Energy-Aware Resource Management for Computing Systems

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

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

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

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

***Abstract*—This corresponds to the material in the invited keynote presentation by H. J. Siegel, summarizing the research in [1], [2].**

We address the problem of assigning dynamically-arriving tasks to machines in a heterogeneous computing environment. These machines execute a workload composed of different tasks, where the tasks have diverse computational requirements. Each task has a utility function associated with it that represents the value of completing that task, and this utility decreases the longer it takes a task to complete. The goal of our resource manager is to maximize the sum of the utilities earned by all tasks arriving in the system over a given interval of time, while satisfying an energy constraint. We describe example energy-aware resource management methods to accomplish this goal, and compare their performance. We also study the bi-objective problem of maximizing system utility and minimizing the system energy consumption. This analysis technique allows system administrators to investigate the trade-offs between these conflicting goals.

***Index Terms*—high performance computing system; heterogeneous distributed computing; energy-aware computing; resource management; bi-objective optimization**

I. Introduction

This is an overview of the material to be discussed in the invited keynote presentation by H. J. Siegel; it summarizes our research in [1], [2].

We model a heterogeneous compute facility and workload being investigated by the Extreme Scale Systems Center (ESSC) at Oak Ridge National Laboratory (ORNL). Our goal is to study techniques that allow system administrators to analyze the trade-offs between system performance and energy consumption, and to maximize the performance of their computing systems while staying within a specified energy constraint. Each task to be exact has a monotonically decreasing utility function associated with it that represents the utility based on the task's completion time. System performance is measured in terms of total utility earned, which is the sum of all utility earned by tasks that complete within the energy-constraint [3].

In a heterogeneous environment, tasks typically have different execution time and energy consumption characteristics when executed on different machines. We develop a novel energy-aware resource management technique to maximize the utility earned while obeying an energy constraint.

We make the following contributions: (a) the design of a new resource management technique that maximizes utility earned within an energy constraint for an oversubscribed heterogeneous computing system, (b) the design of an energy filtering mechanism that is adaptive to the energy remaining in the system, (c) show how heuristics that only maximize performance can become energy-aware by adapting three previous techniques to use an energy filter, (d) model a bi-objective resource allocation problem between total utility earned and total energy consumed, and (e) analyze the effects of using different seeding heuristics.

II. Problem Description

We assume a workload where tasks arrive dynamically throughout the day and the scheduler maps the tasks to machines for execution. Each task in the system has a utility function associated with it (as described in [3]). Utility functions are monotonically-decreasing functions that represent the utility (or value) of completing the task at different times. We assume the utility functions are given by users or system administrators for any task.

Figure 1 illustrates a sample task time-utility function. By evaluating the function at different completion times, we can determine the amount of utility earned. For example, if a task finished at time 20, it would earn twelve units of utility, whereas if the task finished at time 47, it would only earn seven units of utility.

Our computing system environment consists of a suite of heterogeneous machines, where each machine belongs to a specific machine type. 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

978-1-4799-5173-4/14/$31.00 ©2014 IEEE

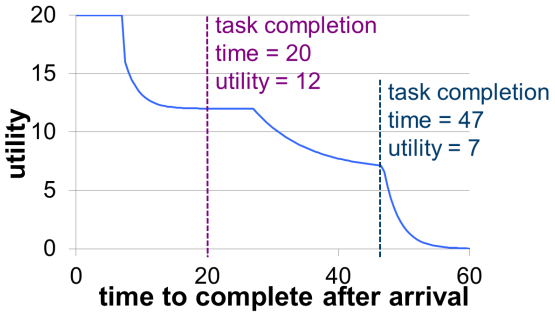

Fig. 1. Task utility function showing values earned at different completion times.

