2014 IEEE 28th International Parallel & Distributed Processing Symposium Workshops

Utility Driven Dynamic Resource Management in an Oversubscribed Energy-Constrained Heterogeneous System

Bhavesh Khemka*, Ryan Friese*, Sudeep Pasricha*†, Anthony A. Maciejewski*, Howard Jay Siegel*†, Gregory A. Koenig‡, Sarah Powers‡, Marcia Hilton§, Rajendra Rambharos§, and Steve Poole‡§

*Department of Electrical and Computer Engineering
†Department of Computer Science
Colorado State University
Fort Collins, CO 80523

‡Oak Ridge National Laboratory
Oak Ridge, TN 37831
§Department of Defense
Washington, DC 20001

Email: {Bhavesh.Khemka, Ryan.Friese, Sudeep, AAM, HJ}@colostate.edu, {Koenig, PowersSS}@ornl.gov, mmskizig@verizon.net, Jendra.Rambharos@gmail.com, SPoole@ornl.gov

***Abstract*—In this paper, we address the problem of scheduling dynamically-arriving tasks to machines in an oversubscribed heterogeneous computing environment. Each task has a monotonically decreasing utility function associated with it that represents the utility (or value) based on the task's completion time. Our system model is designed based on the environments of interest to the Extreme Scale Systems Center at Oak Ridge National Laboratory. The goal of our scheduler is to maximize the total utility earned from task completions while satisfying an energy constraint. We design an energy-aware heuristic and compare its performance to heuristics from the literature. We also design an energy filtering technique for this environment that is used in conjunction with the heuristics. The filtering technique adapts to the energy remaining in the system and estimates a fair-share of energy that a task's execution can consume. The filtering technique improves the performance of all the heuristics and distributes the consumption of energy throughout the day. Based on our analysis, we recommend the level of filtering to maximize the performance of scheduling techniques in an oversubscribed environment.**

***Index Terms*—scheduling; energy-constrained; utility functions; energy filtering;**

I. Introduction

During the past decade, large-scale computing systems have become increasingly powerful. As a result, there is a growing concern with the amount of energy needed to operate these systems [1], [2]. From 2005 to 2010 the electricity consumption of high performance computing (HPC) systems has increased by 56% worldwide [3]. In addition to the rising costs of increased electricity use, some data centers are now unable to increase their computing performance due to physical limitations on the availability of energy. A survey conducted in 2008 showed that 31% of the data centers identified energy availability as a key factor limiting new server deployment [4]. Another example to emphasize this point: Morgan Stanley, a global financial services firm based in New York, is physically unable to draw the energy needed to run a new data center in Manhattan [5]. If examples such as these become commonplace, many HPC systems could be forced to execute workloads with severe constraints on the amount of energy available to be consumed.

The need for additional performance among HPC systems combined with higher energy consumption and costs are making it increasingly important for system administrators to adopt energy-efficient workload execution policies. In an energy-constrained environment, it is desirable for such policies to maximize the performance of the system. This research investigates the design of energy-aware scheduling techniques with the goal of maximizing the performance of a workload executing on an energy-constrained HPC system.

Specifically, we model a compute facility and workload being investigated by the Extreme Scale Systems Center (ESSC) at Oak Ridge National Laboratory (ORNL). The ESSC is a joint venture between the United States Department of Defense (DoD) and Department of Energy (DOE) to perform research and deliver tools, software, and technologies that can be integrated, deployed, and used in both DoD and DOE environments. Our goal is to study techniques that allow system administrators to maximize the performance of their computing systems while staying within a specified energy constraint using intelligent resource allocations (i.e., assignment of tasks to machines). Each task has a monotonically decreasing utility function associated with it that represents the utility (or value) based on the task's completion time. System performance is measured in terms of total utility earned, which is the sum of all utility earned by tasks that complete within the energy-constraint [6], [7]. The computing environment we model, based on the expectations of future DoD and DOE environments, incorporates heterogeneous resources that utilize a mix of different machines to execute workloads with diverse computational requirements.

In a heterogeneous environment, tasks typically have dif-

978-1-4799-4116-2/14 $31.00 © 2014 IEEE
DOI 10.1109/IPDPSW.2014.12

ferent execution time and energy consumption characteristics when executed on different machines. We model our machines to have three different performance states (P-states) in which tasks can execute. By employing different resource allocation strategies, it is possible to manipulate the performance and energy consumption of the system to align with the goals set by the system administrator. We develop a novel energy-aware resource allocation policy that has the goal of maximizing the utility earned while obeying an energy constraint. We compare our policy with three techniques from the literature designed to maximize utility [6], [7], and show that for energy-constrained environments, heuristics that manage their energy usage throughout the day outperform heuristics that only try to maximize utility. We enhance the resource allocation policies by designing an energy filter (based on a simpler method [8]) for our environment. The goal of the filtering technique is to remove high energy consuming allocation choices that consume more energy than an estimated fair-share. This distributes the budgeted energy across the whole day. We perform an in-depth analysis to demonstrate the benefits of our energy filter.

In summary, we make the following contributions: (a) the design of a new resource management technique that maximizes utility earned within an energy constraint for an oversubscribed heterogeneous computing system based on environments of interest to the DoD and DOE, (b) the design of an energy filtering mechanism that is adaptive to the energy remaining in the system, enforces "fairness" in energy consumed by tasks and distributes the energy budgeted for the day throughout the day, (c) show how heuristics that only maximize performance can become energy-aware by adapting three previous techniques to use an energy filter, (d) a sensitivity analysis of energy filtering for all four heuristics and (e) a recommendation on how to select the best level of filtering for a heuristic, based on a detailed analysis of the performance of our heuristics.

The remainder of this paper is organized as follows. The next section formally describes the problem we address and the system model. Section III describes our resource management techniques. We then provide a sample of related work in Section IV. Our simulation and experimental setup are detailed in Section V. In Section VI, we discuss and analyze the results of our experiments. We finish with our conclusion in Section VII.

II. Problem Description

A. System Model

In this study, we assume a workload where tasks arrive dynamically throughout the day and the scheduler maps the tasks to machines for execution. We model our workload and computing system based on the needs of the ESSC. Each task in the system has a utility function associated with it (as described in [6]). Utility functions are monotonically-decreasing functions that represent the utility (or value) of completing the task at different times. We assume the utility functions are given by users or system administrators for any task.

