Sustainable Computing: Informatics and Systems 5 (2015) 14–30

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

Utility maximizing dynamic resource management in an oversubscribed energy-constrained heterogeneous computing system

Bhavesh Khemka[a,*], Ryan Friese[a,**], Sudeep Pasricha[a,b], Anthony A. Maciejewski[a], Howard Jay Siegel[a,b], Gregory A. Koenig[c], Sarah Powers[c], Marcia Hilton[d], Rajendra Rambharos[d], Steve Poole[c,d]

[a] Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523, USA
[b] Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA
[c] Oak Ridge National Laboratory, One Bethel Valley Road, P.O. Box 2008, MS-6164, Oak Ridge, TN 37831-6164, USA
[d] Department of Defense, Washington, DC 20001, USA

ARTICLE INFO

Article history:
Received 3 March 2014
Received in revised form 28 June 2014
Accepted 5 August 2014

Keywords:
High performance computing system
Energy-constrained computing
Heterogeneous distributed computing
Energy-aware resource management

ABSTRACT

The need for greater performance in high performance computing systems combined with rising costs of electricity to power these systems motivates the need for energy-efficient resource management. Driven by the requirements of the Extreme Scale Systems Center at Oak Ridge National Laboratory, we address the problem of scheduling dynamically-arriving tasks to machines in an oversubscribed and energy-constrained heterogeneous distributed computing environment. Our goal is to maximize total "utility" earned by the system, where the utility of a task is defined by a monotonically-decreasing function that represents the value of completing that task at different times. To address this problem, we design four energy-aware resource allocation heuristics and compare their performance to heuristics from the literature. For our given energy-constrained environment, we also design an energy filtering technique that helps some heuristics regulate their energy consumption by allowing tasks to only consume up to an estimated fair-share of energy. Extensive sensitivity analyses of the heuristics in environments with different levels of heterogeneity show that heuristics with the ability to balance both energy consumption and utility exhibit the best performance because they save energy for use by future tasks.

1. 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]. An August 2013 report by Digital Power Group estimates the global Information-Communications-Technologies ecosystem's use of electricity was approaching 10% of the world electricity generation [3]. As another energy comparison, it was using about 50% more energy than global aviation [3]. In 2007, global data center power requirements were 12 GW and in the four years to 2011, it doubled to 24 GW. Then, in 2012 alone it grew by 63% to 38 GW according to the 2012 Datacenter-Dynamics census [4]. Some data centers are now unable to increase their computing performance due to physical limitations on the availability of energy. For example, in 2010, Morgan Stanley, a global financial services firm based in New York, was physically unable to draw the energy needed to run a data center in Manhattan [5]. Many high performance computing (HPC) systems are now being forced to execute workloads with severe constraints on the amount of energy available to be consumed.

The need for ever increasing levels of 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.

* Corresponding author. Tel.: +1 9704811088.
** Co-corresponding author.
E-mail addresses: Bhavesh.Khemka@colostate.edu (B. Khemka), Ryan.Friese@colostate.edu (R. Friese), Sudeep@colostate.edu (S. Pasricha), AAM@colostate.edu (A.A. Maciejewski), HJ@colostate.edu (H.J. Siegel), Koenig@ornl.gov (G.A. Koenig), PowersSS@ornl.gov (S. Powers), mmskizig@verizon.net (M. Hilton), Jendra.Rambharos@gmail.com (R. Rambharos), swpoole@gmail.com (S. Poole).

http://dx.doi.org/10.1016/j.suscom.2014.08.001
2210-5379/

Specifically, we model a compute facility and workload of interest to 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 design resource management techniques that maximize the performance of their computing systems while obeying a specified energy constraint. Each task has a monotonically-decreasing utility function associated with it that represents the task's utility (or value) based on the task's completion time. The system performance is measured in terms of cumulative utility earned, which is the sum of utility earned by all completed tasks [6]. The example 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. We also create and study heterogeneous environments that are very similar to this example environment but have different heterogeneity characteristics, as quantified by a Task-Machine Affinity (TMA) measure [7]. TMA captures the degree to which some tasks are better suited on some unique machines. An environment where all tasks have the same ranking of machines in terms of execution time has zero TMA. In an environment with high TMA, different tasks will most likely have a unique ranking of machines in terms of execution time. It is important to analyze the impact on performance if the TMA of the environment is changed. We model and analyze the performance of low and high TMA environments compared to the example environment based on interests of the ESSC. This analysis also can help guide the selection of machines to use in a computing system (based on the expected workload of tasks) to maximize the performance obtainable from the system.

In a heterogeneous environment, tasks typically have different 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. To keep our simulations tractable, we consider the energy consumed by the system on a daily basis with the goal of meeting the annual energy constraint. We develop four novel energy-aware resource allocation policies that have the goal of maximizing the utility earned while obeying an energy constraint over the span of a day. We compare these policies with three techniques from the literature designed to maximize utility [6] 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 the idea presented in [8]) for our environment. The goal of the filtering technique is to remove high energy consuming allocation choices that use more energy than an estimated fair-share. This step improves the distribution of the allotted energy across the whole day. We perform an in-depth analysis to demonstrate the benefits of our energy filter. We also study the performance of all the heuristics in the low and high TMA environments and perform extensive parameter tuning tests.

In summary, we make the following contributions: (a) the design of four new resource management techniques that maximize the utility earned, given an energy constraint for an oversubscribed heterogeneous computing system, (b) the design of a custom energy filtering mechanism that is adaptive to the remaining energy, enforces "fairness" in energy consumed by tasks, and distributes the energy budgeted for the day throughout the day, (c) a method to generate new heterogeneous environments that have low and high TMA compared to the environment based on interests of the ESSC without changing any of its other heterogeneity characteristics, (d) show how heuristics that only maximize utility can become energy-aware by adapting three previous techniques to use an energy filter, (e) a sensitivity analysis for all the heuristics to the parameter that controls the level of energy-awareness and/or level of energy filtering, (f) an analysis of the performance of all the heuristics in the low and high TMA environments, and (g) a recommendation on how to select the best level of filtering or the best balance of energy-awareness versus utility maximization for heuristics 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 3 describes our resource management techniques. We then provide an overview of related work in Section 4. Our simulation and experimental setup are detailed in Section 5. In Section 6, we discuss and analyze the results of our experiments. We finish with our conclusion and plans for future work in Section 7.

2. Problem description

2.1. System model

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

Our computing system environment consists of a suite of heterogeneous machines, where each machine belongs to a specific machine type (rather than a single large monolithic system, such as Titan [9]). 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. Because the system is heterogeneous, 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.

We assume that for a task of type i on a machine of type j running in P-state k, 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 [10–16]. The APC incorporates both the static power (not affected by the P-state of the task) and the dynamic power (different for different P-states). We can compute the Estimated Energy Consumption ($EEC(i, j, k)$) by taking the product of 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 [17]. 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. Due to the sensitive nature

Table 1
Acronyms used and their meanings

Acronyms	Meaning
ORNL	Oak Ridge National Laboratory
DoD	United States Department of Defense
DOE	United States Department of Energy
ESSC	Extreme Scale Systems Center at ORNL
TMA	Task-Machine Affinity
TDH	Task Difficulty Homogeneity
MPH	Machine Performance Homogeneity
DVFS	Dynamic Voltage and Frequency Scaling
P-states	Performance states of a processor
ETC	Estimated Time to Compute
APC	Average Power Consumed
EEC	Estimated Energy Consumption
ECS	Estimated Computation Speed
U-E wf	Utility-Energy Weighting Factor

of DoD operations, for the ESSC environment, historical data is not publicly available. Therefore, for the simulation study conducted in this paper, we synthetically create our ETC and APC matrices based on the recommendations provided by ESSC.

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 [18]. We do not allow the preemption of tasks, i.e, once a task starts execution, it must execute until completion.

We model three degrees of heterogeneity by varying the TMA of the system [7]. TMA uses singular value decomposition (SVD) for its computation and captures the degree to which certain tasks execute faster on certain unique machines. Section 5.3.2 describes how TMA is computed. Task Difficulty Homogeneity (TDH) and Machine Performance Homogeneity (MPH) are given as orthogonal metrics for quantifying the heterogeneity of the system [7]. TDH and MPH capture the homogeneity in the aggregate performance of tasks and machines, respectively. We study the performance of all the heuristics in an example environment (based on the interests of the ESSC), a modified environment with low TMA, and a modified environment with high TMA. The low and high TMA environments differ only in their TMA compared to the example environment. All three environments have similar values of TDH and MPH.

In an environment with extremely low TMA, all tasks will have the same sorted ordering of machines in terms of execution time. The actual execution time values of different tasks on the machines can be different, but all the tasks will rank the machines in the same order. In contrast, in an extremely high TMA environment, each task will have a unique ordering of machines if the machines were to be sorted in terms of execution time for that task. Environments that have different TMA but similar MPH and TDH do not have more powerful or less powerful machines or more difficult or less difficult tasks in general, but instead they differ in the level of uniqueness of the affinity of different tasks to the machines. An environment with higher TMA does not have more powerful machines, but instead has machines that are suited to perform well for different tasks. On the contrary, in an environment with lower TMA, it is easy to rank machines in terms of performance (irrespective of the tasks). It is desirable to analyze the performance of these different types of systems. Given the expected workload for an environment, such analyses can help guide the selection of resource management heuristics and associated tuning parameters in use. In Section 5.3.2, we describe our technique to create the relatively low and high TMA environments with negligible difference in the values of MPH and TDH of the system. Table 1 mentions the different acronyms used in this study and their meanings.

2.2. Problem statement

The amount of useful work accomplished by a compute system is captured by the total utility it earns from task completions. Recall that we consider a workload model where tasks arrive dynamically. The resource manager does not know which task will arrive next, the utility function of the task, nor its task type. Therefore, the goal of the resource manager is to maximize the total utility that can be earned from completing tasks while satisfying an annual energy constraint.

Many computing centers (including the ESSC environment that we consider) have budget constraints on annual expenditure toward energy consumption of the compute system. This annual energy-cost budget can be translated into an annual energy constraint. To simplify the problem, we divide the annual energy constraint into daily energy constraints and ensure that a given day's energy constraint is met. If models are provided that give the variation in energy cost with the time of the year and give the expected workload arriving into the system during the different times of the year, one could use such information to calculate an appropriately scaled value for a given day's energy constraint ($energy\ constraint_{day}$). If such models are not available, one can approximate $energy\ constraint_{day}$ by dividing 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 $energy\ constraint_{day}$. This artificial energy constraint allows the system to control its energy consumption on a per-day basis to be able to meet the actual energy constraint, which is for the year. We use the duration of a day to keep the simulation time tractable. Instead of one day, we could base our constraint on 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 task's execution time in each day. This is done so that each day gets the utility for the task executions that consumed its energy to permit a fair comparison of different heuristic approaches. Systems such as the ESSC are designed so that constraints on power (i.e., energy per unit time) are not a concern.