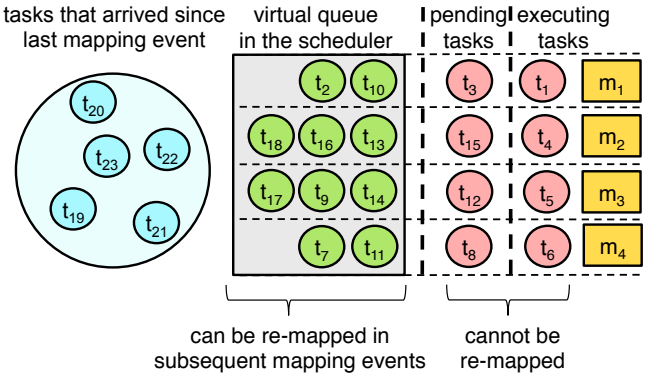

Fig. 2. An example system of four machines showing tasks that are currently executing, waiting in the pending slot, waiting in the virtual queue, and have arrived since the last mapping event (and are currently unmapped).

consumption. We group tasks with similar execution characteristics into task types. We model heterogeneity in our computing system and workload, i.e., machine type A may be faster (or more energy-efficient) than machine type B for certain task types but slower (or less energy-efficient) for others.

For a task of type i on a machine of type j running in P-state k, we assume that we are given (based on historical data or experiments) the Estimated Time to Compute ($ETC(i,j,k)$) and the Average Power Consumption ($APC(i,j,k)$). For the simulation study conducted in this paper, we synthetically create our ETC and APC matrices. We can compute the Estimated Energy Consumption ($EEC(i,j,k)$) by taking the product of its execution time and average power consumption, i.e., $EEC(i,j,k) = ETC(i,j,k) \times APC(i,j,k)$. We model general-purpose machine types and special-purpose machine types [1]. 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.

The goal of the scheduler is to maximize the total utility that can be earned from completing tasks while satisfying an energy constraint for the day.

III. Resource Management

A. Overview

Mapping events occur any time a scheduling decision has to be made. We use batch-mode heuristics that trigger mapping events after one minute time intervals. We assume an oversubscribed environment, so 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.

The task that is next-in-line for execution on a machine is referred to as the pending task. All other tasks that are queued for the machines are said to be in the virtual queues of the scheduler. Figure 2 shows an example system with four machines, the executing tasks, the tasks in the pending slot, and the virtual queues of the scheduler. At a mapping event, the batch-mode heuristics make scheduling decisions for a batch of tasks that are composed of the tasks that have arrived since the last mapping event and the tasks that are currently in the virtual queues. This set of tasks is called the mappable tasks. The batch-mode heuristics are not allowed to remap the pending tasks so that the machines do not idle if the currently executing tasks complete while the heuristic is executing.

B. Batch-mode Heuristics

We present a new heuristic that tries to minimize energy while maximizing the utility earned. The Max-Max Utility-Per-Energy Consumption (Max-Max UPE) is a two-stage heuristic based on the concept of the two-stage Min-Min heuristic that has been widely used in the task scheduling literature (e.g., [4]–[9]). In the first stage, the heuristic independently finds for each task in the batch, the machine and P-state that maximizes "utility earned / energy consumption." If none of the machine-P-state choices for a task satisfy the day's energy constraint nor start the execution of the task within the current day, then, the task is postponed to the next day and removed from the batch. In the second stage, the heuristic picks the task-machine-P-state choice from the first stage that provides the overall highest "utility earned / energy consumption." The heuristic assigns the task to that machine, removes that task from the set of mappable tasks, and repeats this process iteratively until all tasks are either mapped or postponed.

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

The Min-Min Completion Time (Min-Min Comp) heuristic is a fast heuristic adapted from [3] 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.

The Max-Max Utility (Max-Max Util) heuristic introduced in [3] 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 is similar to the Max-Max Util heuristic, but in each stage it maximizes "utility earned / execution time," as opposed to maximizing utility. This heuristic selects assignments that earn the most utility per unit time, which can be beneficial in an oversubscribed system.

C. Dropping Low Utility Earning Tasks

We use the technique to drop tasks with low potential utility at the current time to tolerate the oversubscription. Dropping a task means that it will never be mapped to a machine. 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.

D. Energy Filtering

The goal of our new energy filter technique we have designed is to remove potential allocation choices (task-machine-P-state combination) from a heuristic's consideration if the choice of allocation consumes more energy than an estimated fair-share energy budget. The value of the $task\ budget$ is recomputed at the start of every mapping event and is an estimate of the amount of fair-share energy that we want an execution of a task to consume. The heuristics consider only those task-machine-P-state allocation choices that consume less energy than the $task\ budget$.