Tasks are assumed to be independent (they do not require inter-task communication) and can execute concurrently (each on a single machine, possibly with parallel threads). This is typical of many environments such as [36]. We do not allow the preemption of tasks, i.e, once a task starts execution, it must execute until completion.

Our computing system environment consists of a suite of heterogeneous machines, where each machine belongs to a specific machine type (rather than a large monolithic system, such as Titan). Machines belonging to different machine types may differ in their microarchitectures, memory modules, and/or other system components. We model the machines to contain CPUs with dynamic voltage and frequency scaling (DVFS) enabled to utilize three different performance states (P-states) that offer a trade-off between execution time and power consumption. We group tasks with similar execution characteristics into task types. Tasks belonging to different task types may differ in characteristics such as computational intensity, memory intensity, I/O intensity and memory access pattern. The type of a task is not related to the utility function of the task. We model heterogeneity in our computing system and workload, i.e., machine type A may be faster (or more energy-efficient) than machine type B for certain task types but slower (or less energy-efficient) for others.

For a task of type i on a machine of type j running in P-state k, we assume that we are given the Estimated Time to Compute ($ETC(i,j,k)$) and the Average Power Consumption ($APC(i,j,k)$). It is common in the resource management literature to assume the availability of this information based on historical data or experiments [9]–[15]. This assumption is also valid for our ESSC environment. For the simulation study conducted in this paper, we synthetically create our ETC and APC matrices. We can compute the Estimated Energy Consumption ($EEC(i,j,k)$) by taking the product of its execution time and average power consumption, i.e., $EEC(i,j,k) = ETC(i,j,k) \times APC(i,j,k)$. We model general-purpose machine types and special-purpose machine types [16]. The special-purpose machine types execute certain special-purpose task types much faster than the general-purpose machine types, although they may be incapable of executing the other task types.

B. Problem Statement

Recall that we consider a workload model where tasks arrive dynamically throughout the day. However, the scheduler does not have prior knowledge of the arrival pattern of the tasks, the utility functions of the task, nor their task type. The goal of the scheduler is to maximize the total utility that can be earned from completing tasks while satisfying an annual energy constraint. To help meet the annual energy constraint, we constrain the energy consumption of the computing system for each day of the year. We can calculate an appropriately scaled value for a given day's energy constraint ($energy\ constraint_{day}$) by taking the ratio of the total energy remaining for the year and the number of days remaining in the year. This reduces the problem to that of maximizing the total utility earned per

day while obeying $energy\ constraint_{day}$. We perform our simulations to gather the results for the duration of a day to keep the simulation time tractable, but we could use any interval of time (e.g., two hours, six months, a year). If a task starts execution on one day and completes execution on the next, the utility earned and the energy consumed for each day is prorated based on the proportion of the tasks execution time in each day. This is done so that each day gets the utility for the executions that consumed its energy to permit a fair comparison of different heuristic approaches. For ESSC, constraints on power (energy per second) are not a concern.

III. Resource Management

A. Overview

It is common to use heuristics for solving the task to machine resource allocation problem as it has been shown, in general, to be NP-complete [17]. There are different types of dynamic heuristics (also known as on-line heuristics [18]) that can be used to quickly assign incoming tasks to machines. Mapping events occur any time a scheduling decision has to be made. We use batch-mode heuristics that trigger mapping events after fixed time intervals as they performed best in our previous work [6]. To account for oversubscription, we use a technique that drops tasks with low potential utility at the current time. We also design an energy filter that helps guarantee the energy constraint by avoiding allocating tasks that use more than their "fair-share" of energy. Our simulation results show the benefit of this technique.

Mapping events for our batch-mode heuristics are triggered every minute. If the execution of the previous mapping event takes longer than a minute then the next mapping event is triggered after the previous one completes execution. The task that is next-in-line for execution on a machine is referred to as the pending task. All other tasks that are queued for the machines are said to be in the virtual queues of the scheduler. Figure 1 shows an example system with four machines, the executing tasks, the tasks in the pending slot, and the virtual queues of the scheduler. At a mapping event, the batch-mode heuristics make scheduling decisions for a batch of tasks that are composed of the tasks that have arrived since the last mapping event and the tasks that are currently in the virtual queues. This set of tasks is called the mappable tasks. The batch-mode heuristics are not allowed to remap the pending tasks so that the machines do not idle if the currently executing tasks complete while the heuristic is executing. In this study, we adapt three batch-mode heuristics (from our previous work [6], [7]) to the new environment, design a new energy-aware batch-mode heuristic, and analyze and compare their performances.

We design the batch-mode heuristics such that they are not allowed to schedule a task that will violate the day's energy constraint. The batch-mode heuristics also are not allowed to map a task to a machine on which it will start execution on the next day. This is because the next day may have different values for the energy constraint (if the current day consumes less energy than budgeted for the day) and therefore the scheduling decision for this task may be different. If the task will violate the current day's energy constraint or if no machine can start the execution of the task within the current day, then the task's consideration is postponed to the next day. At the start of the next day, all postponed tasks are added to the batch and the heuristics make mapping decisions for these tasks as well.

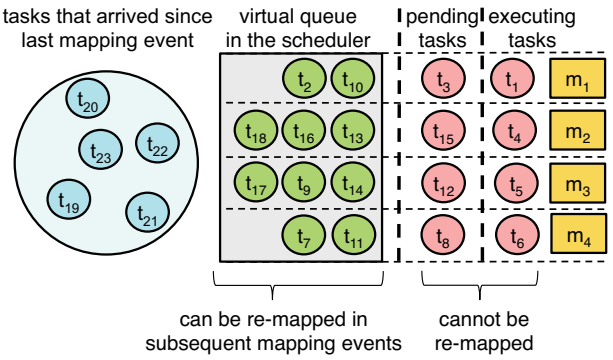


Fig. 1. An example system of four machines showing tasks that are currently executing, waiting in the pending slot, waiting in the virtual queue, and have arrived since the last mapping event (and are currently unmapped).