3. Resource management

3.1. 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 [19]. A mapping event occurs 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 (i.e., more tasks arrive than can possibly be executed while they are still valuable), 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. Fig. 1 shows a small example system with four machines, the executing tasks, the tasks in the pending slots, and the virtual queues of the scheduler. At a mapping event, the batch-mode heuristics make scheduling decisions

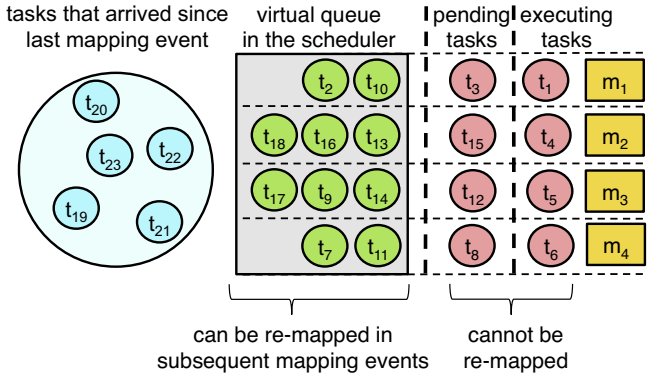


Fig. 1. An example system of four machines showing tasks that are currently executing, waiting in pending slots, waiting in the virtual queue, and have arrived since the last mapping event (and are currently unmapped).

for a set of tasks comprising 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 set. 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]) to the new environment, design four new energy-aware batch-mode heuristics, and analyze and compare their performances.

The scheduling decisions made for a task may depend on the energy that currently remains in the system for the day (among other factors). The value of the per day energy constraint can change each day based on the energy that has been consumed thus far in the year. Therefore, we want to avoid making scheduling decisions for a task that starts its execution on the next day. Therefore, the batch-mode heuristics are not allowed to map a task to a machine where it will start execution on the next day. In addition, the batch-mode heuristics are not allowed to schedule a task that will violate the day's energy constraint.

If no machine can start the execution of the task within the current day or if the task will violate the current day's energy constraint, 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 mappable tasks set and the heuristics make mapping decisions for these tasks as well.

3.2. Batch-mode heuristics

We present four new heuristics that try to maximize the utility earned while being energy-aware. The Max-Max Utility-Per-Energy consumption (**Max-Max UPE**) heuristic 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 [20–29]. In the first stage, the heuristic finds for each task independently in the mappable tasks set the machine and P-state that maximizes "utility earned/energy consumption." If none of the machine-P-state choices for this 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 mappable tasks set. 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, updates the machine's ready time, and repeats this process iteratively until all tasks are either mapped or postponed.

The Weighted Utility (**Weighted Util**) heuristic is designed to explicitly control the extent to which allocation decisions should be biased toward maximization of utility versus minimization of energy consumption. It does this by using a utility-energy weighting factor (*U-E weighting factor* or *U-E wf*) that controls the relative significance of normalized utility and normalized energy in the heuristic's objective function. To normalize the utility value across different allocation choices, we divide by the maximum utility any task in the system could have (*max util*). For normalizing energy consumption, we determine the highest EEC value from all possible task, machine, and P-state combinations (*max energy consumption*). The Weighted Util heuristic is also a two-stage heuristic. In the first stage, for each mappable task, it finds the machine-P-state pair that has the highest value for the weighted expression:

$$(1 - U\text{-}E\,wf) \times \frac{utility\ earned}{max\ util} - U\text{-}E\,wf \times \frac{energy\ consumed}{max\ energy\ consumption}. \quad (1)$$

As done in Max-Max UPE, if no allocation choices for this task satisfy the day's energy constraint or start the execution of the task within the current day, then, the task is removed from the mappable tasks set and is postponed to be considered the next day. In the second stage, it picks from the pairs of the first stage the one that has the highest value for the above expression, makes that assignment, removes that task from the set of mappable tasks, updates that machine's ready time, and repeats the two stages iteratively until all tasks have either been mapped or postponed.

By normalizing the utility and the energy terms in the weighted expression, we ensure that each of those terms has a value between 0 and 1. This makes it convenient to compare utility and energy in a single expression as we do. The value of *U-E wf* can be varied between 0 and 1 to bias the effect of the normalized energy term to the value of the weighted expression.

The Weighted Utility-Per-Time (**Weighted UPT**) heuristic is similar to the Weighted Util heuristic but the normalized utility term in expression 1 is replaced by "normalized utility earned/normalized execution time." To normalize the execution time, we determine the minimum execution time in the ETC matrix from all task, machine, P-state choices (*min execution time*). The weighted expression for this heuristic is:

$$(1 - U\text{-}Ewf) \times \frac{utility\ earned/execution\ time}{max\ util/\min\ execution\ time} - U\text{-}E\,wf \times \frac{energy\ consumed}{max\ energy\ consumption}. \quad (2)$$

The Weighted Utility-Per-Energy consumption (**Weighted UPE**) heuristic is similar to the Weighted Util heuristic but the normalized utility term in expression 1 is replaced by "normalized utility earned/normalized energy consumption." To normalize energy, we determine the minimum energy consumption value in the EEC matrix from all task, machine, P-state choices (*min energy consumption*). The weighted expression for this heuristic is:

$$(1 - U\text{-}E\,wf) \times \frac{utility\ earned/energy\ consumed}{max\ util/\min\ energy\ consumption} - U\text{-}E\,wf \times \frac{energy\ consumed}{max\ energy\ consumption}. \quad (3)$$

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 while there still remains energy in the day.

The Min-Min Completion time (**Min-Min Comp**) heuristic is a fast heuristic adapted from [6] 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] 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 [6] 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.

We collectively refer to the Weighted Util, Weighted UPT, and Weighted UPE heuristics as the weighted heuristics. We also collectively refer to the Min-Min Comp, Max-Max Util, Max-Max UPT, and Max-Max UPE heuristics as the non-weighted heuristics.

The weighted heuristics can be viewed as more generalized versions of their non-weighted counterparts. For example, the Weighted Util heuristic can be viewed as a more generalized version of the Max-Max Util heuristic. If *U-E wf* = 0, then Weighted Util reduces to Max-Max Util. For higher values of *U-E wf*, the Weighted Util heuristic is more energy-aware and has the goal of simultaneously maximizing utility while minimizing energy.

3.3. Dropping low utility earning tasks

We use a technique to drop tasks with low potential utility at the current time (introduced in our previous work [6]). Dropping a task means that it will never be mapped to a machine. This operation allows the batch-mode heuristics to 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 users that are satisfied.

The dropping operation reduces the number of scheduling choices to consider and therefore at a mapping event the dropping operation is performed before the heuristic makes its 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 allocation decisions.

Because of oversubscription in our environment, the number of tasks in the mappable tasks set 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 mappable tasks set and enable the heuristics to complete their execution within the mapping interval time (a minute). This allows the heuristics to move any new high utility-earning task to the front of the virtual queue to complete its execution sooner.

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 instead of being postponed.

3.4. Energy filtering

The goal of our new energy filter technique is to remove potential allocation choices (task-machine-P-state combinations) from a heuristic's consideration if the allocation choice 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 based on the energy remaining in the day and the time remaining in the day. Therefore, the value of the *task budget* is recomputed at the start of every mapping event. We do not recompute the value of the *task budget* within a mapping event (based on the allocations made by the heuristic in that mapping event) because we want the *task budget* to only account for execution information that is guaranteed to occur (i.e., executing and pending tasks).

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 Eq. 4.

$$energy\ \ remaining = energy\ \ constraint_{day} - energy\ consumed - energy\ scheduled \quad (4)$$

To estimate the *task budget*, the filter also needs to compute the time remaining in the day within which the above energy can be consumed. The availability time of a machine is set to either the completion time of the last task to be queued for the machine or the current time, whichever is greater. At the start of the mapping event, the last task to be queued for 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 availability time of the machine. Fig. 2 shows its computation for an example small-scale system with three machines. As shown, even though machine m3 is not executing a task 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 task types, machine types, and P-states is represented as *average execution time*. The energy filtering technique needs to estimate the total number 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 execution time*. The energy

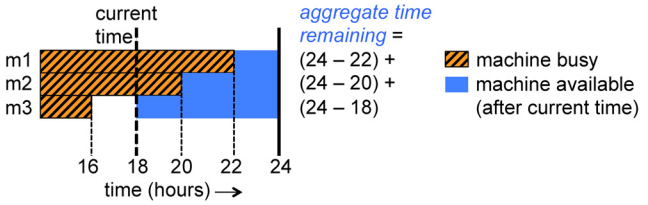


Fig. 2. An example system of three machines showing the computation of *aggregate time remaining*. It represents the total computation time available from the current time till the end of the day.

filter has to use *average execution time* because the resource manager 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* imply more leeway for high energy allocation choices to pass through the filter, whereas a low value for the *energy leniency* would filter out many more choices. This value is determined empirically. The *task budget* is computed using Eq. 5.

$$task\ \ budget = energy\ \ leniency \times \frac{energy\ \ remaining}{\left(\frac{aggregate\ \ time\ \ remaining}{average\ \ execution\ \ time}\right)} \tag{5}$$

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*.

4. Related work

Heterogeneous task scheduling in an energy-constrained computing environment is examined in [27]. The authors model an environment where devices in an ad-hoc wireless network are limited by battery capacity and each task has a fixed priority. This differs significantly for our environment where we model a larger and more complex heterogeneous system with a utility performance metric based on the exact completion time of each task rather than a metric that aims to finish more higher priority tasks, as in their work. Additionally, in our study, the energy available for use under the constraint is shared across all resources, while in [27] each resource has its own energy constraint (a battery).

In [6], the concept of utility functions to describe a task's time-varying importance is introduced. The authors deal with the problem of maximizing the total utility that can be earned from task completions. Energy is not considered at all in that paper. In this work, we are concerned with maximizing utility while obeying an energy constraint. For accomplishing this, we design four new energy-aware heuristics and an energy filtering technique that adapts to the remaining energy. We perform extensive analysis of all the heuristics along with parameter tuning tests. We also create low and high TMA environments and examine the performance of the heuristics in these environments.