We denote $energy\ consumed$ as the total energy that has been consumed by the system in the current day. The $energy\ scheduled$ will account for the energy that will be consumed by all tasks that are currently executing and the tasks that are in the pending slots. The total energy that can be used by heuristics (without violating the day's energy constraint) is denoted by $energy\ remaining$ and is computed as $energy\ constraint_{day} - energy\ consumed - energy\ scheduled$.

The ready time of a machine is set to either the completion time of the executing task, completion time of the pending task, or the current time, whichever is greater. The total time remaining for computations (summed across machines) in the day is denoted as the $aggregate\ time\ remaining$. We compute it by summing across machines the difference between the end time of the day and the ready time of the machine.

The average of the execution time values of all tasks, machines, and P-states is represented as $average\ exec\ time$. The energy filtering technique needs to estimate the total number of tasks that can be executed in the remaining part of the day. It does this by taking the ratio of $aggregate\ time\ remaining$ and $average\ exec\ time$. To adjust the value of the $task\ budget$ around its estimate, we use a multiplier called $energy\ leniency$. This value is determined empirically. The $task\ budget$ that is used to compare the energy consumption of potential allocation choices against is computed using Equation 1.

$$task\ budget = energy\ leniency \times \frac{energy\ remaining}{\left(\frac{aggregate\ time\ remaining}{average\ exec\ time}\right)} \quad (1)$$

IV. Bi-Objective Optimization

In many real world problems, it is important to consider more than one goal or objective. It is often the case that these multiple objectives can directly or indirectly compete with each other. Optimizing for one objective may cause the other objective to be negatively impacted. We try to maximize utility earned while minimizing energy consumed. In general, a well-structured resource allocation that uses more energy will earn more utility and one that uses less energy will earn less utility. For a given resource allocation, the total energy consumed is the sum of the energy consumed by each task to finish executing based on the EEC values.

When a problem contains multiple conflicting objectives, there is a *set* of optimal solutions. This set of optimal solutions may not be known, thus it is important to find a set of solutions that are as close as possible to the optimal set. The set of solutions that we find that best approximates the optimal set is called the set of Pareto optimal solutions [10]. The Pareto optimal set can be used to construct a Pareto front that can illustrate the trade-offs between the two objectives.

We select the eligible solutions by examining solution dominance. Dominance is used to compare two solutions to one another. For one solution to dominate another, it must be better than the other solution in at least one objective, and better than or equal in the other objective.

We adapt a popular multi-objective optimization genetic algorithm, the Nondominated Sorting Genetic Algorithm II (NSGA-II) [11] for our bi-objective problem. The details of our implementation technique, including the design of the chromosomes, crossover and mutation operations, are mentioned in [1].

V. Simulation Setup

A. Energy-Constrained Problem

1) Overview: We average the results of our experiments across 48 simulation trials. Each of the trials represents a new workload of tasks (with different utility functions, task types, and arrival times), and a different compute environment by using new values for the entries in the ETC and APC matrices (but without changing the number of machines). All of the parameters used in our simulations are set to closely match the expectations for future environments of interest to the ESSC.

2) Workload Generation: A method for generating utility functions can be found in [3]. Tasks can start at one of four maximum utility values: 8, 4, 2, or 1. We generate the arrival patterns to closely match expected workloads that are of interest to ESSC [2]. In this environment, general-purpose tasks arrive in a sinusoidal pattern and special-purpose tasks follow a bursty arrival pattern.

3) Execution Time and Power Modeling: In our simulation environment, approximately 50,000 tasks arrive during the duration of a day and each of them belong to one of 100 task types. The compute system that we model has 13 machine types consisting of a total of 100 machines. Among these 13 machine types, four are special-purpose machine types while the remaining are general-purpose machine types. The special-purpose machines execute a subset of task types (which are special with respect to them) 10 times faster than the general-purpose machines. The special-purpose machines do not have the ability to run tasks of other task types. We assume that all machines have three major P-states in which they can operate. Our method used to generate the entries for the EPC and APC matrices are detailed in [2].

4) Obtaining an Energy Constraint: For simulation purposes, we 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 [2]) for a full 24-hour time period, disregarding the energy constraint. Based on these results, we average the total energy consumption throughout the day across 48 simulation trials and use 70% of this average value as our energy constraint.