B. Batch-mode Heuristics

We present a new heuristic that tries to minimize energy while maximizing the utility earned. The Max-Max Utility-Per-Energy Consumption (Max-Max UPE) is a two-stage heuristic based on the concept of the two-stage Min-Min heuristic that has been widely used in the task scheduling literature [19]–[29]. In the first stage, the heuristic independently finds for each task in the batch, the machine and P-state that maximizes "utility earned / energy consumption." If none of the machine-P-state choices for a task satisfy the day's energy constraint nor start the execution of the task within the current day, then, the task is postponed to the next day and removed from the batch. In the second stage, the heuristic picks the task-machine-P-state choice from the first stage that provides the overall highest "utility earned / energy consumption." The heuristic assigns the task to that machine, removes that task from the set of mappable tasks, and repeats this process iteratively until all tasks are either mapped or postponed.

For comparison, we analyze the following three utility maximization heuristics to examine how heuristics that do not consider energy perform in an energy-constrained environment. These heuristics assign tasks until there is no remaining energy in the day.

The Min-Min Completion Time (Min-Min Comp) heuristic is a fast heuristic adapted from [6], [7] and is a two-stage heuristic like the Max-Max Utility-Per-Energy heuristic. In the first stage, this heuristic finds for each task the machine and P-state choice that completes execution of the task the soonest. This also will be the machine-P-state choice that earns the highest utility for this task (because we use monotonically-

decreasing utility functions). In the second stage, the heuristic picks the task-machine-P-state choice from the first stage that provides the earliest completion time. This batch-mode heuristic is computationally efficient because it does not explicitly perform any utility calculations.

The Max-Max Utility (Max-Max Util) heuristic introduced in [6], [7] is also a two-stage heuristic like the Min-Min Comp heuristic. The difference is that in each stage Max-Max Util maximizes utility, as opposed to minimizing completion time. In the first stage, this heuristic finds task-machine-P-state choices that are identical to those found in the first stage of the Min-Min Comp heuristic. In the second stage, the decisions made by Max-Max Util may differ from those of Min-Min Comp. This is because picking the maximum utility choice among the different task-machine-P-state pairs depends both on the completion time and the task's specific utility function.

The Max-Max Utility-Per-Time (Max-Max UPT) heuristic introduced in [7] is similar to the Max-Max Util heuristic, but in each stage it maximizes "utility earned / execution time," as opposed to maximizing utility. This heuristic selects assignments that earn the most utility per unit time, which can be beneficial in an oversubscribed system.

C. Dropping Low Utility Earning Tasks

We use the technique to drop tasks with low potential utility at the current time (introduced in our previous work [7]). Dropping a task means that it will never be mapped to a machine. The dropping operation helps the batch-mode heuristics tolerate high oversubscription. Due to the oversubscribed environment, if a resource allocation heuristic tried to have all tasks execute, most of the task completion times would be so long that the utility of most tasks would be very small. This would negatively impact users as well as the overall system performance. Given the performance measure is the total utility achieved by summing the utilities of the completed tasks, dropping tasks leads to higher system performance, as well as more satisfied users.

The dropping operation is performed before a heuristic is called for making its scheduling decisions so the heuristic is not required to make as many scheduling decisions. When a mapping event is triggered, we determine the maximum possible utility that each mappable task could earn on any machine assuming it can start executing immediately after the pending task is finished. If this utility is less than a dropping threshold (determined empirically), we drop this task from the set of mappable tasks. If the utility earned is not less than the threshold, the task remains in the mappable tasks set and is included in the batch-mode heuristic's allocation decisions.

Because of the high oversubscription in our environment, the number of tasks in the batch increases quickly. This can cause the heuristic execution time to be long enough to delay the trigger of subsequent mapping events. This results in poor performance because it now takes longer for the heuristics to service any high utility earning task that may have arrived. By the time the next mapping event triggers, the utility from this task may decay substantially. By dropping tasks with low potential utility at the current time, we reduce the size of the batch and enable the heuristics to complete their execution within the mapping interval time (a minute). This allows the heuristics to move quickly any high utility earning task up to the front of the virtual queue to complete its execution soon.

If a batch-mode heuristic postpones a task to the next day, a check is performed to make sure that the maximum possible utility that the task could earn (at the start of the next day) is greater than the dropping threshold. If it is not, the task is dropped and it is not postponed.

D. Energy Filtering

The goal of our new energy filter technique we have designed is to remove potential allocation choices (task-machine-P-state combination) from a heuristic's consideration if the choice of allocation consumes more energy than an estimated fair-share energy budget. We call this budget the $task\ budget$. The value of the $task\ budget$ needs to adapt to the energy being consumed by the system and should account for the energy remaining in the day and the time of the day. Therefore, the value of the $task\ budget$ is recomputed at the start of every mapping event.

We denote $energy\ consumed$ as the total energy that has been consumed by the system in the current day, and $energy\ scheduled$ as the energy that will be consumed by tasks queued for execution. At the start of a mapping event, the virtual queued tasks are removed from the machine queues and inserted into the mappable tasks set. Therefore, $energy\ scheduled$ will account for the energy that will be consumed by all tasks that are currently executing and the tasks that are in the pending slot. The total energy that can be scheduled by heuristics (without violating the day's energy constraint) is denoted by $energy\ remaining$. It is computed using Equation 1.

$$\begin{aligned} energy\ remaining = {} & energy\ constraint_{day} \\ & - energy\ consumed \\ & - energy\ scheduled \end{aligned} \tag{1}$$

The ready time of a machine is set to either the completion time of the last queued task for the machine or the current time, whichever is greater. At the start of the mapping event, the last queued task on a machine will be the pending task. The total time remaining for computations (summed across machines) in the day is denoted as the $aggregate\ time\ remaining$. We compute it by summing across machines the difference between the end time of the day and the ready time of the machine. Figure 2 shows its computation for an example system with three machines. As shown, even though machine m3 is not executing anything after time 16, the available compute time from that machine is obtained by taking the difference between end of the day and the current time.