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 (i.e., makespan) 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 undersubcribed 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 (based on each task's completion time) and not the number of tasks that meet their hard deadlines. The authors of [8] use heuristics that map each task to a machine as soon as the task arrives and do not allow remapping, whereas in our environment we map groups of tasks at a time, allowing us to use more information when making allocation decisions and can also remap tasks in the virtual queue. The concept of an energy filter is used in [8], and we build on that for a more complex filter.

In [32], the authors formulate a bi-objective resource allocation problem to analyze the trade-offs between makespan and energy consumption. Their approaches use total makespan as a measure of system performance as opposed to individual task utility values as we do in our work. Additionally, they model static environments where the entire workload is a single 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.

5. Simulation setup

5.1. Overview

We simulate the arrival and mapping of tasks over a two day span 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. All the simulation experiments were run on the ISTeC Cray System at Colorado State University [18]. In this system, each node has 24 cores. The nodes are scheduled to a user as a whole, and therefore, we average the results of our experiments across 48 simulation trials (instead of say 50). Each of the trials represents a new workload of tasks (with different utility functions, task types, and arrival times), and a different computing 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.

5.2. Workload generation

A utility function for each task in a workload is given and each task has a maximum utility value that starts at one of 8, 4, 2, or 1. These values are based on the plans of the ESSC, but for other environments, different values of maximum utility may be used. Furthermore, for our environment we have four choices of maximum utility but in other environments greater or fewer choices

may be used. A method for generating utility functions can be found in [6].

For our simulation study, we generate the arrival patterns to closely match patterns of interest to ESSC [6]. In this environment, general-purpose tasks arrive in a sinusoidal pattern and special-purpose tasks follow a bursty arrival pattern.

5.3. Execution time and power modeling

5.3.1. Example environment

This example environment is based on the expectation of some future DoD/DOE environments. In our simulation environment, approximately 50,000 tasks arrive during the duration of a day and each of them belongs to one of 100 task types. Out of the 100 task types, 83 were general-purpose and 17 were special-purpose task types. Each task type has approximately the same number of tasks in it. Furthermore, each task's utility function is generated using the method in [6]. 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. Each of the four special-purpose machine types has 2, 2, 3, and 3 machines on them, respectively, for a total of ten special-purpose machines. The remaining 90 machines are general-purpose and are split into the remaining nine machine types as follows: 5, 5, 5, 10, 10, 10, 10, 15, and 20. The machines of a special-purpose machine type run a subset of special-purpose task types approximately ten times faster on average than the general-purpose machines can run them (as discussed below). The special-purpose machines do not have the ability to run tasks of other task types. In our environment, three to five special-purpose task types are special for each special-purpose machine type.

We assume that all machines have three P-states in which they can operate. We use techniques from the Coefficient of Variation (COV) method [33] 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. The mean value of the static power for the machines was set to 66 watts and the mean dynamic power was set to 133 watts. 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. We refer to this set of matrices as the example environment.

5.3.2. Low and high TMA environments

We modify the ETC matrices at the highest power P-state of the example environment to create low and high TMA environments with minimal difference in the MPH and TDH of the environments. All three of these measures are functions of the Estimated Computation Speed (ECS) matrices. An ECS matrix is created by taking the reciprocal of each entry in the ETC matrix. Fig. 3 shows two 3×3 ECS matrices that significantly differ in TMA but would have the same value for TDH. In the high TMA ECS matrix, each task has a unique ranking of the machines in terms of speed of execution.

very low TMA	machine1	machine 2	machine 3
task 1	25	10	5
task 2	25	10	5
task 3	25	10	5

high TMA	machine1	machine 2	machine 3
task 1	25	10	5
task 2	5	25	10
task 3	10	5	25

Fig. 3. Two sample 3×3 ECS matrices that have equal Task Difficulty Homogeneity but very different values of Task Machine Affinity. In the matrix with the high TMA, each task has a unique ranking of machines in terms of execution speed, i.e., for task 1 the best to worst machines are: 1, 2, and 3, whereas for task 2 the ranking of machines would be: 2, 3, and 1. In contrast, in the very low TMA matrix, all tasks would have the same ranking of machines.

To make the TMA measure orthogonal to the MPH and TDH measures, alternate row and column normalizations are performed on the ECS matrix so that the matrix has equal row sums and equal column sums (called a standard matrix) before the TMA is computed for the ECS matrix [7]. As mentioned in [7], this iterative procedure is not guaranteed to converge to a standard matrix if the ECS matrix can be organized into a block matrix (after reordering rows and columns) with one block containing only 0s. Such a matrix is said to be decomposable [7]. In our environment, the entries in the ECS matrix for the special-purpose machines and the tasks that are not special on them contain 0s (because the special-purpose machines are unable to execute them), and therefore, our ECS matrices are decomposable. To overcome this problem, we remove the columns of the special-purpose machines from the matrices and compute the TMA of only the general-purpose-machines matrix that has the general-purpose machines. We then modify this part of the ECS matrix to have low and high TMA. For each of the low and high TMA matrices, we then obtain the values for the special-purpose tasks (on the one machine on which they are special) by taking the average of the entries of this task from the general-purpose-machines matrix and multiplying that average speed by 10. By doing this, we retain the characteristics of the special-purpose machines and tasks as desired by the ESSC at ORNL, but we are able to study the performance of the heuristics in environments with different TMA. The TMA of the general-purpose-machines ECS matrices do not capture the actual TMA of the whole environment. However, as we are only concerned with creating relatively low and high TMA matrices compared to the example environment, our computation of the TMA measure is valid for our purposes.

As mentioned in [7], after a standard matrix is obtained (by performing alternate row and column normalizations), the first step for computing the TMA of the matrix is to determine the SVD. An ECS matrix with T task types and M machine types has dimension $T \times M$. The SVD results in the factorization $U\Sigma V^T$. The U and V^T are orthogonal matrices representing the column and the row space, respectively. The matrix Σ is a diagonal matrix consisting of $min(T, M)$ singular values (σ_i) along the diagonal in a monotonically decreasing order, i.e., $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{min(T,M)} \geq 0$. The singular values represent the degree of correlation (linear dependence) between the columns of the ECS matrix. The higher the first singular value (σ_1), the more correlated the columns of the matrix are. The higher the other singular values ($\sigma_2, \sigma_3, \ldots, \sigma_{min(T,M)}$), the less correlated all the columns are. When the correlation in the columns of

Table 2
Range of Task-Machine Affinity (TMA) values for the 48 simulation trials of the different environments

Type of ETC	TMA range
example environment	0.082–0.091
low TMA	$<10^{-15}$
high TMA	0.14–0.18

the ECS matrix is low (i.e., high TMA), most tasks will have unique ranking of the machines in terms of execution speed. Alternatively, when the correlation in the columns of the ECS matrix is high (i.e., low TMA), most tasks will have the same ranking of machines in terms of execution speed performance. The TMA of the matrix is defined as the average of all the non-maximum singular values (i.e., not including σ_1) of the standard matrix:

$$TMA = \sum_{i=2}^{\min(T,M)} \frac{\sigma_i}{(\min(T,M)-1)}. \quad (6)$$

For creating the low TMA matrices, we want to have an environment where the columns of the ECS matrix are completely correlated. We do this by removing the effect of all non-maximum singular values and only retain the characteristics of the first singular value. This is equivalent to taking the rank-1 approximation of the ECS matrix (i.e., make all the non-maximum singular values to be equal to 0).

For creating the high TMA matrices, simply increasing the non-maximum singular values or decreasing the maximum singular value will typically result in matrices with negative values or result in very little increase in the TMA. Negative values in an ECS matrix are meaningless representations of execution speed of a task on a machine and are therefore undesirable. We design an iterative method to increase the components of the ECS matrix in the directions that result in higher TMA while making sure that negative values are never introduced into the matrix. To do this, we first take the SVD of the ECS matrix A. Let u_1, u_2, u_3, etc., represent the columns of the U matrix and let v_1^T, v_2^T, v_3^T, etc., represent the rows of the V^T matrix resulting from the SVD of A. Our goal is to increase the non-maximum singular values (without introducing negative values) to get a new ECS matrix with a higher TMA. We examine how much of the matrix resulting from the product of u_2 and v_2^T we can add to A, without making any of the elements in the matrix negative. We then do this with u_3 and v_3^T, and continue to do this iteratively for all the non-maximum singular values. This allows one to increase the TMA of the environment without having any negative values in the matrix.

These procedures to create low and high TMA environments ensure that only the TMA of the environment is affected while maintaining the other characteristics of the matrix. The MPH and the TDH of the low and high TMA environments have negligible difference compared to that of the example environment. Table 2 shows the range of TMA values for the matrices of the example environment and also of the low and high TMA environments (that we created using our technique described above). Recall that the construction of these TMA environments are for simulation experiments to evaluate the heuristics. They are not part of the heuristics or system model.

5.4. Obtaining an energy constraint

In many real-world scenarios, an annual energy budget is typically given for an HPC system. As mentioned in Section 2.2, we can estimate the energy constraint of the current day using a given annual energy constraint to help ensure each day uses an appropriate 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 [6]) for a full 24-hour time period, disregarding the energy constraint. Based on the resource allocations generated by this heuristic, we average the total energy consumption throughout the day across 48 simulation trials and use 70% of this average value as our energy constraint. As a result, for our simulations, we used an energy constraint value of 1.11 GJ per day.

6. Results

6.1. 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 HPC cluster [18]). We first discuss the performance of the heuristics in the energy-constrained environment when not using the energy filtering technique. All the heuristics used a dropping threshold of 0.5 units of utility to tolerate the oversubscription, i.e., a task is dropped if the best possible utility it can earn is lower than 0.5. We use a dropping threshold of 0.5 units of utility as it gave the best performance in our previous work [6]. 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. We then analyze the performance of the heuristics in the relatively low and the high TMA environments and finally compare the best performing case for all the heuristics in the different types of environments.