B. Bi-Objective Problem

1) Datasets and Experiments: The dataset consists of the ETC and APC data based on real performance data. We simulate 4000 tasks arriving within the span of an hour. In [1], we also experiment with two other data sets.

We ran two experiments. For the first experiment, we let the NSGA-II execute for a given number of generations (iterations) and then we analyze the trade-offs between utility earned and energy consumed. The second experiment compares the effect of using different seeds within the initial population on the evolution of the Pareto fronts. The seeding heuristics used for these experiments are described in the section below.

2) Seeding Heuristics: Seeding heuristics provide the NSGA-II with initial solutions that try to intelligently optimize one or both of the objectives. The seeds may help guide the genetic algorithm into better portions of the search space faster than an all random initial population. We implement the following four heuristics: Min Energy, Max Utility, Max Utility-per-Energy, and Min-Min Completion Time. The details os these four seed heuristics are mentioned in [1].

VI. Simulation Results

A. Energy-Constrained Study

1) Overview: All results shown in this section display the average over 48 simulation trials with 95% confidence interval error bars. For comparison purposes, we implement a Random heuristic that randomly maps tasks to machines. All the heuristics (including Random) used a dropping threshold of 0.5 units of utility to tolerate the oversubscription.

2) Results without Energy Filtering: Figure 3 shows the total utility earned by the four heuristics that we examined with and without filtering and compare their performance with that of the Random heuristic. We first discuss the performance of the heuristics without the energy filtering. Random performed the worst because it does not consider the completion time, utility earned, or energy consumed when making assignment decisions. Max-Max UPE earns the highest utility even though it consumes the same amount of energy as the others, i.e., approaches the energy constraint. This is because the heuristic accounts for energy consumption while trying to maximize utility. All the heuristics hit the energy constraint for the day and were not allowed to map any more tasks that may violate the constraint.

3) Results with Energy Filtering: The extent to which energy filtering is performed is controlled by the $energy\ leniency$ term (see Equation 1). 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.

For all heuristics, when we use $energy\ leniency$ values from 0.3 to 0.6, the filtering is so strict that it prevents the heuristic from using all of the available energy that was budgeted for the day resulting in fewer tasks being executed and causing a drop in the total utility earned throughout the day. When using high values of $energy\ leniency$, 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.

Figures 4a and 4b show the utility trace and energy trace for the Max-Max UPT heuristic, 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

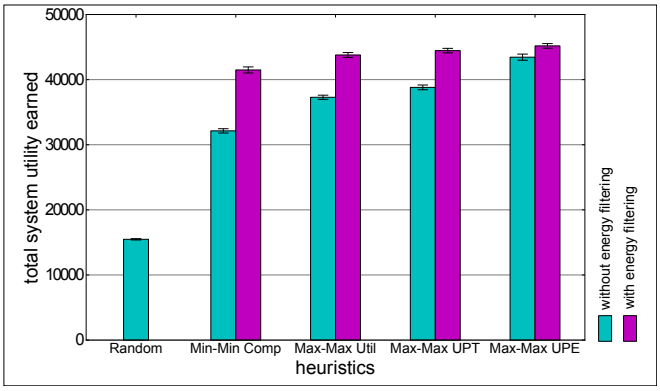

Fig. 3. Total utility earned by the heuristics in the no-filtering case and the filtering case with their best $energy\ leniency$. These are compared to the total utility earned by a Random assignment.

tasks are unable to execute and are dropped from the system earning no utility.

The energy trace charts show the adaptive ability of the filtering technique. 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. With high values of $energy\ leniency$, the filter eventually adapts to lower its value of $task\ budget$. Figure 3 shows the total utility earned by the heuristics in their best $energy\ leniency$ case with filtering and in the no-filtering case compared to the utility earned by Random. Max-Max UPE with energy filtering earns approximately 280% of the utility earned by Random.

The best performance for each of the heuristics comes at an appropriate $energy\ leniency$ that allows the total energy consumption of the heuristic to hit the energy constraint of the day right at the end of the day allowing utility to be earned at a constant rate throughout the entire day. Our energy filtering technique gives this ability to the heuristics.

