2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum

An Analysis Framework for Investigating the Trade-offs Between System Performance and Energy Consumption in a Heterogeneous Computing Environment

Ryan Friese*, Bhavesh Khemka*, Anthony A. Maciejewski*, Howard Jay Siegel*†,
Gregory A. Koenig‡, Sarah Powers‡, Marcia Hilton§, Jendra Rambharos§, Gene Okonski§,and Stephen W. Poole‡§
*Department of Electrical and Computer Engineering
†Department of Computer Science
Colorado State University
Fort Collins, CO, 80523
‡Oak Ridge National Laboratory
Oak Ridge, TN 37830
§Department of Defense
Washington, DC 20001
Email: Ryan.Friese@rams.colostate.edu, {Bhavesh.Khemka, HJ, aam}@colostate.edu,
{koenig, powersss, SPoole}@ornl.gov, {mmskizig, okonskitg}@verizon.net, jendra.rambharos@gmail.com

***Abstract*—Rising costs of energy consumption and an ongoing effort for increases in computing performance are leading to a significant need for energy-efficient computing. Before systems such as supercomputers, servers, and datacenters can begin operating in an energy-efficient manner, the energy consumption and performance characteristics of the system must be analyzed. In this paper, we provide an analysis framework that will allow a system administrator to investigate the trade-offs between system energy consumption and utility earned by a system (as a measure of system performance). We model these trade-offs as a bi-objective resource allocation problem. We use a popular multi-objective genetic algorithm to construct Pareto fronts to illustrate how different resource allocations can cause a system to consume significantly different amounts of energy and earn different amounts of utility. We demonstrate our analysis framework using real data collected from online benchmarks, and further provide a method to create larger data sets that exhibit similar heterogeneity characteristics to real data sets. This analysis framework can provide system administrators with insight to make intelligent scheduling decisions based on the energy and utility needs of their systems.**

***Keywords*-bi-objective optimization; energy-aware computing; heterogeneous computing; resource allocation; data creation**

I. Introduction

During the past decade,large datacenters (comprised of supercomputers, servers, clusters, farms, storage, etc.) have become increasingly powerful. As a result of this increased performance the amount of energy needed to operate these systems has also grown. It was estimated that between the years 2000 and 2006 the amount of energy consumed by high performance computing (HPC) systems more than doubled [1]. In 2006 an estimated 61 billon kWh was consumed by servers and datacenters, approximately equal to 1.5% of the total United States energy consumption for that year. This amounted to $4.5 billion in electricity costs [1]. Since 2005, the total amount of electricity used by HPC systems increased by another 56% worldwide and 36% in the U.S. Additionally, during 2010, global HPC systems accounted for 1.5% of total electricity use, while in the U.S., HPC systems accounted for 2.2% [2].

With the cost of energy and the need for greater performance rising, it is becoming increasingly important for HPC systems to operate in an energy-efficient manner. One way to reduce the cost of energy is to minimize the amount of energy consumed by a specific system. In this work, we show that we can reduce the amount of energy consumed by a system by making intelligent scheduling decisions. Unfortunately, consuming less energy often leads to a decrease in the performance of the system [3]. Thus, it can be useful to examine the trade-offs between minimizing energy consumption and maximizing computing performance for different resource allocations. Current resource managers, such as MOAB, cannot reasonably determine the trade-offs between performance and energy.

In this research, we model a computing environment and corresponding workload that is being investigated by the Extreme Scale Systems Center (ESSC) at Oak Ridge National Laboratory (ORNL). The ESSC is a joint venture between the United States Department of Defense (DoD) and Department of Energy (DOE) to provide research, tools, software, and technologies that can be utilitized in both DoD and DOE environments. Our goal is to design an analysis framework that a system administrator can use to analyze the energy and performance trade-offs of a system by using different resource allocations (i.e. mapping of tasks to machines). System performance is measured in terms of total utility earned, where each task in the workload is assigned a time utility function that monotonically decreases in value the longer a task remains in the system [4]. The computing

978-0-7695-4979-8/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPSW.2013.142

environment is a heterogeneous mix of machines and tasks that represents an environment with system characteristics and workload parameters that are based on the expectations of future DoD and DOE environments.