6.2. Example environment results in no-filtering case

Fig. 4 shows the total utility earned by the heuristics in the filtering and no-filtering cases. We first discuss the performance of the heuristics in the no-filtering case. Our four new heuristics outperform the heuristics from the literature (i.e., Min-Min Comp, Max-Max Util, and Max-Max UPT). Among the non-weighted heuristics, Max-Max UPE earns the highest utility even though it consumes the same amount of energy as the others. This is because

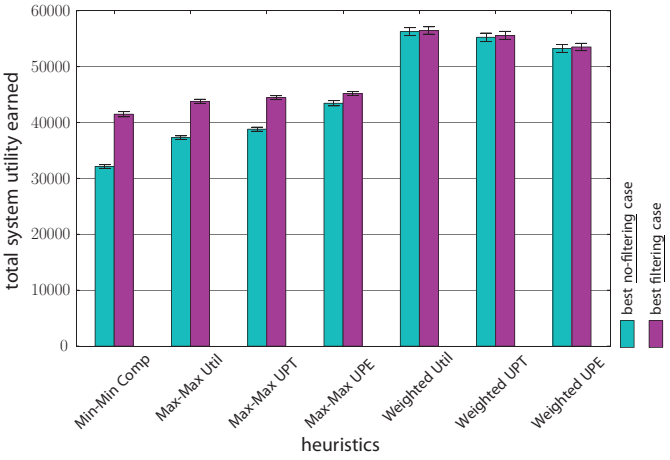


Fig. 4. Total utility earned by the heuristics in the no-filtering case and their best filtering case (case with the best performing value of *energy leniency*). For the weighted heuristics, in both the cases, the best performing $U-E$*weighting factor* was chosen. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. These results are for the example environment. The results are averaged over 48 trials with 95% confidence intervals.

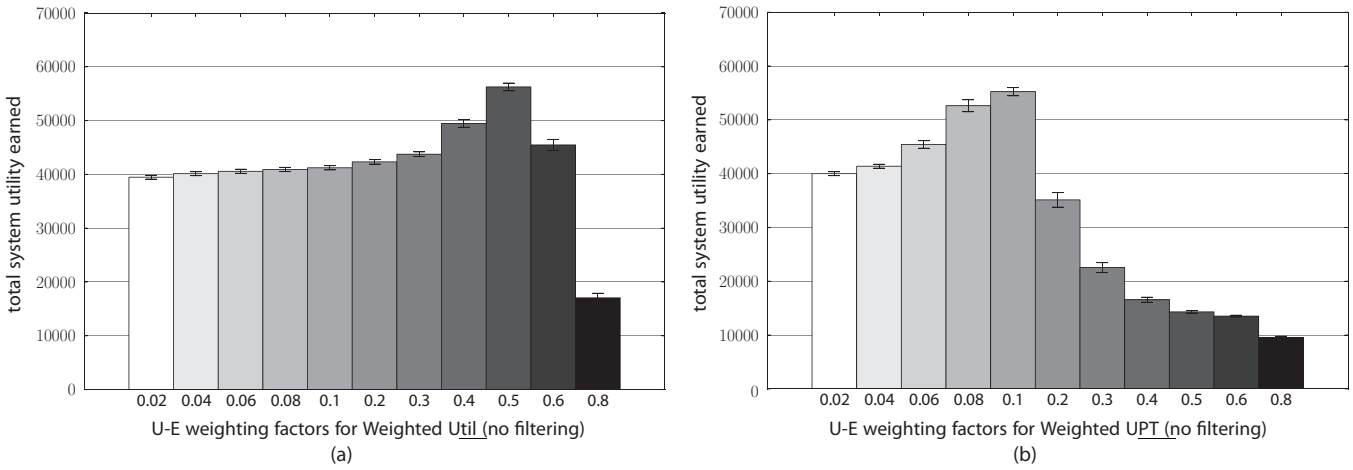


Fig. 5. Tests showing the total utility earned in the no-filtering case as the *U-E weighting factor* is varied for (a) Weighted Util and (b) Weighted UPT. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. These results are for the example environment. The results are averaged over 48 trials with 95% confidence intervals.

the heuristic accounts for energy consumption while trying to maximize utility, and thus is able to avoid high energy-consuming allocation choices without significantly affecting the utility earned. Once the allotted energy for the day (*energy constraint*$_{day}$) is consumed, the heuristics were not allowed to map any more tasks until the following day.

The weighted heuristics perform much better than the other heuristics because of their ability to balance the extent to which they want to bias their allocations toward utility maximization versus energy minimization. For each of the weighted heuristics, tests were performed with different values of the *U-E weighting factor* from 0.02 to 0.8. A higher *U-E weighting factor* biases the allocation decisions more toward energy minimization. Fig. 5a and b show the total utility earned by the Weighted Util and Weighted UPT heuristics for different values of the $U-E weightingfactor$. Similarly, Fig. 6a and b show the total energy consumption by the Weighted Util and Weighted UPT heuristics for different values of the U-E weighting factor. Weighted UPE showed similar trends as the Weighted UPT heuristic.

As we vary the *U-E weighting factor* from 0.02 to 0.8, the utility earned by the weighted heuristics increases and then decreases. At very low values of the *U-E weighting factor*, the energy term in the weighted expression has very little impact and the Weighted Util and Weighted UPT heuristics approaches the Max-Max Util and Max-Max UPT heuristics, respectively. With very high values of the *U-E weighting factor*, the heuristics are too conservative in their energy expenditure and only execute tasks that consume the least energy with little regard to the utility being earned. As can be seen in Fig. 5a and b, the best performance is obtained between these extremes. For the Weighted Util heuristic, the best performance is obtained at a *U-E weighting factor* that is larger than the best performing *U-E weighting factor* for the Weighted UPT and Weighted UPE heuristics. This is because the Weighted Util heuristic completely depends on the energy term in the weighted expression to be energy-aware. The Weighted UPE and the Weighted UPT heuristics already are able to account for energy minimization (directly or indirectly) using the first part of their weighted expression, and therefore need a smaller portion of the energy term to help make good low energy consuming choices. The energy consumption by these heuristics always hits the energy constraint for the weighting factors lower than the best performing weighting factor. Higher values of the *U-E weighting factor* further minimize energy consumption (so it unnecessarily goes below the constraint) leading to reduced performance.

To illustrate why *U-E weighting factor* of 0.5 is the best for the Weighted Util heuristic, Fig. 7a and b show the trace of the total

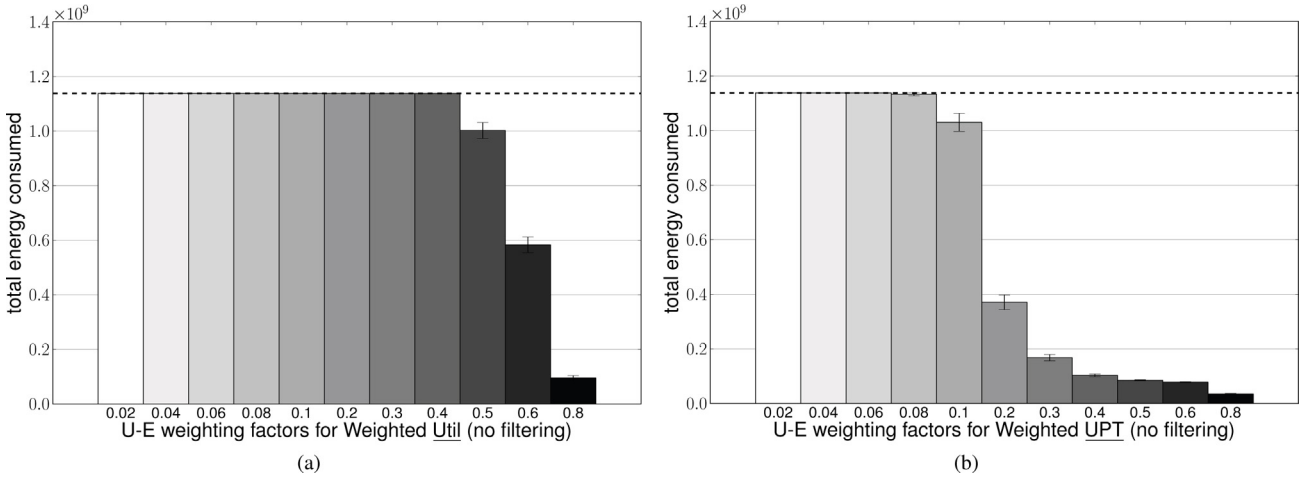


Fig. 6. Tests showing the total energy consumption in the no-filtering case as the *U-E weighting factor* is varied for (a) Weighted Util and (b) Weighted UPT. The dashed horizontal line shows the energy constraint of the system. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. These results are for the example environment. The results are averaged over 48 trials with 95% confidence intervals.

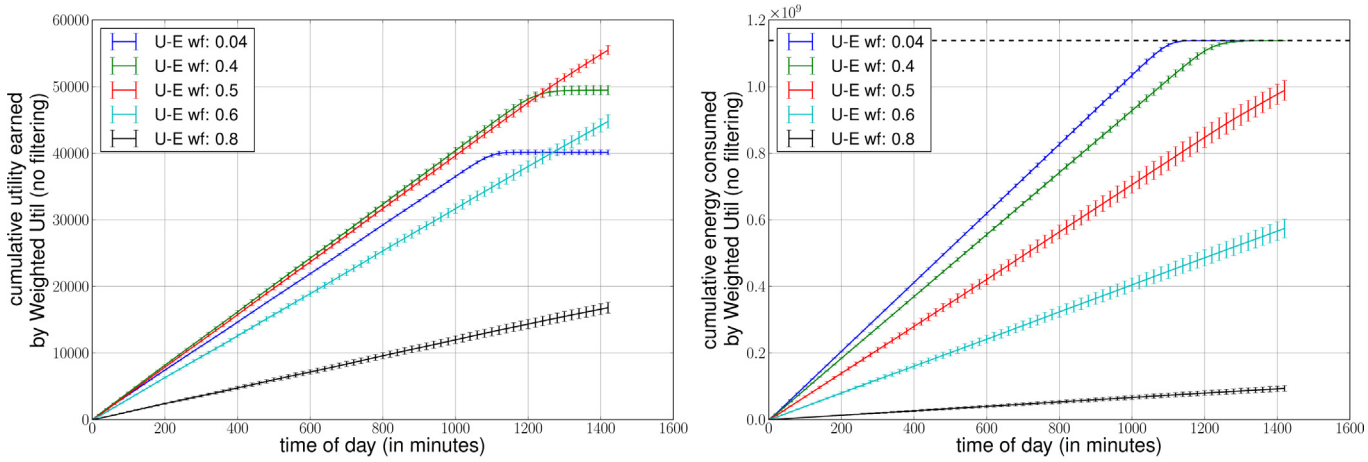


Fig. 7. Traces of (a) the cumulative utility earned and (b) the cumulative energy consumption for the Weighted Util heuristic in the no-filtering case throughout the day at 20 minute intervals at different *U-E weighting factors*. The dashed horizontal line in (b) shows the energy constraint of the system. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. These results are for the example environment. The results are averaged over 48 trials with 95% confidence intervals.

utility being earned and the trace of the total energy consumption for the Weighted Util heuristic in the no-filtering case for different values of the *U-E weighting factor*. For *U-E weighting factors* 0.4 and lower, the energy constraint is hit before the end of the day and therefore high utility-earning tasks that arrive after that point in time are unable to execute. This causes a drop in performance because no utility is earned from such tasks after this point. Values of *U-E weighting factor* higher than 0.5 are too conservative in energy consumption and have too much energy left over at the end of the day.