The average of the execution time values of all tasks, machines, and P-states is represented as $average\ exec\ time$. The energy filtering technique needs to estimate the total number

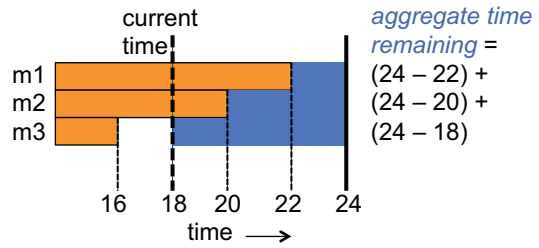


Fig. 2. An example system of three machines showing the computation of *aggregate time remaining*. It represents the total computation time available from the current time till the end of the day.

of tasks that can be executed in the remaining part of the day. It does this by taking the ratio of *aggregate time remaining* and *average exec time*. The energy filter has to use *average exec time* because the scheduler is unaware of the type of tasks that may arrive or which machine or P-state they will be assigned to for the rest of the day.

To adjust the value of the *task budget* around its estimate, we use a multiplier called *energy leniency*. Higher values of the *energy leniency* factor imply more leeway for high energy allocation choices to pass through the filter, whereas a low value for the *energy leniency* would filter many more choices. This value is determined empirically.

The *task budget* that is used to compare the energy consumption of potential allocation choices against is computed using Equation 2.

$$task\ budget = energy\ leniency \times \frac{energy\ remaining}{\left(\frac{aggregate\ time\ remaining}{average\ exec\ time}\right)} \quad (2)$$

This *task budget* is recomputed at the start of each mapping event and is an estimate of the amount of fair-share energy that we want an execution of a task to consume. At each mapping event, the heuristics consider only those task-machine-P-state allocation choices that consume less energy than the *task budget*.

IV. Related Work

Heterogeneous task scheduling in an energy-constrained computing environment is examined in [26]. The authors model an environment where devices in an ad-hoc wireless network are limited by battery capacity. This differs significantly for our environment where we model a larger and more complex heterogeneous system with a utility based performance metric. Additionally, in our study, the energy available for use under the constraint is shared across all resources, while in [26] each resource contains its own energy constraint.

An energy-constrained task scheduling problem in a wireless sensor network environment is studied in [30]. The authors analyze how the presence of an energy constraint affects the schedule length when executing a set of dependent tasks. A wireless sensor network is significantly different from the environment we are modeling. In our model, each task contributes a certain amount of utility to the system. We are not concerned with minimizing a schedule length, as tasks continuously arrive through out the day.

In [31], a set of dynamically arriving tasks with individual deadlines are allocated to machines within a cluster environment with the goal of conserving energy. Specifically, the authors try to optimize the energy consumption while meeting the constraint of completing all tasks by their deadlines. Our environment tries to maximize the total utility earned while operating under an energy constraint. Additionally, [31] uses constant arrival patterns in an undersubscribed system, while our work focuses on highly oversubscribed environments where tasks arrive in varying sinusoidal or bursty patterns.

A dynamic resource allocation problem in a heterogeneous energy-constrained environment is studied in [8]. Tasks within this system contain individual deadlines, and the goal is to complete as many tasks by their individual deadlines as possible within an energy constraint. This is a different problem from our work as we are trying to maximize the utility earned not the number of tasks that meet their hard deadlines. The authors of [8] use heuristics that map each task to machine as soon as the task arrives with no remapping, whereas in our environment we map batches of tasks at a time, allowing us to use more information when making allocation decisions and can remap tasks in the virtual queue. This allows heuristics to move high utility earning tasks that have recently arrived in the system to the front of the queues in order to earn more utility.

In [32] the authors formulate a bi-objective resource allocation problem to analyze the trade-offs between makespan and energy consumption. Their approaches use makespan as a measure of system performance as opposed to utility as we do in our work. Additionally, they model static environments where the workload is a bag-of-tasks, unlike our work that considers a system where tasks arrive dynamically. In our work, we consider maximizing the utility earned while meeting an energy constraint whereas [32] does not consider an energy constraint in its resource allocation decisions.

V. Simulation Setup

A. Overview

We simulate the arrival and mapping of tasks for a span of two days with the first day used to bring the system up to steady-state operation. We collect our results (e.g., total utility earned, energy consumed) only for the second day to avoid the scenario where the machines start with empty queues. We average the results of our experiments across 48 simulation trials. Each of the trials represents a new workload of tasks (with different utility functions, task types, and arrival times), and a different compute environment by using new values for the entries in the ETC and APC matrices (but without changing the number of machines). All of the parameters used in our simulations are set to closely match the expectations for future environments of interest to the ESSC.

B. Workload Generation

The utility functions for all the tasks in a workload are given, and can start at one of four maximum utility values: 8, 4, 2, or 1. These values are based on the needs of the ESSC, but for other environments, different values and greater or fewer number of values may be used. A method for generating utility functions can be found in [6], [33].

We generate the arrival patterns to closely match expected workloads that are of interest to ESSC [34]. In this environment, general-purpose tasks arrive in a sinusoidal pattern and special-purpose tasks follow a bursty arrival pattern.

C. Execution Time and Power Modeling

In our simulation environment, approximately 50,000 tasks arrive during the duration of a day and each of them belong to one of 100 task types. The compute system that we model has 13 machine types consisting of a total of 100 machines. Among these 13 machine types, four are special-purpose machine types while the remaining are general-purpose machine types. The special-purpose machines run a subset of task types that are special with respect to them 10 times faster than the general-purpose machines. The special-purpose machines do not have the ability to run tasks of other task types. In our environment, three to five task types were special for each special-purpose machine type.

We assume that all machines have three major P-states in which they can operate. We use techniques from the Coefficient of Variation (COV) method [35] to generate the entries of the ETC and APC matrices in the highest power P-state. The mean value of execution time on the general-purpose and the special-purpose machine types is set to ten minutes and one minute, respectively. To generate the dynamic power values for the intermediate P-state and the lowest power P-state, we scale the dynamic power to 75% and 50%, respectively, of the highest power P-state. The execution time for these P-states are also generated by scaling the execution time of the highest power P-state. To determine the factor by which we will scale the execution time of the highest power P-state for the intermediate and lowest power P-states, we sample a gamma distribution with a mean value that is approximately $1/\sqrt{(\%\ scaled\ in\ power)}$. For example, the lowest power P-state's execution time will be scaled by a value sampled from a gamma distribution that has a mean approximately equal to $1/\sqrt{0.5}$. The execution time of any task is guaranteed to be the shortest in the highest power P-state, but the most energy-efficient P-state can vary across tasks. These are done to model reality where the impact on execution time and energy consumption by switching P-states depends on the CPU-intensity/memory-intensity of the task, overhead power of the system, etc.