In a heterogeneous environment, tasks may have different execution and power consumption characteristics when executed on different machines. One can change the performance and energy consumption of the system by using different resource allocations. A resource allocation is defined to be a complete mapping of tasks to machines, where we assume the number of tasks is much greater than the number of machines. To create these resource allocations, we model this scheduling problem as a bi-objective optimization problem that maximizes utility earned and minimizes energy consumed. We adapt the Nondominated Sorting Genetic Algorithm II (NSGA-II) [5] to create the resource allocations.

To analyze the effect of different resource allocations, we implement our method in a static and offline environment. To create the offline environment we simulate a trace over a specified time period of the modeled environment. This allows us to gather information about the number of tasks that arrived during this time period, as well as the arrival times and types of tasks that were present in the system. The availability of this information makes this a static resource allocation problem. The knowledge gained from studies such as this can be used to set the parameters needed for designing dynamic or online allocation heuristics.

We utilize execution and power consumption characteristics from real machines and applications. We further use this real characteristic data to create a larger data set in order to simulate larger systems. In all cases, our results show that by using different resource allocations a system administrator can greatly change the amount of utility earned and power consumed based on the needs of their system.

In this paper we make the following contributions:

1) Modeling a bi-objective resource allocation problem between total utility earned and total energy consumed to address concerns about energy-efficient computing, specifically for environments of DoD and DOE interest.
2) Creating and evaluating many intelligent resource allocations to show that the utility earned and energy consumed by the system can change greatly.
3) Demonstrating our method by using real machine and task data.
4) Providing a method to create synthetic data sets that preserve the heterogeneity measures found from real data sets.
5) Analyzing the effect of using different seeding heuristics on the evolution of solutions found by the genetic algorithm.

The remainder of the paper is organized as follows. Related work is discussed in Section II. We construct the system model, and our real data set, in Section III. In Section IV, we describe our bi-objective optimization problem and the NSGA-II. Our simulation setup is detailed in Section V. Section VI analyzes our simulation results. Finally, our conclusion and future work will be given in Section VII.

II. Related Work

The authors of [3] formulate a bi-objective resource allocation problem to analyze the trade-offs between makespan and energy consumption. Their approach is concerned with minimizing makespan as their measure of system performance, opposed to maximizing utility in our approach. Additionally, they model an environment where the workload is a bag of tasks, not a trace from a dynamic system. This is important because they do not consider arrival times or the specific ordering of tasks on machine. Finally, the authors do not demonstrate their approach using real historical data.

A bi-objective optimization problem between makespan and reliability is used to solve heterogeneous task scheduling problems in [6] and [7]. We perform the bi-objective optimization between utility earned and energy consumed.

The study in [8] implements a weighted sum simulated annealing heuristic to solve a bi-objective optimization problem between makespan and robustness. One run of this heuristic produces a single solution, and different weights can be used to produce different solutions. This differs from our approach in that we independently evaluate our two objective functions that allows us to create a Pareto front containing multiple solutions with one run of our algorithm.

In [9], the authors minimize makespan and total tardiness to solve the bi-objective flowshop scheduling problem. The authors utilize a Pareto-ant colony optimization approach to create their solutions. This work differs from ours by not using a genetic algorithm nor is it concerned with utility nor energy consumption.

In [10], a job-shop scheduling problem is modeled as a bi-objective optimization problem between makespan and energy consumption. The authors model a homogeneous set of machines whereas our work models a heterogeneous set of machines. The work in [10] also differs from ours by using an algorithm that only produces a single solution.

Heterogeneous task scheduling in an energy-constrained computing environment is examined in [11]. The authors model an environment where devices in an ad-hoc wireless network are limited by battery capacity. The heuristics in [11] create a single solution while ours creates multiple solutions. In our study, we are not concerned with an energy-constrained system, but instead we try to minimize the total energy consumed.