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 maximizing utility. In the no-filtering case, Weighted Util is approximately 28% better than Max-Max UPE, and Max-Max UPE is approximately 11% better than Max-Max UPT (the best-performing utility maximization heuristic from the literature [6]).

6.3. Example environment 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 Eq. 5). 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. We first analyze the performance of the non-weighted heuristics in Section 6.3.1, and then examine the weighted heuristics in Section 6.3.2.

6.3.1. Non-weighted heuristics

Fig. 8a and b 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. Fig. 9 show 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 sensitivity tests of the energy consumed for Min-Min Comp, Max-Max Util, and Max-Max UPT showed trends similar to Max-Max UPE.

In general, for the non-weighted heuristics, as we increase the value of *energy leniency* from 0.3, the utility earned increases and

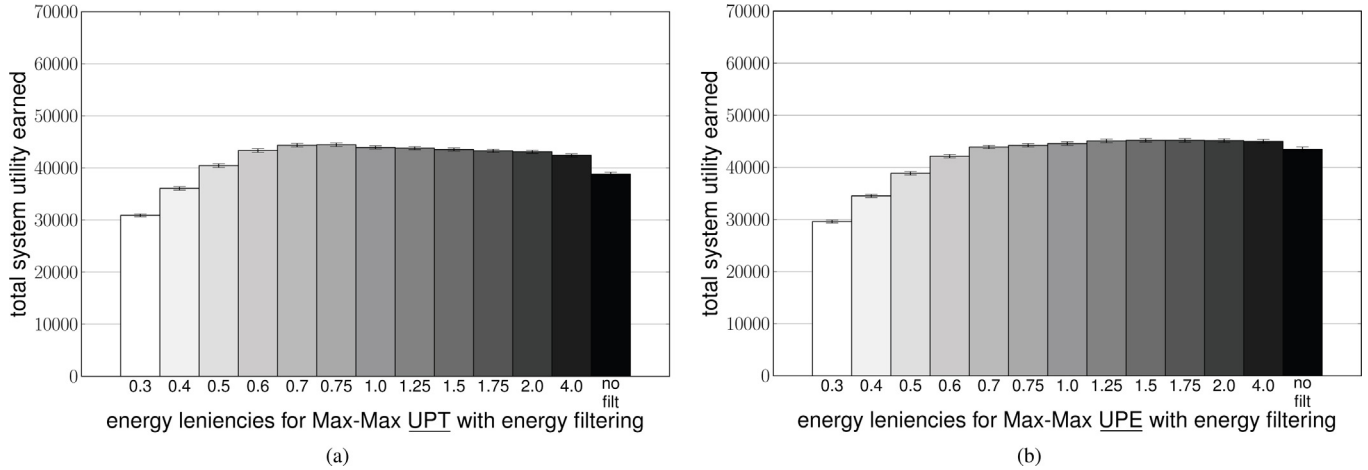


Fig. 8. Sensitivity tests showing the total utility earned as the *energy leniency* is varied for (a) Max-Max UPT and (b) Max-Max UPE. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. These results are for the example environment. The results are averaged over 48 trials with 95% confidence intervals.

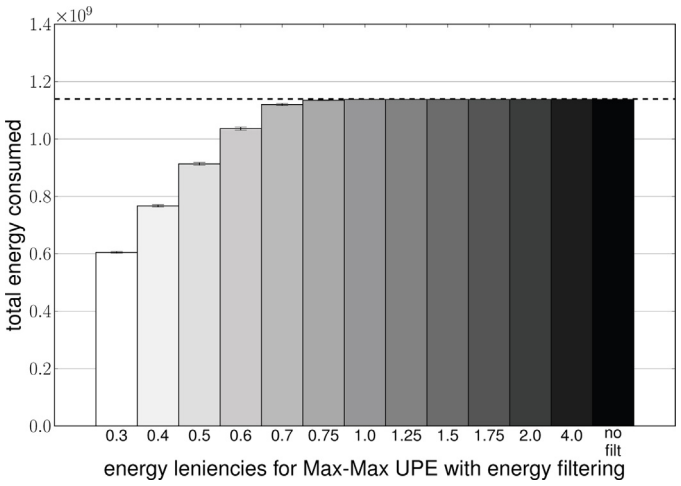


Fig. 9. Sensitivity tests showing the total energy consumed as the *energy leniency* is varied for the Max-Max UPE heuristic. The dashed horizontal line shows the energy constraint of the system. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. These results are for the example environment. The results are averaged over 48 trials with 95% confidence intervals.

then decreases as we approach the no-filtering case. All of these 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 1.0–4.0. 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 of *energy leniency* 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 the non-weighted 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.

Fig. 10a and b show the utility trace for the Max-Max UPT and the Max-Max UPE heuristics, respectively. Fig. 11a and b 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 are 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 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. As a result, Max-Max UPE is able to execute more tasks and earn higher utility.

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 compared to the lower energy remaining for higher values. Therefore, because 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 Fig. 11a and b. 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 Fig. 11a and b.

The best performance for each of the non-weighted 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, saving enough energy for any high-utility earning tasks that may arrive at later parts in the day. Higher values of *energy leniency* (above 1.5) result in the energy

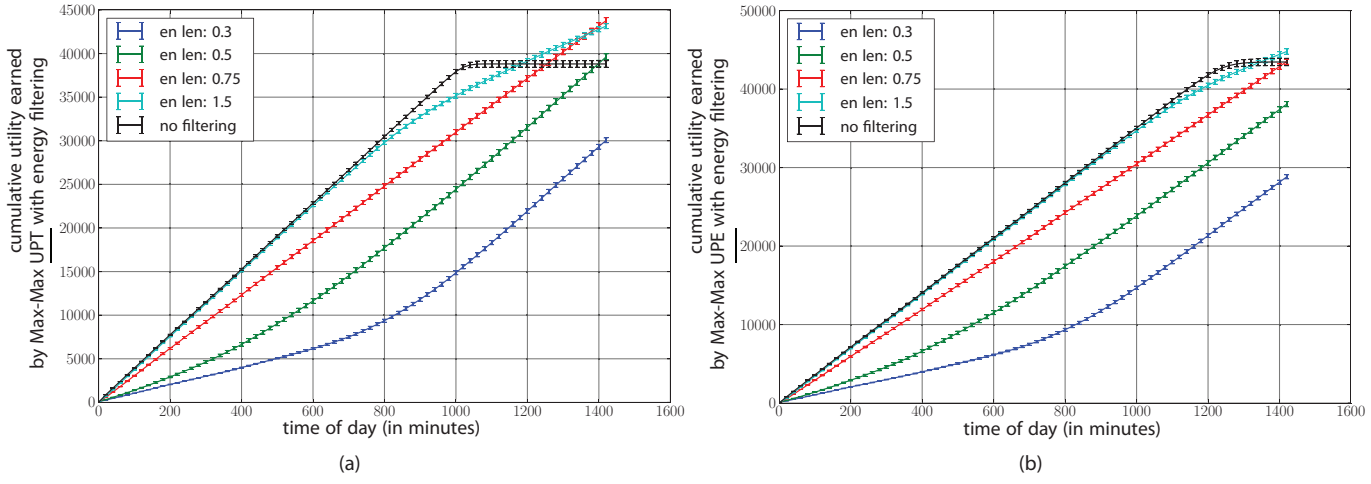


Fig. 10. Traces of the cumulative utility earned throughout the day at 20 minute intervals as the *energy leniency* (en len) is varied for the (a) Max-Max UPT and (b) Max-Max UPE heuristics. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. These results are for the example environment. The results are averaged over 48 trials with 95% confidence intervals.

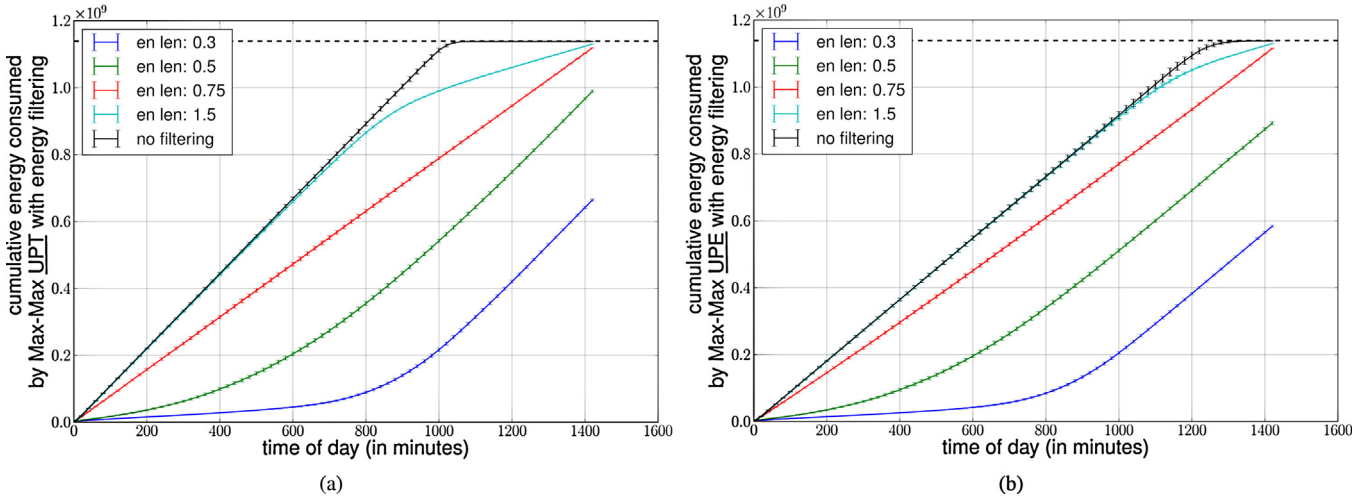


Fig. 11. Traces of the cumulative energy consumed throughout the day at 20 minute intervals as the *energy leniency* (en len) is varied for the (a) Max-Max UPT and (b) Max-Max UPE heuristics. The dashed horizontal line shows the energy constraint of the system. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. These results are for the example environment. The results are averaged over 48 trials with 95% confidence intervals.

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. By doing so relatively low utility-earning 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 these heuristics.

Among the non-weighted heuristics, Max-Max UPE performs the best and its performance is the least sensitive to the value of *energy leniency*. These are because this heuristic accounts for the energy consumed to earn each unit of utility. The performance of all the non-weighted heuristics improves because of the energy filtering technique. 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. This can be used to design filters for such heuristics in energy-constrained environments to maximize performance.