B. Bi-objective Study

In Figure 5, we show the location of numerous Pareto fronts corresponding to different initial populations. Each of the four subplots show the Pareto fronts through a specific number of NSGA-II iterations. Each marker within the subplots represents a complete resource allocation. Each marker style represents a different population.

Figure 5 shows the Pareto fronts when 4000 tasks scheduled. First, in the top left subplot we see that after 1000 iterations the populations have formed distinct Pareto fronts due to the use of the different seeds within each population. The figure shows the benefit of using the seeds in the initial populations over the all random initial population. This occurs because the random initial population has to rely on only crossover and mutation to find better solutions, whereas the seeded populations have the advantage of a solution that is already trying to make smart resource allocation decisions.

In the 1,000,000 iteration subplot, the region highlighted by the circle represents the solutions that earn the most utility per energy consumed. This circle does not represent the "best" solution in the front, because all solutions along the front are "best" solutions, each representing a different trade-off between utility and energy. To the left of this region, the system can earn relatively larger amounts of utility for relatively small increases in energy. To the right of this region, the system could consume a relatively larger amount of energy but would only see a relatively small increase in utility.

VII. Conclusions

A heuristic's performance in our system is measured in terms of the total utility that it could earn from task completions. We designed an energy-aware heuristic (Max-Max UPE), compared its performance to other heuristics, and integrated an energy filtering technique into our environment. We showed that in an energy-constrained environment our energy-aware heuristic earns more utility than heuristics that only optimize for utility. We also demonstrated that our new energy filtering technique improves the performance of all the heuristics by distributing the consumption of the budgeted energy throughout the day. We showed that by using the NSGA-II we can create well defined Pareto fronts that facilitate a trade-off analysis between the energy consumed and utility earned. We further showed how using different seeding heuristics within the initial populations affected the evolution of the Pareto fronts.

Acknowledgments

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

References

[1] R. Friese, B. Khemka, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, J. Rambharos, G. Okonski, and S. W. Poole, "An analysis framework for investigating the trade-offs between system performance and energy consumption in a heterogeneous computing environments," in *22nd Heterogeneity in Computing Workshop (HCW 2013), in IPDPS 2013 Workshops & PhD Forum*, May 2013, pp. 19–30.

[2] B. Khemka, R. Friese, S. Pasricha, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, R. Rambharos, and S. Poole, "Utility driven dynamic resource management in an oversubscribed energy-constrained heterogeneous system," in *23rd Heterogeneity in Computing Workshop (HCW 2014), in IPDPS 2014 Workshops & PhD Forum*, May 2014, pp. 58–67.

[3] L. D. Briceño, B. Khemka, H. J. Siegel, A. A. Maciejewski, C. Groer, G. Koenig, G. Okonski, and S. Poole, "Time utility functions for modeling and evaluating resource allocations in a heterogeneous computing systems," in *20th Heterogeneity in Computing Workshop (HCW 2011), in IPDPS 2011 Workshops & PhD Forum*, May 2011, pp. 7–19.