D. Obtaining an Energy Constraint

In real-world scenarios, an annual energy budget is given for an HPC system. As mentioned in Section II-B, we can estimate the energy constraint of the current day using a given annual energy constraint to help ensure each day uses an equal portion of the remaining energy from the annual budget.

For simulation purposes, we need to create an energy constraint that we can use to analyze our resource management techniques. We first run Max-Max UPT (the heuristic that provides the best utility earned from our previous work [7]) for a full 24-hour time period, disregarding the energy constraint. Based on these results, we average the total energy consumption throughout the day across 48 simulation trials and use 70% of this average value as our energy constraint. We obtain the simulated annual energy constraint by multiplying this value by the number of days in a year. For our simulations, we used a value of 405.84 GJ for the year, which averages out to 1.11 GJ per day.

VI. Results

A. Overview

All results shown in this section display the average over 48 simulation trials with 95% confidence interval error bars (the simulator uses two 24 core nodes on the Colorado State University ISTeC Cray [36]). We first discuss the performance of the four heuristics in the energy-constrained environment when not using the energy filtering technique. For comparison purposes, we implement a Random heuristic that randomly maps tasks to machines. All the heuristics (including Random) used a dropping threshold of 0.5 units of utility to tolerate the oversubscription. We use a dropping threshold of 0.5 units of utility as it gave the best performance in our previous work [7]. When selecting a dropping threshold, we must consider the level of oversubscription of the environment in addition to the utility values of tasks. We then examine the effect of the filtering mechanism on the heuristics with a sensitivity study and an analysis of the performance. Finally, we compare all heuristics when using the best filtering case for each of the heuristics and suggest how one can tune the level of filtering to obtain the best performance for any heuristic.

B. Results without Energy Filtering

Figure 3 shows the total utility earned by the four heuristics that we examined with and without filtering and compare their performance with that of the Random heuristic. We first discuss the performance of the heuristics without the energy filtering. Random performed the worst because it does not consider the completion time, utility earned, or energy consumed when making assignment decisions. Min-Min Comp performed the worst among the batch-mode heuristics because it does not explicitly account for utility. Both Max-Max Util and Max-Max UPT perform better than Min-Min Comp as they try to maximize the utility earned from each allocation, but Max-Max UPT performs better than Max-Max Util as it is able to to account for both utility and the execution time of the tasks on machines, which is helpful in an oversubscribed system like the one we consider. Max-Max UPE earns the highest utility even though it consumes the same amount of energy as the others, i.e., approaches the energy constraint. This is because the heuristic accounts for energy consumption

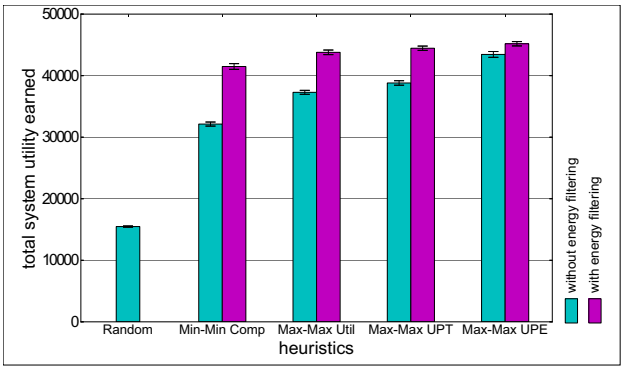


Fig. 3. Total utility earned by the heuristics in the no-filtering case and the filtering case with their best *energy leniency* case. These are compared to the total utility earned by a Random assignment. Results averaged over 48 trials with 95% confidence intervals.

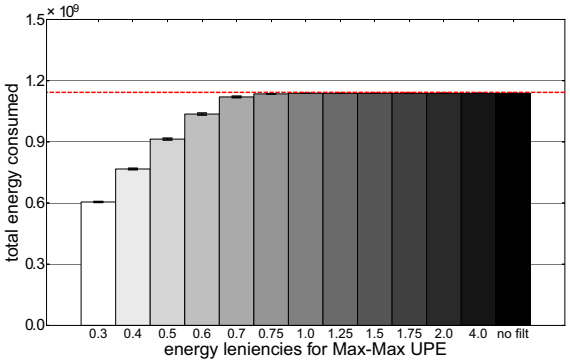


Fig. 4. Sensitivity tests showing the total energy consumed as the *energy leniency* is varied for the Max-Max UPE heuristic. The energy constraint is shown by the dashed line. Results averaged over 48 trials with 95% confidence intervals.

while trying to maximize utility, and thus is able to avoid high energy-consuming allocation choices without significantly affecting the utility earned. All the heuristics hit the energy constraint for the day and were not allowed to map any more tasks that may violate the constraint. These results indicate that for energy-constrained environments it is best to use heuristics that consider energy, rather than heuristics that try to solely optimize for performance. Without energy filtering, Max-Max UPE is approximately 11% better than Max-Max UPT (the best performing utility maximization heuristic).

C. Results with Energy Filtering

We now examine the effect of the energy filtering mechanism on the batch-mode heuristics. The extent to which energy filtering is performed is controlled by the *energy leniency* term (see Equation 2). A higher value for the *energy leniency* would result in a higher value of the *task budget* and would therefore let more allocation choices pass through the filter. Alternatively, a lower value of *energy leniency* would let fewer allocations pass through the filter. Not using filtering implies an *energy leniency* value of infinity. We performed a sensitivity test for all the heuristics by varying the value of *energy leniency* from 0.3 to 4.0, and compared the performance with the no-filtering case.

Figures 5a and 5b show the effect of varying the value of *energy leniency* on the total utility earned by the Max-Max UPT and the Max-Max UPE heuristics, respectively. Figure 4 shows the energy consumption of the Max-Max UPE heuristic as the *energy leniency* value is varied. Sensitivity tests of the utility earned for the Min-Min Comp and Max-Max Util heuristics show trends similar to that of the Max-Max UPT heuristic. While the energy consumed for Min-Min Comp, Max-Max Util, and Max-Max UPT are similar to Max-Max UPE.