Minimizing energy while meeting a makespan robustness constraint in a static resource allocation environment is studied in [12]. This work is not an explicit bi-objective optimization. Our work differs by using utility maximization as an objective instead of minimizing makespan.

A dynamic resource allocation problem in an energy-constrained environment is studied in [13]. Solutions to this problem must complete as many tasks as they can while staying within the energy budget of the system. In our work, we model a trace of a dynamic system allowing us to create a static allocation. We also do not constrain the amount of energy our system can consume.

Mapping tasks to computing resources is also an issue in hardware/software co-design [14]. This problem domain differs from ours, however, because it typically considers the hardware design of a single chip. Our work assumes a given collection of heterogeneous machines.

III. System Model

A. Overview

Our system environment is modeled based on the needs of the ESSC at ORNL. The system is intended to provide a model of both the machines and workload used in such an environment. This system model has been designed with detailed collaboration between members of Colorado State University, ORNL, and the United States DoD to ensure it accurately captures the needs and characteristics of the ESSC.

B. Machines

This model consists of a suite of M heterogeneous machines, where each machine in this set belongs to a specific machine type μ. Machine types exhibit heterogeneous performance, that is machine type A may be faster than machine type B for some task types but slower for others [15]. Machines types also exhibit heterogeneous energy consumption and belong to one of two categories. The first category is general-purpose machines. General-purpose machines are machines that are able to execute any of the task types in the system, and they make up the majority of the machines in the environment. The other category of machines is special-purpose machines. Machines within this category can only execute a small subset of task types and are incapable of executing the remaining task types. Special-purpose machines generally exhibit a 10x decrease in the execution times of the task types they can execute compared to the general-purpose machines. The heterogeneity between the machine types can be attributed to differences in micro-architectures, memory modules, hard disks, and/or other system components.

C. Workload

In the ESSC environment, tasks arrive dynamically throughout the day. Once a task arrives, the utility earned by a task may start to decay, see Subsection IV-B1. Utility dictates how much useful work a given task can accomplish. Utility is represented by a monotonically-decreasing function with respect to time. Therefore the sooner a task completes execution the higher utility it might earn [4]. Because we are performing a bi-objective analysis of the system, we consider a trace of tasks that arrive into the system within a specified amount of time (e.g., one hour). The arrival times of each task in the trace must be recorded to accurately calculate how much utility a given task earns.

Every task in the trace is a member of a given task type. Each task type has unique performance and energy consumption characteristics for executing on the machine types. Similar to the machine types, task types belong to one of two categories; general-purpose tasks types and special-purpose tasks types. General-purpose tasks types are task types that can only execute on the general-purpose machine types. A special-purpose task type can execute on a specific special-purpose machine type at an increased rate of execution (compared to the general-purpose machine type), and it is also able to execute on the general-purpose machine types.

D. Execution and Energy Consumption Characteristics

It is common in resource allocation research to assume the availability of information about the performance characteristics of the tasks types and machine types present in a system (e.g., [16, 17, 18, 19]). This information is contained within an Estimated Time to Compute (ETC) matrix. An entry in this matrix, ETC(τ,μ), represents the estimated time a task of type τ will take to execute on a machine of type μ. The values contained within this matrix can be synthetically created to model various heterogeneous systems [15], or the values can be obtained from historical data (e.g., [18, 17]). The assumption that ETC values can be obtained from historical data is valid for the intended ESSC environments.

In this study, we also require information about the power consumption characteristics of the task types and machine types. We call this set of data the Estimated Power Consumption (EPC) matrix. An entry in this matrix, EPC(τ,μ), represents the average amount of power a task of type τ consumes when executing on a machine of type μ. Different EPC values can represent different task type energy characteristics, e.g., computationally intensive tasks, memory intensive tasks, or I/O intensive tasks. The values within the EPC matrix can also be created synthetically or gathered from historical data. In this study we utilize ETC and EPC matrices that contain both real historical data (described in Section III-D1), and synthetic data derived from the historical data (described in Section III-D2).

