The K-percent Best algorithm [1], [6], [7], [13], [23], [30], [31] combines both the MET and the Maximum Utility heuristics by choosing machines based on both these heuristics. When a new task arrives, the K-percent Best algorithm picks a machine in two stages. In the first screening process, it picks K-percent of the best machines ($M \cdot K/100$) that give the lowest execution times among all the machines. In the next step, among the machines chosen in the first step, it picks the machine that gives the maximum utility. In this way, K-percent Best picks a machine that is partially an MET machine and partially a Maximum Utility machine as well. The K parameter controls the extent to which the K-percent Best algorithm is biased toward the MET or the Maximum Utility machine. For example, if K is set to 100 then in the first stage all the machines would be selected and the result of K-percent Best would be to choose the maximum utility machine. However, if K is set to 100/M, then in the first stage only one machine is picked and that is the MET machine. The procedure for the K-percent Best heuristic is shown in Figure 7. In our implementation, the K parameter was empirically determined to be 50, that is, 50% of the MET machines were chosen in the first phase.

C. Batch Mode Heuristics

1) Min-Min Completion Time

The Min-Min heuristic [6], [7], [11], [13], [16], [18], [21], [23], [24], [30], [35], [36], [38], [41] is a two phase greedy heuristic. In the first phase, it independently finds the minimum completion time machine for every task. In the next phase, for every task-machine pair found in the first phase, it picks the pair that has the earliest completion time (second Min). It then assigns that task to that machine. The procedure used to implement Min-Min Completion Time is shown in Figure 8.

2) Max-Max Utility

The Max-Max heuristic [7], [23], [31], [38] is similar in principle to the Min-Min heuristic. The only difference is that we maximize the utility in each of the two phases. The procedure for Max-Max Utility is shown in Figure 9.

3) Sufferage

The Sufferage heuristic [4], [6], [8], [11], [14], [18], [21], [23], [24], [30], [32], [34], [35], [40] attempts to maximize the utility earned by assigning every task to the machine that would give the highest utility, but when multiple tasks want the same machine it gives preference to the task that would "suffer" the most if it were not assigned to that machine. If $max_{utility}(h)$ is the best obtainable utility of task h, and $smax_{utility}(h)$ is the second highest utility obtainable for task h, then the sufferage value (*suff*) is $suff = max_{utility}(h) - smax_{utility}(h)$. The procedure of the Sufferage heuristic is described in Figure 10.

V. Related Work

The authors in [17] present a system where processes are running on a simulated, symmetric, shared-memory

(1) Consider task h that just arrived.
(2) Assign task h to the machine with the smallest estimated time to compute.

Fig. 6. Procedure for using the Minimum Execution Time (MET) heuristic to generate a resource allocation.

(1) Consider task h that just arrived.
(2) Determine the K-percent fastest execution time machines for task h.
(3) From this set of K-percent best machines, assign h to the machine that can obtain the highest utility.
(4) Update the machine ready time.

Fig. 7. Procedure for using the K-percent Best heuristic to generate a resource allocation.

(1) Make a set composed of the unmapped tasks.
(2) For each task h in the set of unmapped tasks, determine the machine that has the minimum completion time.
(3) From the task/machine pairs found in (2), determine the task/machine pair (task h, machine j) with the overall smallest completion time.
(4) Assign task h to machine j, and update the machine ready time.
(5) Remove h from set of unmapped tasks.
(6) Repeat steps 2 through 5 until all tasks are mapped.

Fig. 8. Procedure for using the Min-Min Completion Time heuristic to generate a resource allocation.

(1) Make a set composed of the unmapped tasks.
(2) For each task h in the set of unmapped tasks, determine the task/machine pair that obtains the maximum utility time.
(3) From the machine found in (2), determine the task/machine pair (task h, machine j) with the overall maximum utility time.
(4) Assign task h to machine j, and update the machine ready times.
(5) Remove h from set of unmapped tasks
(6) Repeat steps 2 through 5 until all tasks are mapped.