In general, for all the heuristics, as we increase the value of *energy leniency* from 0.3, the utility earned increases and then decreases as we approach the no-filtering case. All heuristics benefit from the filtering operation. The best-performing case for Min-Min Comp, Max-Max Util, and Max-Max UPT occurs at an *energy leniency* of 0.75, whereas the Max-Max UPE heuristic performance peaks at an *energy leniency* of 1.5. We observe that the performance benefit for the Max-Max UPE heuristic is less sensitive to the value of *energy leniency*, especially in the range 1.0 to 4.0. The drop in performance for this heuristic in the no-filtering case (compared to its best performance case) is less substantial than the similar difference for the other heuristics. This is because the Max-Max UPE heuristic already accounts for energy consumption, reducing the benefits associated with the energy filter. Therefore, the best-performing case of *energy leniency* for this heuristic is at a higher value than the best-performing case for the other heuristics. The other heuristics require a stricter filtering technique to incorporate energy consumption in allocation choices, therefore they require lower values of *energy leniency* to obtain the best results, because energy is not considered otherwise.

For all heuristics, when we use *energy leniency* values from 0.3 to 0.6, the filtering is so strict that it prevents the heuristic from using all of the available energy that was budgeted for the day. Not being able to use all of the budgeted energy results in fewer tasks being executed and therefore a drop in the total utility earned throughout the day. Alternatively, when using high values of *energy leniency* (and the no-filtering case), all heuristics use all of the day's budgeted energy early in the day and thus are unable to execute tasks that arrive in the later part of the day. We are able to observe this using trace charts that show the gain in total utility and increase in total energy consumption.

Figures 6a and 6b show the utility trace, and Figures 7a and 7b show the energy trace for the Max-Max UPT and the Max-Max UPE heuristics, respectively. For the no-filtering case, we see that the system uses all of the available energy for the day in the early part of the day, and then all future tasks are unable to execute and dropped from the system earning no utility. The no-filtering case for the Max-Max UPE heuristic uses all available energy that was budgeted for the

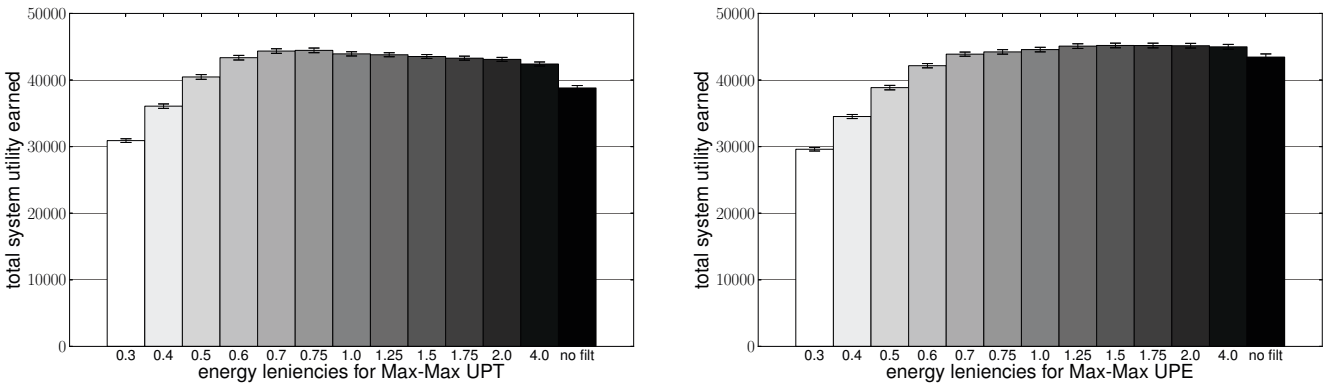


Fig. 5. Sensitivity tests showing the total utility earned as the $energy\ leniency$ is varied for (a) Max-Max UPT and (b) Max-Max UPE. Results averaged over 48 trials with 95% confidence intervals.

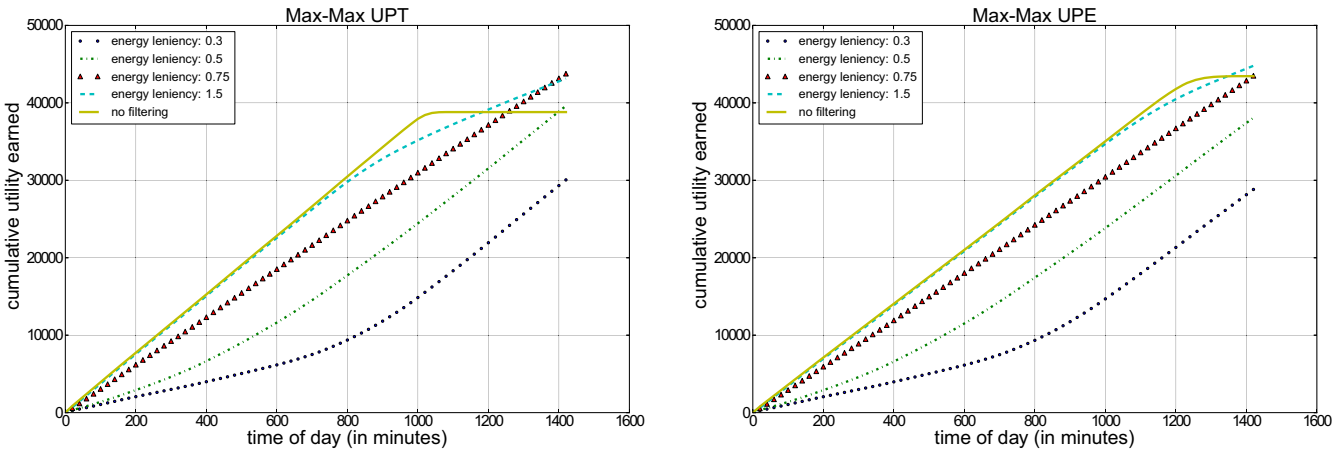
day slightly later (approximately three hours) than the Max-Max UPT heuristic because the heuristic considers energy at each mapping event throughout the day. The slope of its no-filtering energy consumption trace is less steep than the slope of the similar trace for the Max-Max UPT heuristic.