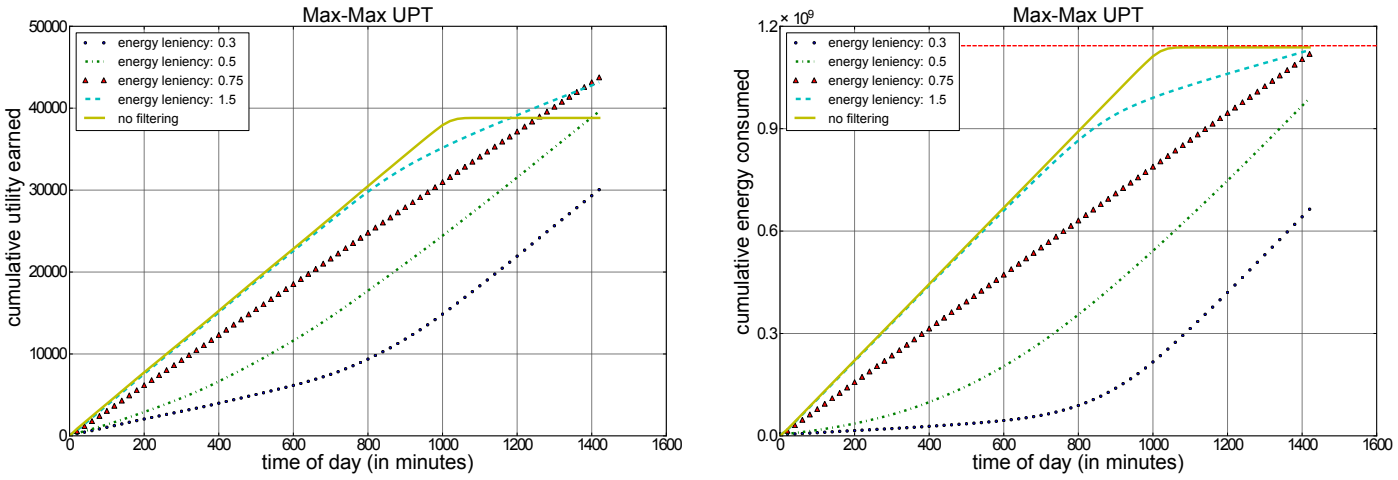

Fig. 4. Trace of (a) the cumulative utility earned and (b) the cumulative energy consumed throughout the day at 20 minute intervals as the *energy leniency* is varied for the Max-Max UPT heuristic. The results are averaged over the 48 trials with confidence intervals being extremely tight.

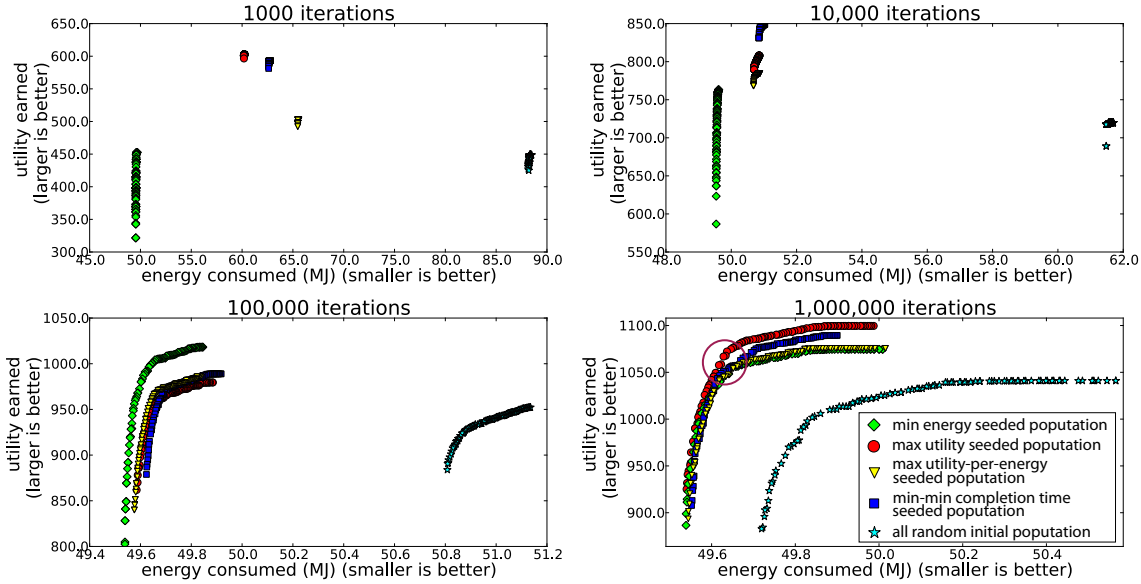

Fig. 5. Pareto fronts of total energy consumed versus total utility earned for different initial seeded populations through various number of iterations. The circled region represents the solutions that earn the most utility per energy spent. Both the y-axis and x-axis values are specific to each subplot.

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

[5] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *J. of the ACM*, vol. 24, no. 2, pp. 280–289, Apr. 1977.

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

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

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

[9] M. Wu and W. Shu, "Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems," in *Int'l. Heterogeneity in Computing Workshop (HCW 2000), in IPDPS 2000*, Mar. 2000, pp. 375–385.

[10] V. Pareto, *Cours d'economie politique*. Lausanne: F. Rouge, 1896.

[11] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.