Fig. 9. Procedure for using the Max-Max Utility heuristic to generate a resource allocation.

multi-processor (SMP) with one to four processing elements. Examples of the systems using the time-utility function shapes defined in [17] can be found in [27] and [28]. Similar to our model, [17] associates a value function with every process that specifies the value that will be earned by the system if it completes execution of that process at any given moment. Every process has a request time associated with it that gives the time from which the process is available for execution. The request time can be in the future, which makes that process unschedulable at this instant, but the scheduler can consider these times to make current scheduling decisions. This is different from our model, where we assume that the scheduler has no prior knowledge of the arrival time of the tasks. Moreover, in [17] the processes can be either periodic or aperiodic, whereas in our case we only model aperiodic tasks. The authors in [17] use four different shapes for the value functions, including an exponential decay model. They present two new algorithms that make decisions based on value density (value divided by processing time). Their simulations show that these algorithms did better than basic scheduling algorithms that consider either only deadline or only execution times. The authors in [17] consider homogeneous processing elements, while we consider a heterogeneous system. We do not use value density for guiding decisions (as done in [17]) of our heuristics because this metric makes sense only in the homogeneous environment of [17] as opposed to our heterogeneous environment.

In [9], the authors consider the problem of scheduling a set of non-preemptive and mutually independent tasks on a single processor. Associated with each task is a Time Value Function, that gives the contribution each task will provide at its completion time. Each task has a soft and a hard deadline. The contribution of a task stays constant at its highest value up to its soft deadline. Beyond its hard deadline, the contribution of a task is zero. Between the soft and the hard deadline, the contribution value of a task falls from its highest value to zero. The aim of the problem is to sequence the set of tasks to the single processor in an order that maximizes the cumulative contribution from the whole task set. Thus, while [9] maximizes the total utility earned, the environment of a single processor versus our environment of a heterogeneous distributed system makes solution techniques quite different for the two cases.

(1) Make a set composed of the unmapped tasks.
(2) While there are still unmapped tasks:
    (a) For each machine find the set of tasks that have their maximum utility on this machine.
        (i) If the set of tasks is size one, then assign the corresponding task to the machine and remove the task from the set of unmapped tasks.
        (ii) If the size of the set of tasks is greater than one, then assign task with the highest *suff*, and remove that task from the set of unmapped tasks.
    (b) The ready times for all machines are updated.

Fig. 10. Procedure for using the Sufferage heuristic to generate a resource allocation.

In [20], the authors consider a single-processor real-time system. They use analytical methods to create performance features that they then optimize. In their model, they represent the utility that a completed job will earn by using time-utility functions. The time-utility functions have non-zero values between the arrival time and the deadline of the jobs. Unlike our model, where every task can have a different shape for its time-utility function, [20] models a system wherein all jobs have the same shape for their time-utility functions. As mentioned before, solution techniques become significantly different when working with single-processor systems (as considered in [20]) as opposed to working with heterogeneous systems with multiple processors (as considered in our model).

Having fixed time-utility function shapes for various jobs might not be an accurate representation of reality. The authors of [26] point out that every user has their known agenda and as a result, every job that they submit should accurately have a time-utility function tailor-made for their needs. This study considers a homogeneous computing system. They propose a model wherein, when High Performance Computing users submit their jobs, they draw out a time-utility function based on their needs. It is up to the users to decide how accurately they want to model the time-utility functions. Thus, similar to our model, the time-utility functions in [26] usually have time intervals (of varying lengths) and different decay rates within each of these intervals. The authors then propose a genetic algorithm for scheduling with the objective of maximizing the total accrued utility. This genetic algorithm has an average execution time of 8,900 seconds. For the purposes of our work in which scheduling decisions must be made at much smaller intervals (e.g., 30 seconds or less), the direct approach employed in the paper is not suitable. Furthermore, in our study, we assume a heterogeneous computing system, and we incorporate the concept of precedence and priority as scale and shape factors to modify the utility function.

The authors in [39] apply time-utility functions to represent the utility that is earned based on the transmission duration for packets between hosts. Each host can send a packet from a given set of possible packets. Each packet has a time-utility function associated with it, and a utility is earned based on the time duration of the transmission. The goal is to maximize the aggregate accrued utility over the arrival of all packets to their respective destinations. Similar to our model, they consider only uni-modal (non-increasing) time-utility functions. They consider six time-utility functions in their study: step, soft-step, linear, exponential, quadratic, and composite. In the study, the authors define a concept that they call "critical times" that is similar to our definition of time intervals. However, before the first "critical time" the utility is at its maximum, and then it drops to zero at the second time interval. In our study, we consider a High Performance Computing system (not packet transmission). Additionally, we have a system where the completion of some tasks is significantly more important than other tasks. We incorporated this priority and precedence information as scale and shape modifiers of general time utility functions.

In [29], the authors apply time-utility functions to model the utility obtained when a thread successfully completes execution on a QNX Neutrino real-time operating system (with a single processor). In our study, we apply utility functions to a heterogeneous High Performance Computing system with thousands of processors. A "task" in our paper is equivalent to a "thread" in [29]. In contrast to our model, they incorporate preemption of threads. In addition, unlike our model, where the utility of function of a task may decrease over time, they have non-real-time threads whose time-utility function is set to a constant value in order to represent threads that do not have time-constrained activity.