The energy trace charts show the adaptive ability of the filtering technique. Recall the $task\ budget$ is dependent on the aggregate time remaining and the energy remaining. When comparing low values of $energy\ leniency$ to high values of $energy\ leniency$, the energy remaining will be similar at the beginning of the day, but later in the day, there will be more energy remaining for low values and compared to the lower energy remaining for higher values. Therefore the $task\ budget$ will change with the energy remaining, it will become larger when there is more energy remaining in the day and smaller when there is less energy remaining in the day. For example, the slope increases for the energy leniency line of 0.3 during the day in Figures 7a and 7b. Similarly, with high values of $energy\ leniency$, the filter eventually adapts to lower its value of $task\ budget$. This is shown by the decrease in slope for the 1.5 energy leniency line in Figures 7a and 7b. Figure 3 shows the total utility earned by the heuristics in their best $energy\ leniency$ case with filtering and in the no-filtering case compared to the utility earned by Random. Max-Max UPE with energy filtering earns approximately 280% of the utility earned by Random.

The best performance for each of the heuristics comes at an appropriate $energy\ leniency$ that allows the total energy consumption of the heuristic to hit the energy constraint of the day right at the end of the day. Higher values of $energy\ leniency$ result in the energy constraint being hit in the earlier part of the day, while lower values of $energy\ leniency$ can result in a strict environment that prevents the consumption of all of the energy budgeted for the day. Therefore, in energy-constrained environments, the best performance is obtained by permitting allocation choices with a fair-share of energy so that the total energy consumption for the day hits the energy constraint right at the end of the day so that relatively low earning utility tasks that arrive early in the day do not consume energy that could be used by relatively higher utility earning tasks arriving later in the day. If the energy consumption is not regulated, then the allocations in the earlier part of the day can consume too much energy preventing task executions later in the day. Our energy filtering technique gives this ability to the heuristics. Therefore, when designing an energy filter for a heuristic in an oversubscribed environment, the best performance is likely to be obtained when the level of filtering is adjusted to distribute the consumption of the energy throughout the day and meet the constraint right at the end of the day.

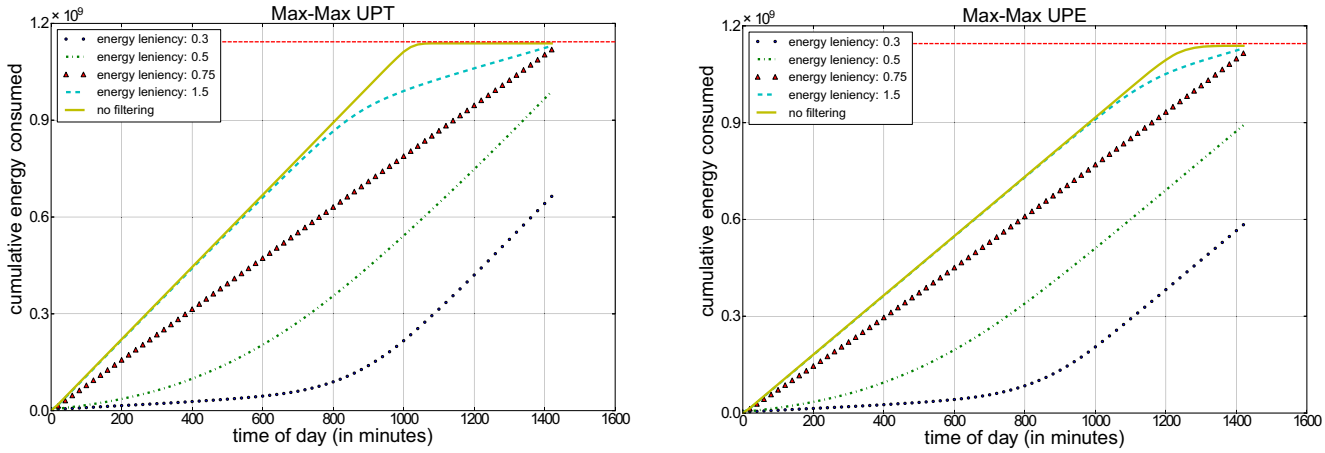
VII. Conclusions and Future Work

In this study, we address the problem of energy-constrained performance maximization. We model an oversubscribed heterogeneous computing system where tasks arrive dynamically and are mapped to machines for execution. The system model is designed based on the needs of the DoD/DOE ESSC environment at ORNL. A heuristic's performance in our system is measured in terms of the total utility that it could earn from task completions. We design an energy-aware heuristic (Max-Max UPE) and compare its performance with heuristics that only optimize utility, and the Random heuristic, and integrate an energy filtering technique into our environment. We show that in an energy-constrained environment our energy-aware heuristic earns more utility than heuristics that only optimize for utility. We also demonstrate that our new energy filtering technique improves the performance of all the heuristics by distributing the consumption of the budgeted energy throughout the day. The energy filtering technique adapts to the energy remaining in the system and accordingly budgets the permitted energy for a task's execution. Max-Max UPE earns 280% of the utility earned by Random in its best performing $energy\ leniency$ case. Max-Max UPE is least sensitive to the level of energy filtering. This is because this heuristic already considers energy consumption. The best performance from the filtering is obtained for all heuristics at a level of filtering that distributes energy consumption approximately



(a) (b)

Fig. 6. Traces of the cumulative utility earned throughout the day at 20 minute intervals as the *energy leniency* is varied for the (a) Max-Max UPT and (b) Max-Max UPE heuristics. The results are averaged over the 48 trials with confidence intervals being extremely tight.



(a) (b)

Fig. 7. Traces of the cumulative energy consumed throughout the day at 20 minute intervals as the *energy leniency* is varied for the (a) Max-Max UPT and (b) Max-Max UPE heuristics. The results are averaged over the 48 trials with confidence intervals being extremely tight. The energy constraint is shown by the dashed line in the top of each plot.

equally throughout the day and meets the energy constraint right at the end of the day. This can be used to guide the design of heuristics and filtering techniques in oversubscribed heterogeneous computing environments.

Possible directions for future research include: (a) using our intuition of meeting the energy constraint right at the end of the day to design adaptive techniques that can auto-tune the value of *energy leniency* dynamically, (b) using stochastic estimates of execution time and power consumption to make allocation decisions that are robust to various sources of uncertainty, (c) examining the performance of the scheduling techniques in different types of heterogeneous environments, (d) designing heuristics that use slope information of the utility-functions to make allocation decisions, and (e) considering workloads of dependent and parallel tasks to broaden the scope of this work.