1) Gathering of Historical Data: To accurately model the relationships between machine performance and energy consumption in a heterogeneous suite of machines, we first create ETC and EPC matrices filled with historical data. For this data, we use a set of online benchmarks [20] that tested a suite of nine machines (Table I) over a set of five tasks (Table II). The machines differ by the CPU and motherboard/chipset used, but all the machines use the

Table I
MACHINES (DESIGNATED BY CPU) USED IN BENCHMARK

AMD A8-3870k
AMD FX-8159
Intel Core i3 2120
Intel Core i5 2400S
Intel Core i5 2500K
Intel Core i7 3960X
Intel Core i7 3960X @ 4.2 GHz
Intel Core i7 3770K
Intel Core i7 3770K @ 4.3 GHz

Table II
PROGRAMS USED IN BENCHMARK

C-Ray
7-Zip Compression
Warsow
Unigine Heaven
Timed Linux Kernel Compilation

same amount of memory (16GB) and the same type of hard drive and GPU. For each task, the benchmark produced the average execution time for that task on each of the machines as well as the average power consumed by that task on each of the machines. We were able to place these values into our ETC and EPC matrices. Each of the nine machines from the benchmark represents a specific machine type, and each of the five tasks from the benchmark represents a specific task type. This data provides us with initial ETC and EPC matrices of size 5×9.

2) Creation of Synthetic Data: From the historical data, we are able to derive a larger data set to study how our algorithm performs for a more complex problem. To create a larger data set, we need to increase the number of task types as well as introduce special-purpose machines. The real historical data will be included in this larger data set. For the new synthetic data set to resemble the real historical data as closely as possible, we need to preserve the heterogeneity characteristics of the historical data set. Heterogeneity characteristics of a data set can be measured using various standard statistical measures, such as the coefficient of variation, skewness, and kurtosis [21]. Two data sets that have similar heterogeneity characteristics would have similar values for these three measures.

To create a larger data set (we describe the process using the ETC matrix, but the process is identical for the EPC matrix), our first task is to create more task types. To do this, we first calculate the average execution time of each of the real task types across all the machines, this is also known as the row average of a task type. We then use these row average task execution times as the basis for creating new task types. We calculate the following heterogeneity measures: mean, variation, skewness, and kurtosis (mksv) for the collection of row average task execution times. With the mvsk values we use the Gram-Charlier expansion [22] to create a probability density function (PDF) that produces samples of row average task execution times. By sampling this PDF we can create the row average task execution times for any number of new task types.

Once a desired number of task types are created, the next step is to populate the ETC entries for these new task types. While doing this, we want to preserve the relative performance from one machine to another. We calculate the task type execution time ratio. This ratio is the execution time of a specific task type on a specific machine, divided by the average task execution time (across all machines) for that task type. For example, let us assume task type 1 takes eight minutes to execute on machine type A, but it takes twelve minutes to execute on machine type B. Also assume task type 1 has an average execution time of ten minutes across all machines. On machine type A the task type has a task type execution time ratio of .8 while on machine type B, the task type has a task type execution time ratio 1.2. We calculate this ratio for all of the real task types on all of the real machines. Typically, faster machines will have values less than one for this ratio while slower machines have values greater than one.

Having found the relative performance of the machines to one another, the next step is to capture the task heterogeneities across the individual machines to create the new ETC matrix values. The following procedure is performed individually for each machine. On a given machine, we calculate the heterogeneity values of the task type execution time ratios for the real task types. By using the mvsk values produced, we then create a PDF that produces samples of task type execution time ratios for that specific machine. We sample this PDF to create task type execution time ratios for the new task types on that specific machine. We can then take the task type execution time ratios for the new task types and multiply them by their respective average task type execution time values to produce the actual ETC values for the new task types.