There are also utility functions that vary as a function of the Quality of Service (QoS) associated with the output. For example, a video conferencing task can have multiple utilities associated with the different levels of QoS. The utility gained from executing this task with less quality, e.g., lower bandwidth, would be less than the utility gained from executing it with a higher quality, e.g., higher bandwidth. In [15], the authors explore the

use of utility functions for highly configurable tasks in dynamic real-time systems, i.e., phased-array radars. In this research, the utility a task gains upon completion is not dependent on the completion time. In [15], the utility of a task is based on "QoS dimensions." These dimensions represent an aspect of task quality that is of direct relevance to the user, e.g., a level of audio compression. This type of QoS based utility function is significantly different from our time-utility based functions.

VI. Simulation Trial Setup

A. Generation of Simulation Trials

The target system simulated here has two different types of heterogeneity (partially-consistent and inconsistent), and two different arrival rates (normal and fast). For each of these arrival rates and heterogeneities, 100 different trials are generated.

Each simulation trial has $M = 100$ heterogeneous machines and 10,000 dynamically arriving tasks (from ten different task execution types and four different utility characteristic classes). The ETC matrix, for each trial, will be composed of 100 columns to represent the machines and ten rows for each different task execution type. The procedure for generating the ETC matrices is described in Section VI-B

The priority, precedence, task execution type, and utility characteristic class for the i^{th} task will in general vary from one trial to another, i.e., the properties are assigned independently for each trial. All the simulation trials (both inconsistent and partially-consistent) use the same four utility characteristic classes. The method used to assign arrival times, precedence, and priorities to tasks is described in Section VI-C.

B. Generation of Estimated Time to Compute Matrices

The ETC values are generated using a modified version of the procedure described in [2]. The difference between [2] and how we generate the ETC values is the use of the trimodal and bimodal distributions instead of a single gamma distribution. We use a trimodal distribution (as shown in Figure 11(a)) to generate ETC values for a task that executes significantly faster on special purpose machines, and a bimodal distribution (as shown in Figure 11(b)) for a task that can execute well across multiple machines.

Consider tasks that can make use of some of the special purpose machines. In Figure 11(a), the first mode is a gamma distribution that represents the execution time of a task on special purpose machines. The second mode is a gamma distribution used to represent the execution time of a task on most machines. In the third mode, a pulse is used to represent the execution time of a task on machines that cannot execute it (i.e., a machine not capable of executing that particular task).

A bimodal distribution is used to generate an ETC value for tasks that do not benefit from executing on special purpose machinesThe first mode of the bimodal distribution in Figure 11(b) is used to generate the ETC value of a task on a machine that can execute it (similar to the second mode of Figure 11(a)). While, the second mode is used to represent the execution time of a task on a machine that *cannot* execute it (similar to the third mode of Figure 11(a)). That is, the second mode represents special purpose machines for which these tasks cannot execute (e.g., a C++ program executing on a Java machine).

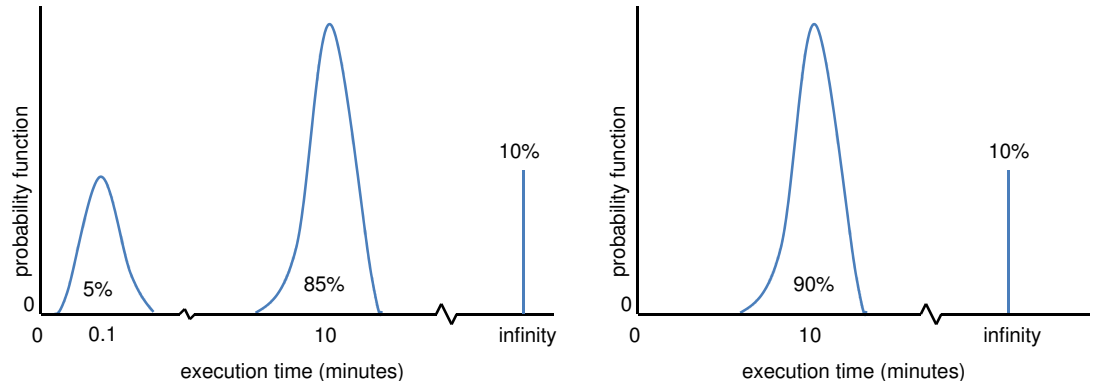
Approximately one-tenth of the task execution types can use special purpose machines that can execute them very quickly. For these task execution types that execute significantly quicker on special purpose machines, the probability of using the first mode in Figure 11(a) is 5%, the probability of the second mode is 85%, and the probability that a machine cannot execute that task is 10%. For the remaining task execution types, the first mode has a probability of 90%, and the second mode has a probability of 10%.