As mentioned in Section 5.4, we set the energy constraint to approximately 70% of the average total energy consumed by an unconstrained execution of the Max-Max UPT heuristic. If the energy constraint was set to a lower percentage than 70%, then the relative performance of the heuristics would be exaggerated, further highlighting the benefit of the filtering and weighting techniques. On the contrary, having a very high energy constraint value, would reduce the relative performance benefit of these techniques. Changing the energy constraint essentially changes the time at which the trace charts of the no filtering-no weighting case start to plateau. This directly impacts the total utility they can earn. The less tighter the energy constraint, the later the trace charts start to plateau, and as a result, the higher the total utility they earn.

6.3.2. Weighted heuristics

The weighted heuristics already have the ability to tune their energy consumption throughout the day and therefore they do not benefit from the energy filtering technique, as shown in Fig. 4. Fig. 12 shows the utility earned by the Weighted Util heuristic for different combinations of *U-E weighting factor* and *energy leniency*. As seen in Fig. 6a, even in the no-filtering case, the Weighted Util heuristic did not consume the total energy budgeted for the day with *U-E weighting factors* 0.5 and above. Therefore, adding energy filtering (that may further limit the energy consumption) to these cases does not help to improve the performance of the heuristic in comparison to the no-filtering case. Using moderate values of *energy leniency* helps in cases where we use lower *U-E weighting factors*, because in these cases, the weighting factor alone is unable to accomplish the desired level of energy minimization. At the best performing *U-E weighting factor* (i.e., 0.5 for Weighted Util), the no-filtering case performs just as well as the best performing *energy leniency* cases.

Both, the filtering technique and the weighting technique, have the ability to regulate the energy consumption throughout the day and allow the energy constraint to be hit only at the end of the day, but the weighting technique performs better than the filtering technique. We now analyze why the best performing *U-E weighting factor* (without any energy filtering, i.e., *energy leniency* of infinity) performs better than the best performing energy filtering case (without using any weighting, i.e., *U-Eweightingfactor* = 0). To study this difference, we plot the utility and energy trace charts of the Weighted Util heuristic for the following three scenarios:

1 no filtering (*energy leniency* = infinity) and no weighting (*U-E weighting factor* = 0),
2 best filtering case (*energy leniency* = 0.75) and no weighting, and
3 no filtering and best weighting case (*U-Eweightingfactor* = 0.5).

These trace charts are shown in Fig. 13a and b. Recall that without the weighting i.e., a *U-E weighting factor* of 0, the weighted heuristics reduce to their non-weighted counterparts (e.g. Weighted Util becomes Max-Max Util).

The weighting case outperforms the filtering case because of two reasons. Each of these reasons can be explained by examining the trace up to the point where the no filtering-no weighting case hits the energy constraint at approximately 1000 minutes. Recall that the no filtering-no weighting case is only attempting to maximize utility with no regard to energy. The first reason why the weighting performs better than the filtering is because the filtering removes allocation choices that consume more energy than the fair-share *task budget* (of a mapping event) without considering the utility

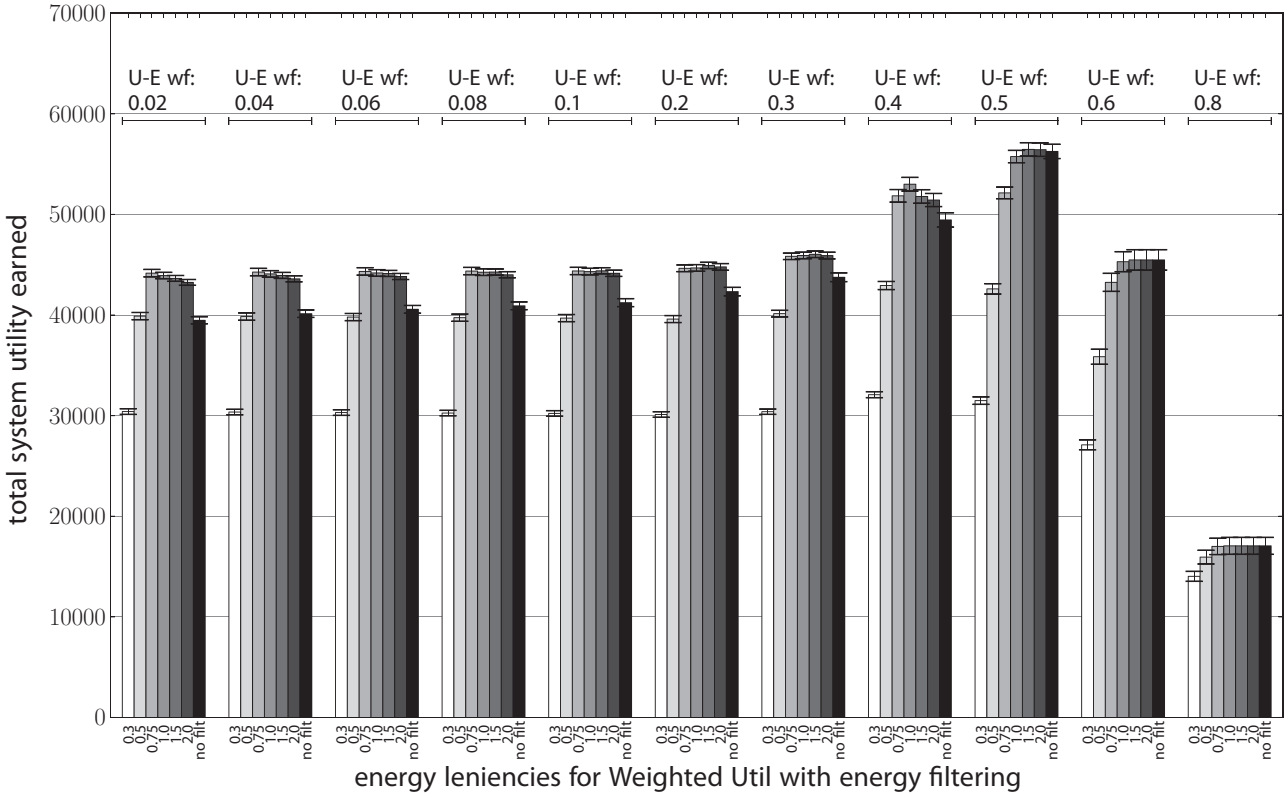


Fig. 12. Sensitivity tests showing the total utility earned for different combinations of *U-E weighting factor* (*U-E wf*) and *energy leniency* for the Weighted Util heuristic. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. These results are for the example environment. The results are averaged over 48 trials with 95% confidence intervals.

that that allocation choice may earn. This causes the filtering case to avoid certain high energy consuming (but high utility earning) allocation choices to execute. This can be seen by the lower values of utility being earned by the heuristic in the filtering case compared to the no filtering-no weighting case up to 1000 minutes. The weighting case does not have this problem as it is able to rank choices in terms of both the utility they earn and the energy they consume. So, if an allocation choice consumes high energy but proportionally earns high utility then this allocation choice may be a viable option in the weighting case. Weighting, if tuned correctly, allows the heuristic to balance the required amount of energy minimization versus utility maximization. The second reason why the weighting case performs better is because the weighting case biases decisions to pick low energy consuming allocation choices, and in our environment this also leads to minimization of execution time. Therefore, Weighted Util gets the ability to make allocations that behave as utility-per-time. Because of the minimization of execution time, we observed that the weighting case was able to complete

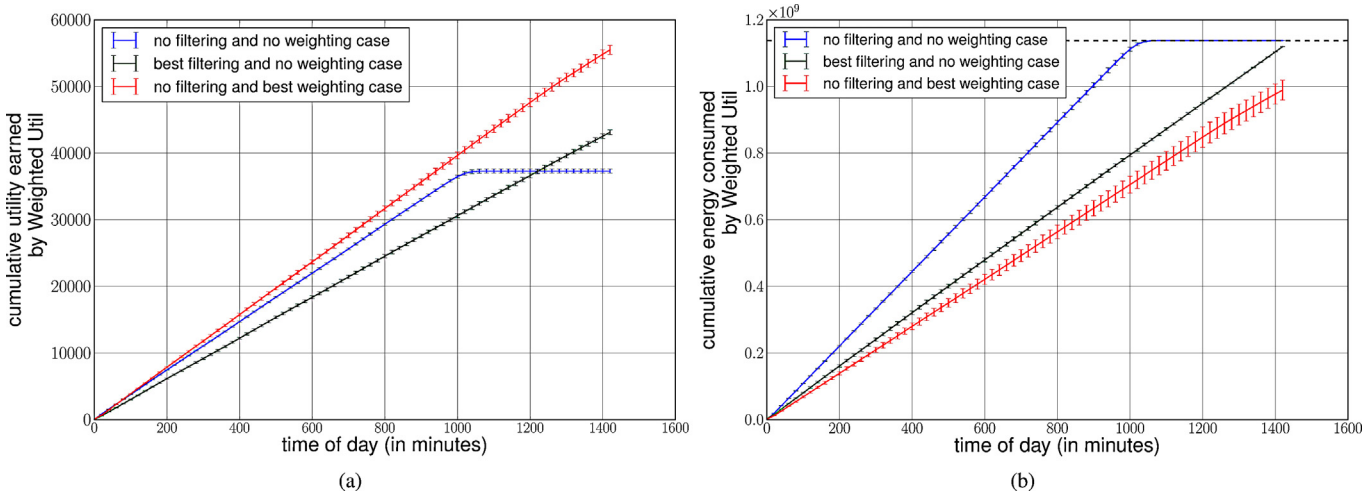


Fig. 13. Traces of (a) the cumulative utility earned and (b) the cumulative energy consumption throughout the day at 20 minute intervals of different cases of using the best/not using energy filtering and/or weighting for the Weighted Util heuristic. The dashed horizontal line in (b) shows the energy constraint of the system. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. These results are for the example environment. The results are averaged over 48 trials with 95% confidence intervals.