ACKNOWLEDGMENTS

This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, supported by the Extreme Scale Systems Center at ORNL, which is supported by the Department of Defense. Additional support was provided by a National Science Foundation Graduate Research Fellowship, and by NSF Grant CCF-1302693. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. This research also used the CSU ISTeC Cray System supported by NSF Grant CNS-0923386. The authors thank Mark Oxley for his valuable comments.

REFERENCES

[1] P. Bohrer, E. N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, and R. Rajamony, "The case for power management in web servers,"

Power Aware Computing, pp. 261–289, 2002.

[2] I. Rodero, J. Jaramillo, A. Quiroz, M. Parashar, F. Guim, and S. Poole, "Energy efficient application-aware online provisioning for virtualized clouds and data centers," *Handbook of Energy-Aware and Green Computing*, pp. 541–548, 2012.

[3] J. Koomey, "Growth in data center electricity use 2005 to 2010," *Analytics Press*, Aug. 2011.

[4] D. Filani, J. He, S. Gao, M. Rajappa, A. Kumar, P. Shah, and R. Nagappan, "Dynamic data center power management: Trends, issues, and solutions," *Intel Technology Journal*, vol. 12, no. 1, pp. 59–67, Feb. 2008.

[5] D. J. Brown and C. Reams, "Toward energy-efficient computing," *Communications of the ACM*, vol. 53, no. 3, pp. 50–58, Mar. 2010.

[6] L. D. Briceño, B. Khemka, H. J. Siegel, A. A. Maciejewski, C. Groer, G. Koenig, G. Okonski, and S. Poole, "Time utility functions for modeling and evaluating resource allocations in a heterogeneous computing systems," in *20th Heterogeneity in Computing Workshop (HCW 2011), in the proceedings of the 25th International Parallel and Distributed Processing Symposium (IPDPS 2011)*, May 2011, pp. 7–19.

[7] B. Khemka, R. Friese, L. D. Briceño, H. J. Siegel, A. A. Maciejewski, G. A. Koenig, C. Groer, G. Okonski, M. M. Hilton, R. Rambharos, and S. Poole, "Utility functions and resource management in an oversubscribed heterogeneous computing environment," *under review*.

[8] B. D. Young, J. Apodaca, L. D. Briceño, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, B. Khemka, S. Bahirat, A. Ramirez, and Y. Zou, "Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environments," *The Journal of Supercomputing*, vol. 63, no. 2, pp. 326–347, Feb. 2013.

[9] H. Barada, S. M. Sait, and N. Baig, "Task matching and scheduling in heterogeneous systems using simulated evolution," in *International Heterogeneity in Computing Workshop (HCW 2001), in the proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2001)*, Apr. 2001, pp. 875–882.

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

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

[13] 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, June 1993.

[14] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," in *International Heterogeneity in Computing Workshop (HCW 1996), in the proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 1996)*, Apr. 1996, pp. 86–97.

[15] 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.

[16] R. Friese, B. Khemka, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, J. Rambharos, G. Okonski, and S. W. Poole, "An analysis framework for investigating the trade-offs between system performance and energy consumption in a heterogeneous computing environments," in *22nd Heterogeneity in Computing Workshop (HCW 2013), in the proceedings of the 27th International Parallel and Distributed Processing Symposium (IPDPS 2013)*, May 2013.

[17] M. R. Gary and D. S. Johnson, *Computers and Intractability: A guide to the theory of NP-Completeness*. W. H. Freeman and Co., 1979.

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

[19] L. D. Briceño, H. J. Siegel, A. A. Maciejewski, and M. Oltikar, "Characterization of the iterative application of makespan heuristics on non-makespan machines in a heterogeneous parallel and distributed environment," *The Journal of Supercomputing*, vol. 62, no. 1, pp. 461–485, Oct. 2012.

[20] L. D. Briceño, H. J. Siegel, A. 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*, vol. 22, no. 11, pp. 1780–1787, Nov. 2011.

[21] Q. Ding and G. Chen, "A benefit function mapping heuristic for a class of meta-tasks in grid environments," in *International Symposium on Cluster Computing and the Grid (CCGRID '01)*, May 2001, pp. 654–659.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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*, vol. 19, no. 11, pp. 1445–1457, Nov. 2008.

[27] 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.

[28] M. Wu and W. Shu, "Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems," in *International Heterogeneity in Computing Workshop (HCW 2000), in the proceedings of the International Parallel and Distributed Processing Symposium (IPDPS 2000)*, Mar. 2000, pp. 375–385.

[29] 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.

[30] Y. Tian, E. Ekici, and F. Ozguner, "Energy-constrained task mapping and scheduling in wireless sensor networks," in *IEEE Mobile Adhoc and Sensor Systems Conference*, Nov. 2005, p. 8.

[31] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of bag-of-tasks applications with deadline constraints on dvs-enables clusters," in *IEEE/ACM International Symposium of Cluster Computing and the Grid (CCGrid 2007)*, 2007, pp. 541–548.

[32] R. Friese, T. Brinks, C. Oliver, A. A. Maciejewski, H. J. Siegel, and S. Pasricha, "A machine-by-machine analysis of a bi-objective resource allocation problems," in *The 2013 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2013)*, July 2013, pp. 3–9.

[33] B. Khemka, R. Friese, L. D. Briceño, H. J. Siegel, A. A. Maciejewski, G. A. Koenig, C. Groer, G. Okonski, M. M. Hilton, R. Rambharos, and S. Poole. Creation of utility functions (username= hcw14 password= hcw14). [Online]. Available: http://www.engr.colostate.edu/HJSAAM/utility.pdf

[34] B. Khemka, R. Friese, L. D. Briceño, H. J. Siegel, A. A. Maciejewski, G. A. Koenig, C. Groer, G. Okonski, M. M. Hilton, R. Rambharos, and S. Poole. Creation of arrival pattern (username= hcw14 password= hcw14). [Online]. Available: http://www.engr.colostate.edu/HJSAAM/arrivalpatterns.pdf

[35] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and Sa. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering, Special Issue, Invited*, vol. 3, no. 3, pp. 195–207, Nov. 2000.

[36] Colorado State University ISTeC Cray High Performance Computing Systems. [Online]. Available: http://istec.colostate.edu/activities/cray