ETC matrices generated for this simulation represent two different types of actual heterogeneity: inconsistent and partially-consistent. An inconsistent matrix is one where if any task t_i has a lower execution time on machine x than machine y, then nothing can be inferred about the relative performance of a task t_j on machine x versus machine y. For a partially-consistent ETC matrix within a consistent sub-matrix, if t_i has a lower execution time on machine x than machine y (both x and y are part of the consistent sub-matrix), then the same is true for any t_k within the sub-matrix. For the partially-consistent matrices simulated here, the consistent sub-matrix was 50% of the task execution types and 50% of the machines. For this study, 100 different ETCs were generated for each type of consistency.

In our study, the following procedure and parameters are used to generate the ETC values. The methodology for generating the value is similar to [2]; however, there are a few differences. For each task execution type using a mean estimated computation time of ten minutes and task coefficient of variation of 0.1, we sample a gamma distribution to obtain a mean for a specific task execution type.

C. Generation of Arrival Rate, Utility Characteristic Class, Precedence, and Priority for each Task

The arriving task has several characteristics that are determined stochastically for the purpose of our simulation. The first characteristic is the arrival time. The



(a) (b)

Fig. 11. The illustrations represent the trimodal and bimodal distributions used to generate the ETC matrices for our simulations. The percentages for the different modes represent the probability of sampling that mode. (a) A trimodal distribution is used to generate the ETC values for the tasks that can run faster on special purpose machines. (b) A bimodal distribution is used to generate the ETC values for the tasks that do not benefit from special purpose machines.

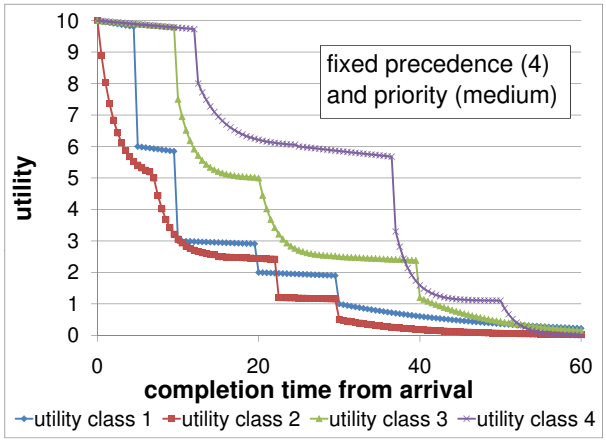


Fig. 12. The four different utility characteristic classes in medium priority and precedence level four (lowest).

TABLE I
THE JOINT PROBABILITY MATRIX SHOWING THE DISTRIBUTION OF TASKS ACROSS THE PAIRS OF POSSIBLE PRECEDENCE LEVELS AND PRIORITY LEVELS.

	precedence			
priority	1	2	3	4
critical	2%	2%	0.05%	0%
high	3.45%	5%	1.5%	3%
medium	0%	10%	10%	10%
low	0%	0%	20%	33%

interarrival times of these tasks follow a Poisson distribution. Therefore, the length of the time interval between the arrival of tasks is an exponential random variable. The mean of this random variable is 5.4 seconds for the normal arrival rate and 4.2 seconds for the fast arrival rate. These arrival rates were selected in conjunction with the mean execution time to ensure the system is oversubscribed.

The next characteristic that needs to be determined is the utility characteristic class. There are *four* different classes in this study, and each class has an equal probability of being selected. The corresponding parameters of a utility characteristic class (time intervals, utility at the beginning of the interval, and decay rate modifier) are representative of workloads anticipated in future computing environments of DOE and DoD interest. Figure 12 shows the utility functions for four different utility characteristic classes with a critical priority and a precedence level four.

The precedence and priority of a task were selected using a matrix that represents joint probabilities of a precedence at a given priority level. An example of this table is shown in Table I. In this figure, for example, a task has a 2% probability of having a "critical" level priority and precedence level of one ($p(priority = \text{“}critical\text{”} \cap precedence = 1) = 2\%$). For this study, the maximum utility values for each priority level are $\pi_1 = 1000$, $\pi_2 = 100$, $\pi_3 = 10$, and $\pi_4 = 1$. The values used for precedence decay are $\beta_1 = 0.6$, $\beta_2 = 0.2$, $\beta_3 = 0.1$, $\beta_4 = 0.01$.