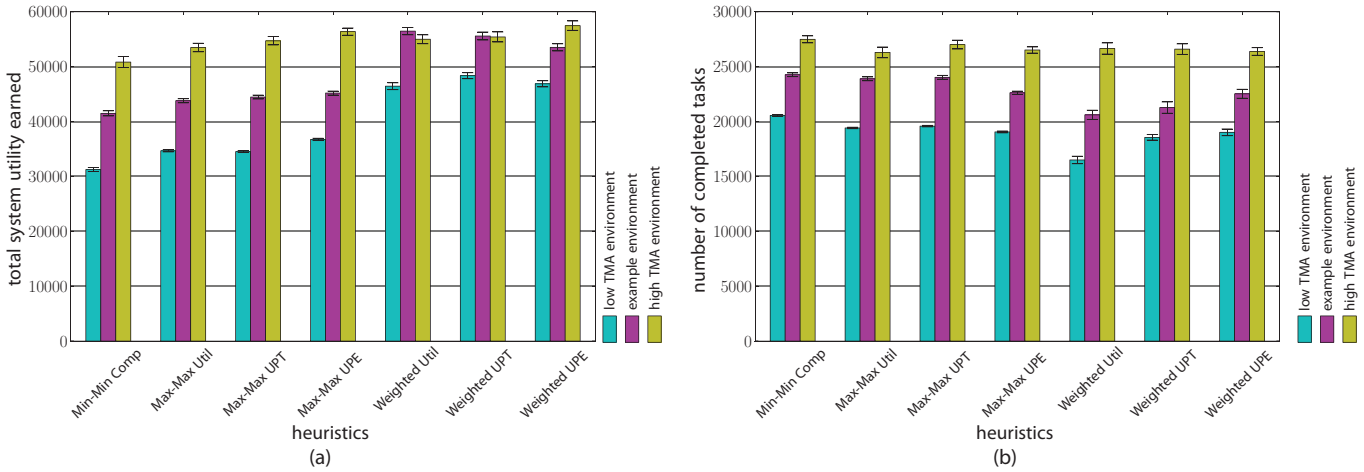


Fig. 14. (a) Total utility earned and (b) total number of completed tasks by the best performing cases for all the heuristics with energy filtering in the three types of environments: low TMA, example, and high TMA. The simulated system has 100 machines and approximately 50,000 tasks arriving in the day. The results are averaged over 48 trials with 95% confidence intervals.

many more tasks compared to the no filtering-no weighting case. This causes the heuristic's weighting case to earn higher utility than the unconstrained no filtering-no weighting case even in the region up to 1000 minutes.

The weighting and filtering techniques both allow a heuristic to regulate its energy consumption throughout the day and permit task executions in the later part of the day that help them to earn more utility than a case that does not use either weighting or filtering. Filtering does not attempt to minimize energy consumption, it only tries to prune high energy consuming choices. It makes its filtering mechanism stricter or more lenient if the energy consumption rate is higher or lower than what it should ideally be, respectively. As opposed to this, weighting works by attempting to minimize energy consumption right from the start of the allocation process. It picks allocation choices that have a good balance of earning utility versus consuming energy.

Fig. 4 shows the total utility earned by the heuristics in the no-filtering case and their best *energy leniency* case. The non-weighted heuristics have a significant performance increase with the energy filtering because it allows them to control their energy expenditure throughout the day. The weighted heuristics already have the ability to control their energy consumption and therefore do not have any increase in performance when using energy filtering.

As mentioned in Section 5.4, we set the energy constraint to approximately 70% of the average total energy consumed by an unconstrained execution of the Max-Max UPT heuristic. Changing the energy constraint essentially changes the time at which the trace charts of the no filtering and no weighting case start to plateau. This directly impacts the total utility they can earn. The looser the energy constraint, the later the trace charts start to plateau, and as a result, the higher the total utility they earn. If the energy constraint was set to a lower percentage than 70%, then the relative performance of the heuristics would be exaggerated, further highlighting the benefit of the filtering and weighting techniques. On the contrary, having a very high energy constraint value (loose constraint), would reduce the relative performance benefit of these techniques.

6.4. Low and high TMA environment results with filtering

We ran similar experiments as mentioned thus far for all the heuristics with the low and high TMA environments. These environments have the same set of task utility functions, task arrival times, oversubscription levels, dropping threshold, overall aggregate performance of the machines and tasks with similar values of MPH and TDH compared to the example environment. The only difference is the TMA of the environment, and that affects the uniqueness by which certain task types execute faster on certain machine types. In the low TMA environment, all tasks have the same ranking of machines in terms of execution time, whereas in the high TMA environment, most tasks have unique ranking of the machines in terms of execution time performance. We ran parameter tuning tests to find the best performing *energy leniency* case for the non-weighted heuristics and the best performing *energy leniency* and *U-E weighting factor* case for the weighted heuristics. Here, we summarize the conclusions obtained from these tests. The actual results of the tests are omitted from the draft for brevity.

For the non-weighted heuristics in the low TMA environment, the best performance was obtained at a higher value of *energy leniency*. This was because in the low TMA environments, as all tasks have the same ranking of the machine types in terms of execution time, fewer tasks get to execute on the machines where they have better execution times. On average, this results in longer task execution times than that in the example environment. Therefore, the *average execution time* estimate is not as accurate as it was in the example environment. To compensate for this, the best performance is obtained at higher values of *energy leniency*. See Eq. 5 for understanding how an increase in *energy leniency* helps to make up for the lower value of the *average execution time* estimate.

For the weighted heuristics in the low TMA environment, the best performance was obtained at lower values of the *U-E weighting factor* (i.e., more weighting toward utility term) compared to the *U-E weighting factors* for the best case in the example environment. The energy term of the weighted expression does not account for how oversubscribed the machine is. In a low TMA environment, all tasks would have the same ranking of machine-P-state choices in terms of energy consumption. Assigning all tasks to these energy efficient machines would oversubscribe them resulting in low utility being earned from the tasks. By having a lower value for the *U-E weighting factor*, the preferred allocation choice would be less biased toward the minimization of energy consumption, therefore allowing for a better load balance in a low TMA environment.

In contrast, in high TMA environments, the non-weighted heuristics performed their best at lower values of *energy leniency*. In such environments, different tasks have different ranking of the machine types in terms of execution time and this makes it easier for tasks to be assigned to their best execution time machine type. As a result, the estimate of the *average execution time* (calculated

based on ETC values) is not as accurate as it was in the example environment. The best performance is obtained at lower values of *energy leniency* as that helps to compensate for the higher value of the *average execution time* estimate.

For the weighted heuristics, in high TMA environments, the best performance is obtained at higher values of *U-E weighting factor* (i.e., more weighting toward energy term). This is because in the high TMA environment, different tasks would generally have different machine-P-state choices that execute fast and consume less energy. Therefore, biasing the scheduling decisions more toward the minimum energy consumption choices provides automatic load balancing across the machine types and assigns tasks to the machine types that execute them the best (i.e., quickest and least energy consuming).

Fig. 14a shows the utility earned by the best performing case of each of the heuristics in the low and high TMA environments in comparison to the best performance obtained in the example environment. The overall trend is that heuristics earn less utility in the low TMA environment and more utility in the high TMA environment compared to the example environment. This is because in the low TMA environment, all tasks have the same machine type as their best execution time machine type and as a result fewer tasks get to execute on the machines of this type. This leads to longer execution times on average for the tasks and results in fewer tasks being pushed through the system during the day. Alternatively, in the high TMA environment, different task types execute fastest on different machine types, and therefore, assigning tasks to their best execution time machine type is feasible as it implicitly provides load balancing. In such an environment, the average execution time of the tasks is lower and more tasks complete execution during the day earning higher utility overall. In high TMA environments, resource allocation decisions are easier to make (as minimizing execution time provides some load balancing) and therefore the performance of all the heuristics is quite similar. Although, the weighted heuristics still perform slightly better than their non-weighted counterparts. Fig. 14b shows that compared to the example environment, the number of tasks completed for the low TMA environment is significantly fewer while the number of tasks completed for the high TMA environment is significantly greater. The average execution time of a task across the machines (taken from the ETC) is similar for the low and example environment and slightly higher for the high TMA environment. We still get better performance in the high TMA environment from all the heuristics because of the diversity in the tasks' execution time performance across machine types. The number of dropped tasks can be obtained by subtracting the numbers in Fig. 14b from 50,000 (which is approximately the total number of tasks arriving into the system).

7. Conclusions and future work

In this study, we address the problem of energy-constrained utility 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 types of systems of interest to DoD/DOE. A heuristic's performance in our system is measured in terms of the total utility that it could earn from task completions. We design four heuristics for this problem and compare their performance with other heuristics adapted from our previous work, and integrate an energy filtering technique into our environment.

We show that in an energy-constrained environment our energy-aware heuristics earns more utility than heuristics that only optimize for utility. Our new energy filter helps to improve the performance of all the non-weighted 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. For the non-weighted heuristics, the best performance from the filtering is obtained for all heuristics at a level of filtering that distributes energy consumption approximately 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.

The weighted heuristics have the ability to minimize energy consumption throughout the day and can be tuned so that their energy consumption meets the energy constraint right at the end of the day. These heuristics outperform their non-weighted counterparts (even when they use energy filtering). This is because, filtering considers energy regardless of utility while weighting considers them together. The filtering removes certain high energy-consuming allocation choices, but among the remaining choices it simply picks the one that maximizes the objective of the heuristic. Alternatively, the weighted heuristics rank all allocation choices by accounting for both utility and energy and have the ability to balance the degree to which they are energy-aware. As a result, these heuristics perform much better than the non-weighted heuristics (even when they use filtering).

In the low and high TMA environments, all the heuristics earn lower and higher utility overall, respectively. This is because the higher the TMA of the environment, the higher the number of tasks that can be completed because more tasks can be assigned to the machine on which they have the fastest execution time. This is because assigning tasks to their best execution time machine implicitly balances the load. Also, in high TMA environments, we observe that mapping decisions are easier to make and most heuristics perform similarly.

One of the main goals of this study was to perform an in-depth analysis of the performance of the energy-constrained heuristics. As part of future work, we plan to use our knowledge of the performance of the heuristics, filtering, and weighting techniques to design adaptive techniques that can auto-tune the value of *energy leniency* and/or *U-E weighting factor* dynamically. Other possible directions for future research include: (1) designing energy-aware robust resource allocation techniques that account for various sources of uncertainty, such as stochastic task execution times, (2) creating different patterns for the arrival of utility into the system (e.g., high utility tasks arriving for certain fixed times of the day) and designing techniques that can account for and adapt to these changes, (3) designing heuristics that use the information about the slope of the utility-functions to make allocation decisions, (4) considering workloads of dependent and parallel tasks to broaden the scope of this work, (5) extending our model to simulate task and machine failures, and (6) exploring how to represent service level agreements using utility functions.