VII. RESULTS

A comparison between the simulation results with an arrival rate of 11.1 tasks per minute (Figure 13) and 14.3 tasks per minute (Figure 14) show the advantage of using batch mode heuristics in a highly oversubscribed environment. The main difference between the results shown in Figures 13 and 14 is that the performance of the immediate mode heuristics degrades much more

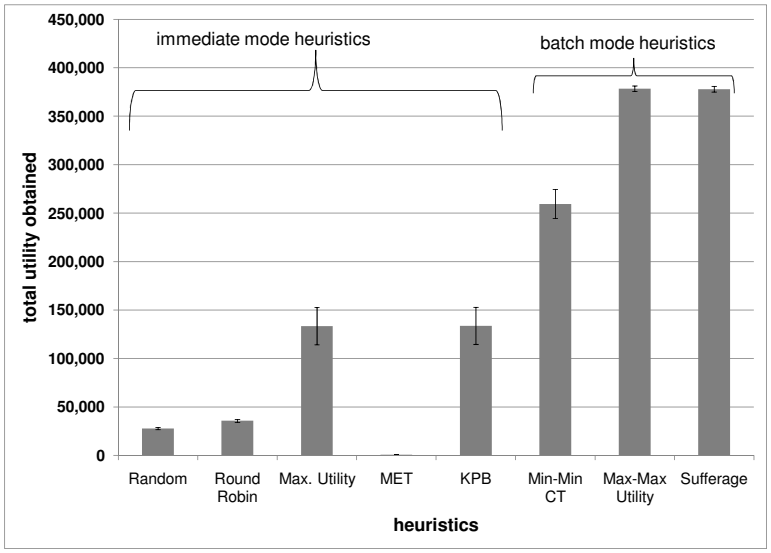


Fig. 13. The averaged results of 100 trials for the inconsistent machine heterogeneity with arrival rate of 11.1 tasks per minute. The error bars represent the 95% confidence intervals.

than the performance of the utility-based batch mode heuristics.

The inconsistent heterogeneity results show that the best performing heuristics are Sufferage and Max-Max Utility. Results from the partially-consistent heterogeneity show very similar relative performance among heuristics compared to the inconsistent heterogeneity results with both arrival rates. In all the results, the Sufferage has overlapping confidence intervals with Max-Max Utility. The Sufferage heuristic runs slower than the other heuristics (see Table II). In general, as shown in Table II, the immediate mode heuristics execute in less than one millisecond, while the batch mode heuristics utility based heuristics can take from 200ms to 500ms. This difference could become more significant if we consider a larger system.

The performance of the best immediate mode heuristics is not as good as the utility-based batch mode heuristics. The best performing immediate mode heuristics are the Maximum Utility and K-percent Best heuristics, which have an average utility of approximately 135,000 units for the results with an arrival rate of 11.1 tasks per minute. The best performing batch heuristics are on average 23% of the maximum possible system utility if the system is not oversubscribed, i.e., the sum of the maximum utility value of all 10,000 tasks. The Minimum Execution Time heuristic performs poorly because there are only 10 task execution types; therefore, the system only has 10 minimum execution time machines and the other $M - 10 = 90$ machines are never used.

The simulation results with an arrival of 14.3 tasks per minute (Figure 14) show how important it is to keep tasks in the virtual queue if the system is very oversubscribed. The performance at 14.3 tasks per minute degrade for all heuristics; however, the performance of utility-based batch mode degrades by approximately 7% while the performance of immediate mode heuristics degrade by approximately 50%. The utility-based batch mode heuristics are better at scheduling critical and high priority tasks when the system is oversubscribed, because they can ignore lower priority tasks in favor of higher priority tasks. Therefore, they are able to outperform the immediate mode heuristics that are forced to allocate all tasks as they arrive.

VIII. Conclusions

The primary goal of this study is to define a general model of task-based time-utility functions and overall system performance. Although our model is based primarily on guidance from ESSC customers, we believe the model is sufficiently flexible to be useful in other environments as well. The second goal is to design and evaluate heuristics that can be used to derive resource allocations that attempt to maximize this system performance measure. The results show that the performance of the the Sufferage and Max-Max heuristics are better than the other heuristics considered in this study. In this simulation environment, heuristic execution times are

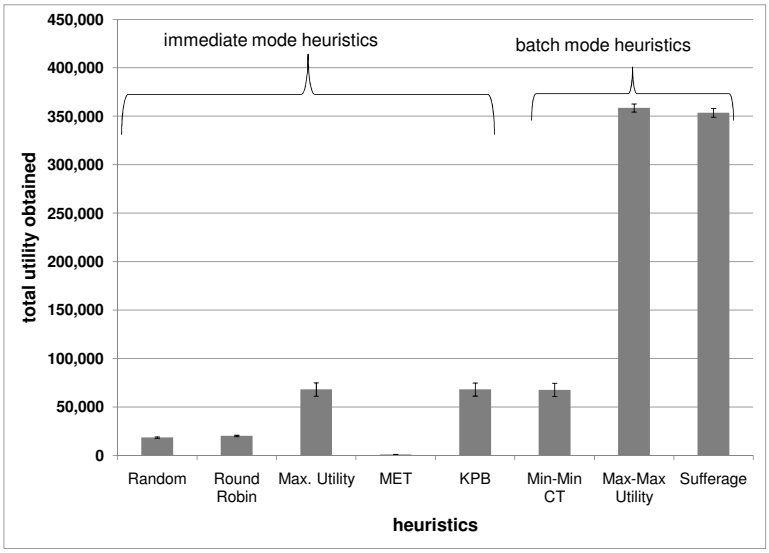


Fig. 14. The averaged results of 100 trials for the inconsistent machine heterogeneity with arrival rate of 14.3 tasks per minute. The error bars represent the 95% confidence intervals.

TABLE II
TABLE SHOWING THE AVERAGE EXECUTION TIME FOR HEURISTICS FOR ALL 10,000 TASKS.

	heuristics							
arrival	Random	Round Robin	Max. Utility	MET	KPB	Min-Min CT	Max-Max Utility	Sufferage
normal	0.46ms	0.56ms	0.67ms	0.74ms	1.34ms	0.1ms	0.3s	0.4s
fast	0.45ms	0.54ms	0.67ms	0.736ms	1.28ms	0.12ms	0.34s	0.5s

significantly higher for the batch mode heuristics than for immediate mode heuristics. We intend to observe how these times change as the system scales.

Possible future work includes using stochastic estimates of execution time, avoiding task starvation by implementing a penalty function, varying the size of the partially consistent sub-matrix, incorporating hard deadlines, considering additional heuristic approaches, incorporating energy consumption in the system model, using wall clock completion times in addition to completion times that are relative to arrival times, allowing parallel and dependent tasks (i.e., tasks modeled as directed acyclic graphs), permitting preemption, and implementing these heuristics in an existing scheduler within the ESSC.

Acknowledgments: *The authors thank Abdulla Al-Qawasmeh and Dalton Young for their valuable comments.*

REFERENCES

[1] I. Al-Azzoni and D. G. Down, "Linear programming based affinity scheduling for heterogeneous computing systems," in *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '07)*, Jun. 2007.

[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, pp. 195–207, Nov. 2000.

[3] H. Barada, S. M. Sait, and N. Baig, "Task matching and scheduling in heterogeneous systems using simulated evolution," in *10th IEEE Heterogeneous Computing Workshop (HCW '01)*, Apr. 2001, pp. 875–882.

[4] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, S. Spring, A. Su, and D. Zagorodnov, "Adaptive computing on the grid using AppLeS," *IEEE Trans. on Parallel and Distributed Systems (TPDS)*, vol. 14, no. 4, pp. 369–382, Apr. 2003.

[5] 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.

[6] L. D. Briceno, M. Oltikar, H. J. Siegel, and A. A. Maciejewski, "Study of an iterative technique to minimize completion times of non-makespan machines," in *16th Heterogeneous Computing Workshop (HCW 2007), in the proceedings of the 21st International Parallel and Distributed Processing Symposium*, Mar. 2007.

[7] L. D. Briceño, H. J. Siegel, A. Maciejewski, M. Oltikar, J. Brateman, J. White, J. Martin, and K. Knapp, "Heuristics for robust resource allocation of satellite weather data processing onto a

heterogeneous parallel system," *IEEE Transactions on Parallel and Distributed Systems*, accepted to appear.

[8] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," in *9th IEEE Heterogeneous Computing Workshop (HCW 2000)*, Mar. 2000, pp. 349–363.

[9] K. Chen and P. Muhlethaler, "A scheduling algorithm for tasks described by time value function," *Journal of Real-Time Systems*, vol. 10, no. 3, pp. 293–312, May 1996.