Acknowledgments

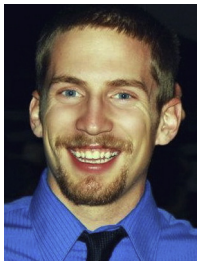
This research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, supported by the Extreme Scale Systems Center at ORNL, which is supported by the Department of Defense. Additional support was provided by a National Science Foundation Graduate Research Fellowship, and by NSF Grants CCF-1302693 and CCF-1252500. 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 and Eric Jonardi for their valuable comments. A preliminary version of portions of this material appeared in [34]. This work builds upon the workshop paper with the design of three new heuristics (i.e., the weighted heuristics) for the energy-constrained environment that perform better than their corresponding conference counterparts (i.e., the non-weighted heuristics). We analyze why the weighting technique outperforms the filtering technique even though they both regulate the energy consumption and hit the energy constraint close to the end of the day. We also design a method to create ETCs of lower and higher Task Machine Affinity than the example environment, i.e., different degrees of heterogeneity. We perform extensive heuristic parameter tuning tests and analyze the performance of all the heuristics in the example, low, and high TMA environments.

References

[1] P. Bohrer, E.N. Elnozahy, T. Keller, M. Kistler, C. Lefurgy, C. McDowell, R. Rajamony, The case for power management in web servers, in: R. Graybill, R. Melhem (Eds.), Power Aware Computing, ser. Series in Computer Science, Springer, USA, 2002.
[2] I. Rodero, J. Jaramillo, A. Quiroz, M. Parashar, F. Guim, S. Poole, Energy-efficient application-aware online provisioning for virtualized clouds and data centers, in: International Green Computing Conference, August, 2010, pp. 31–45.
[3] M.P. Mills, The Cloud Begins With Coal – Big Data, Big Networks, Big Infrastructure, and Big Power, Digital Power Group, Available: http://www.tech-pundit.com/wp-content/uploads/2013/07/Cloud_Begins_With_Coal.pdf
[4] DatacenterDynamics Industry Census, Available: http://www.datacenterdynamics.com/blogs/industry-census-2012-emerging-data-center-markets
[5] D.J. Brown, C. Reams, Toward energy-efficient computing, Commun. ACM 53 (3) (2010, March) 50–58.
[6] B. Khemka, R. Friese, L.D. Briceno, H.J. Siegel, A.A. Maciejewski, G.A. Koenig, C. Groer, G. Okonski, M.M. Hilton, R. Rambharos, S. Poole, Utility functions and resource management in an oversubscribed heterogeneous computing environment, IEEE Trans. Comput. (2014) (accepted to appear).
[7] A.M. Al-Qawasmeh, A.A. Maciejewski, R.G. Roberts, H.J. Siegel, Characterizing task-machine affinity in heterogeneous computing environments, in: 20th Heterogeneity in Computing Workshop (HCW 2011), Proceedings of the IPDPS 2011 Workshops & PhD Forum (IPDPSW), May, 2011, pp. 33–43.
[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, Y. Zou, Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environments, J. Supercomput. 63 (2) (2013, February) 326–347.
[9] Introducing Titan, 2014, June, Available: https://www.olcf.ornl.gov/titan/
[10] H. Barada, S.M. Sait, N. Baig, Task matching and scheduling in heterogeneous systems using simulated evolution, in: 10th Heterogeneous Computing Workshop (HCW 2001), Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), April, 2001, pp. 875–882.
[11] M.K. Dhodhi, I. Ahmad, A. Yatama, An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems, J. Parallel Distrib. Comput. 62 (9) (2002, September) 1338–1361.
[12] A. Ghafoor, J. Yang, A distributed heterogeneous supercomputing management system, IEEE Comput. 26 (6) (1993, June) 78–86.
[13] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurr. 6 (3) (1998, July) 42–51.
[14] A. Khokhar, V.K. Prasanna, M.E. Shaaban, C. Wang, Heterogeneous computing: challenges and opportunities, IEEE Comput. 26 (6) (1993, June) 18–27.
[15] H. Singh, A. Youssef, Mapping and scheduling heterogeneous task graphs using genetic algorithms, in: 5th Heterogeneous Computing Workshop (HCW'96), April, 1996, pp. 86–97.
[16] D. Xu, K. Nahrstedt, D. Wichadakul, QoS and contention-aware multi-resource reservation, Clust. Comput. 4 (2) (2001, April) 95–107.
[17] R. Friese, B. Khemka, A.A. Maciejewski, H.J. Siegel, G.A. Koenig, S. Powers, M. Hilton, J. Rambharos, G. Okonski, 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), Proceedings of the IPDPS 2013 Workshops & PhD Forum (IPDPSW), May, 2013, pp. 19–30.
[18] Colorado State University ISTeC Cray High Performance Computing Systems, Available: http://istec.colostate.edu/activities/cray
[19] M.R. Gary, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., 1979.
[20] 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, B. Yao, A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems, J. Parallel Distrib. Comput. 61 (6) (2001, June) 810–837.
[21] L.D. Briceño, H.J. Siegel, A.A. Maciejewski, M. Oltikar, J. Brateman, J. White, J. Martin, K. Knapp, Heuristics for robust resource allocation of satellite weather data processing onto a heterogeneous parallel system, IEEE Trans. Parallel Distrib. Syst. 22 (11) (2011, November) 1780–1787.
[22] Q. Ding, 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.
[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, S.S. Yellampalli, Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment, J. Parallel Distrib. Comput. 67 (2) (2007, February) 154–169.
[24] S. Ghanbari, M.R. Meybodi, On-line mapping algorithms in highly heterogeneous computational grids: a learning automata approach, in: International Conference on Infomation and Knowledge Technology (IKT'05), May, 2005.
[25] Z. Jinquan, N. Lina, J. Changjun, A heuristic scheduling strategy for independent tasks on grid, in: International Conference on High-Performance Computing in Asia-Pacific Region, November, 2005.
[26] K. Kaya, B. Ucar, C. Aykanat, Heuristics for scheduling file-sharing tasks on heterogeneous systems with distributed repositories, J. Parallel Distrib. Comput. 67 (3) (2007, March) 271–285.
[27] J.-K. Kim, H.J. Siegel, A.A. Maciejewski, R. Eigenmann, Dynamic resource management in energy constrained heterogeneous computing systems using voltage scaling, IEEE Trans. Parallel Distrib. Syst. 19 (11) (2008, November) 1445–1457.
[28] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, R.F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, J. Parallel Distrib. Comput. 59 (2) (1999, November) 107–121.
[29] M. Wu, W. Shu, Segmented min-min: a static mapping algorithm for meta-tasks on heterogeneous computing systems, in: 9th Heterogeneous Computing Workshop (HCW 2000), March, 2000, pp. 375–385.
[30] Y. Tian, E. Ekici, F. Ozguner, Energy-constrained task mapping and scheduling in wireless sensor networks, in: IEEE Mobile Adhoc and Sensor Systems Conference, November, 2005, p. p8.
[31] K.H. Kim, R. Buyya, J. Kim, Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled 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, 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] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, Sa. Ali, Representing task and machine heterogeneities for heterogeneous computing systems, Tamkang J. Sci. Eng. 3 (3) (2000, November) 195–207 (Special Issue, Invited).
[34] B. Khemka, R. Friese, S. Pasricha, A.A. Maciejewski, H.J. Siegel, G.A. Koenig, S. Powers, M. Hilton, R. Rambharos, S. Poole, Utility driven dynamic resource management in an oversubscribed energy-constrained heterogeneous system, in: 23rd Heterogeneity in Computing Workshop (HCW 2014), Proceedings of the IPDPS 2014 Workshops & PhD Forum (IPDPSW), May, 2014, pp. 58–67.

Bhavesh Khemka is a PhD student and research assistant in the Electrical and Computer Engineering Department at Colorado State University. He received his BE degree in Electrical and Electronics Engineering from Hindustan College of Engineering affiliated with Anna University, India. His research interests include fault-tolerant, energy-aware, and robust resource allocations in heterogeneous and distributed computing environments.



Ryan Friese received dual BS degrees in Computer Engineering and Computer Science from Colorado State University in 2011. He received his MS degree in Electrical Engineering from Colorado State University in the Summer of 2012. He is currently a PhD student at Colorado State University. He is a United States National Science Foundation graduate research fellow. His research interests include heterogeneous computing as well as energy-aware resource allocation.

Sudeep Pasricha is an Associate Professor in the Electrical and Computer Engineering Department at Colorado State University. His research interests include embedded, mobile, and high performance computing, with an emphasis on energy-aware and fault-tolerant design. He received his PhD in Computer Science from University of California, Irvine in 2008. He is a Senior Member of the IEEE and ACM.

Anthony A. Maciejewski received the BSEE, MS, and PhD degrees from The Ohio State University in 1982, 1984, and 1987. From 1988 to 2001 he was a professor of Electrical and Computer Engineering at Purdue University, West Lafayette. He is currently a Professor and Department Head of Electrical and Computer Engineering at Colorado State University. He is a Fellow of the IEEE. A complete vita is available at: http://www.engr.colostate.edu/∼aam

Howard Jay 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. From 1976 to 2001, he was a professor at Purdue. He is an IEEE Fellow and an ACM Fellow. He received BS degrees from MIT, and the PhD degree from Princeton. Homepage: www.engr.colostate.edu/∼hj.

Gregory A. Koenig is an R&D Staff member at Oak Ridge National Laboratory where his work involves developing scalable system software and tools for ultrascale-class parallel computers. He holds a PhD (2007) and MS (2003) in computer science from the University of Illinois at Urbana-Champaign as well as three BS degrees (mathematics, 1996; electrical engineering technology, 1995; computer science, 1993) from Indiana University-Purdue University Fort Wayne.

Sarah Powers is an R&D staff member at Oak Ridge National Laboratory in the Computer Science and Mathematics Division. Her work integrates techniques from operations research and mathematics to solve problems in a variety of application areas including High Performance Computing. She received her MS and PhD degrees in Operations Research & Statistics from Rensselaer Polytechnic Institute, and BS in Mathematics from Gordon College.

Marcia Hilton is a 1997 graduate of the University of Kentucky with a Bachelor of Science degree in Computer Science. She currently works for the Department of Defense specializing in Application Frameworks and Automatic Processing Systems. She is project technical lead on an Automatic Data Processing System that makes use of commercial schedulers to execute a variety of well defined tasks. She is involved in the implementation of a stable meta-scheduler that will enable a diverse set of customers to intelligently share a common set of resources.

Rajendra Rambharos graduated from the University of Central Florida in 2006 with a BS in Computer Engineering. After graduation, he began work with Harris Corporation, headquartered in Melbourne, Florida, where he gained experience working on the software systems for the FAA telecommunications network for air traffic control. Following Harris, he joined the Department of Defense, working on automated data processing systems in a high availability environment. He is currently a software and systems developer for an intelligent task scheduler architecture.

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 swpoole@gmail.com.