[10] M. K. Dhodhi, I. Ahmad, and A. Yatama, "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 62, no. 9, pp. 1338–1361, Sep. 2002.

[11] Q. Ding and G. Chen, "A benefit function mapping heuristic for a class of meta-tasks in grid environments," in *CCGRID '01: 1st International Symposium on Cluster Computing and the Grid*, May 2001.

[12] A. Ghafoor and J. Yang, "A distributed heterogeneous supercomputing management system," *IEEE Computer*, vol. 26, no. 6, pp. 78–86, Jun. 1993.

[13] S. Ghanbari and M. R. Meybodi, "On-line mapping algorithms in highly heterogeneous computational grids: A learning automata approach," in *International Conference on Information and Knowledge Technology (IKT '05)*, May 2005.

[14] ——, "Learning automata based algorithms for mapping of a class of independent tasks over highly heterogeneous grids," in *European Grid Conference (EGC '05)*, Feb 2005, pp. 681–690.

[15] S. Ghosh, J. P. Hansen, R. Rajkumar, and J. Lehoczky, "Integrated resource management and scheduling with multi-resource constraints," in *25th Real-Time Systems Symposium*, Dec. 2004.

[16] 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.

[17] E. Jensen, C. Locke, and H. Tokuda, "A time-driven scheduling model for real-time systems," in *IEEE Real-Time Systems Symposium*, Dec. 1985, pp. 112–122.

[18] Z. Jinquan, N. Lina, and J. Changjun, "A heuristic scheduling strategy for independent tasks on grid," in *Eighth International Conference on High-Performance Computing in Asia-Pacific Region 2005*, Nov. 2005.

[19] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42–51, Jul. 1998.

[20] M. Kargahi and A. Movaghar, "Performance optimization based on analytical modeling in a real-time system with constrained time/utility functions," *IEEE Transactions on Computers*, Pre-Print 2011.

[21] K. Kaya, B. Ucar, and C. Aykanat, "Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories," *Journal of Parallel and Distributed Computing*, vol. 67, no. 3, pp. 271–285, Mar. 2007.

[22] A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, vol. 26, no. 6, pp. 18–27, Jun. 1993.

[23] J.-K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S. S. Yellampalli, "Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment," *Journal of Parallel and Distributed Computing*, vol. 67, no. 2, pp. 154–169, Feb. 2007.

[24] J.-K. Kim, H. J. Siegel, A. A. Maciejewski, and R. Eigenmann, "Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling," *IEEE Transactions on Parallel and Distributed Systems, Special Issue on Power-Aware Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1445–1457, Nov 2008.

[25] C. M. Krishna and K. G. Shin, *Real-Time Systems*. McGraw-Hill, 1997.

[26] C. B. Lee and A. E. Snavely, "Precise and realistic utility functions for user-centric performance analysis of schedulers," in *16th International Symposium on High Performance Distributed Computing (HPDC '07)*. New York, NY, USA: ACM, 2007, pp. 107–116.

[27] P. Li, B. Ravindran, H. Cho, and E. D. Jensen, "Scheduling distributable real-time threads in tempus middleware," in *Tenth International Conference on Parallel and Distributed Systems (ICPADS '04)*, 2004, pp. 187–194.

[28] P. Li, B. Ravindran, S. Suhaib, and S. Feizabadi, "A formally verified application-level framework for real-time scheduling on posix real-time operating systems," *IEEE Transactions on Software Engineering*, vol. 30, no. 9, pp. 613–629, Sep. 2004.

[29] P. Li, H. Wu, B. Ravindran, and E. D. Jensen, "A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints," *IEEE Transactions on Computers*, vol. 55, no. 4, pp. 454–469, Apr. 2006.

[30] 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.

[31] 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," *Journal of Supercomputing, Special Issue on Grid Technology*, vol. 42, no. 1, pp. 33–58, Jan. 2007.

[32] D. Paranhos, W. Cirne, and F. Brasileiro, "Trading cycles for information: Using replication to schedule bag-of-tasks applications on computational grids," in *International Conference on Parallel and Distributed Computing*, Aug. 2003.

[33] B. Ravindran, E. D. Jensen, and P. Li, "On recent advances in time/utility function real-time scheduling and resource management," in *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2005)*, May 2005, pp. 55–60.

[34] P. SaiRanga and S. Baskiyar, "A low complexity algorithm for dynamic scheduling of independent tasks onto heterogeneous computing systems," in *43rd annual Southeast Regional Conference - Volume 1*, Mar. 2005, pp. 63–68.

[35] V. Shestak, J. Smith, H. J. Siegel, and A. A. Maciejewski, "Stochastic robustness metric and its use for static resource allocations," *Journal of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.

[36] S. Shivle, H. J. Siegel, A. A. Maciejewski, P. Sugavanam, T. Banka, R. Castain, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, and J. Velazco, "Static allocation of resources to communicating subtasks in a heterogeneous ad hoc grid environment," *Journal of Parallel and Distributed Computing, Special Issue on Algorithms for Wireless and Ad-hoc Networks*, vol. 66, no. 4, pp. 600–611, Apr. 2006.

[37] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," in *5th IEEE Heterogeneous Computing Workshop (HCW 1996)*, Apr. 1996, pp. 86–97.

[38] P. Sugavanam, H. J. Siegel, A. A. Maciejewski, M. Oltikar, A. Mehta, R. Pichel, A. Horiuchi, V. Shestak, M. Al-Otaibi, Y. Krishnamurthy, S. Ali, J. Zhang, M. Aydin, P. Lee, K. Guru, M. Raskey, and A. Pippin, "Robust static allocation of resources for independent tasks under makespan and dollar cost constraints," *Journal of Parallel and Distributed Computing*, vol. 67, no. 4, pp. 400–416, Apr. 2007.

[39] J. Wang and B. Ravindran, "Time-utility function-driven switched ethernet: Packet scheduling algorithm, implementation, and feasibility analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 2, pp. 119–133, Feb. 2004.

[40] B. Wei, G. Fedak, and F. Cappello, "Scheduling independent tasks sharing large data distributed with bittorrent," in *The 6th IEEE/ACM International Workshop on Grid Computing*, Nov. 2005.

[41] M. Wu and W. Shu, "Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems,"

in *9th Heterogeneous Computing Workshop (HCW '00)*, Mar. 2000, pp. 375–385.

[42] 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.

BIOGRAPHIES

Luis D. Briceño received his Ph.D. degree in Electrical and Computer Engineering from Colorado State University, and his B.S. degree in Electrical and Electronic Engineering from the University of Costa Rica. He is currently a post doctoral scholar at Colorado State University. His research interests include heterogeneous parallel and distributed computing.

Bhavesh Khemka is a Ph.D. student and research assistant in the Electrical and Computer Engineering Department at Colorado State University. He received his B.E. degree in Electrical and Electronics Engineering from Hindustan College of Engineering affiliated with Anna University, India. His research interests include robust resource allocations in heterogeneous and distributed computing environments.

H. J. Siegel 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. Prof. Siegel is a Fellow of the IEEE and a Fellow of the ACM. 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 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. For more information, please see www.engr.colostate.edu/∼hj.

Anthony A. Maciejewski received the B.S., M.S., and Ph.D. degrees in Electrical Engineering in 1982, 1984,and 1987, respectively, all from The Ohio State University. 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 IEEE. A complete vita is available at www.engr.colostate.edu/∼aam.

Christopher Groër is a research staff scientist in the Computational Mathematics Group at Oak Ridge National Lab. His research interests include large scale combinatorial optimization problems, job scheduling, and High Performance Computing. He worked as an applied research mathematician for the Department of Defense from 2002 to 2008 and received a B.A. from Vanderbilt (math and economics), M.A. from University of Georgia (mathematics), and a Ph.D. from the University of Maryland (applied math and scientific computation).

Gregory Koenig is an R & D Associate at Oak Ridge National Laboratory where his work involves developing scalable system software and parallel tools for ultrascale-class parallel computers. His research interests include middleware for grid, cloud, and on-demand/utility computing incorporating technologies such as virtualization, fault detection and avoidance, and resource scheduling. He holds a Ph.D. (2007) and M.S. (2003) in Computer Science from the University of Illinois at Urbana-Champaign as well as three B.S. degrees (Mathematics, 1996; Electrical Engineering Technology, 1995; Computer Science, 1993) from Indiana University-Purdue University Fort Wayne.

Gene Okonski is a Systems Architect for the Department of Defense. His areas of focus are on distributed computing, data processing architectures, and information technology efficiency. He has been involved as a technologist and leader of a number of large scale development activities spanning some twenty years. He received his B.S. Degree in Electrical Engineering with a minor in Economics from Colorado State University (1988). He received an M.S. in Engineering from Johns Hopkins University School of Engineering with a concentration in Systems Engineering (1993).

Steve Poole currently divides his time between duties at the Department of Defense and Oak Ridge National Laboratory. He is the Chief Scientist and Director of Special Programs. He can be reached at spoole@ornl.gov.