The final step is to now create the special-purpose machines types. Based on the expectations of future DoD and DOE environments, special-purpose machine types are modeled to perform around 10x faster than the general-purpose machine types for a small number of task types (two to three for each special purpose machine type). To create these machines we first select the number of task types that can be executed on the special-purpose machines types. Then the average execution time (across all the machines) for each of these selected task types is found. The average execution time for each task type is divided by ten and is then used as the ETC value for the special-purpose machine type. When calculating EPC values, the average power consumption across the machines is *not* divided by ten.

This method allows us to create larger data sets that exhibit similar heterogeneity characteristics when compared to the real data. This method can be used to create both ETC

and EPC matrices, with the exception of special-purpose EPC values as stated above.

IV. Bi-Objective Optimization

A. Overview

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. In this work we try to maximize utility earned while minimizing energy consumed. We will show in our results that these two objectives do compete with each other. 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. This relationship occurs because we model utility as a monotonically-decreasing performance measure, that is, the longer a task takes to execute, the less utility it may earn. In [3], it is shown that spending more energy may allow a system to complete all the tasks within a batch sooner.

In Section IV-B, we define the two objective functions we are optimizing. Section IV-C describes which solutions should be considered when solving a bi-objective optimization. The genetic algorithm used for this study is briefly explained in Section IV-D.

B. Objective Functions

1) Maximizing Utility: One objective is maximizing total system utility earned. Total Utility Earned is a metric used to measure the performance of a computing system. Utility can be thought of as the amount of useful work a system accomplishes [4]. The amount of utility a task earns is determined by the completion time of the task, as well as by three sets of parameters; priority, urgency, and utility characteristic class. These parameters form what is called a time-utility function. This function is a monotonically-decreasing function [4].

Priority represents the total amount of utility a task could possibly earn, i.e., how important a task is. Urgency represents the rate of decay of utility a task may earn as a function of completion time. Utility characteristic class allows the utility function to be separated into discrete intervals. Each interval can have a beginning and ending percentage of maximum priority, as well as an urgency modifier to control the rate of decay of utility. The definition of these parameters are based on the needs of the ESSC. The value of these parameters in an actual system are determined by system administrators (not individual users) and are policy decisions that can be adjusted as needed.

Figure 1 illustrates a sample task time-utility function. We see that the function is monotonically decreasing. Additionally, we see the different intervals existing in the 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.

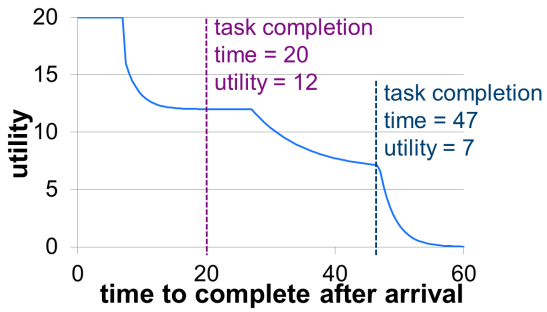

Figure 1. Task time-utility function showing values earned at different completion times.

Every task t in the system is assigned its own utility function $\Upsilon(t)$. This function returns the utility earned by that task when it completes execution. Tasks that have hard deadlines can be modeled as having their utility decay to zero by a specific time. To optimize for total utility, the goal is to maximize the sum of utilities earned over all tasks in the system. Given that there are T tasks in the system, total utility earned, denoted U, can be defined as

$$U = \sum_{\forall t \in T} \Upsilon(t). \tag{1}$$

2) Minimizing Energy Consumed: The other objective we optimize for is minimizing total energy consumed. For a given resource allocation, the total energy consumed is the sum of the energy consumed by each task to finish executing. We first calculate the Expected Energy Consumption (EEC) for a given task on a given machine. This is found by multiplying the ETC and EPC values for that task on that machine. Assume the function $\Phi(t)$ returns the task type of a given task t and the function $\Omega(m)$ returns the machine type of a given machine. The expected energy consumption of task t on machine m can then be given as

$$EEC[\Phi(t), \Omega(m)] = ETC[\Phi(t), \Omega(m)] \times EPC[\Phi(t), \Omega(m)]. \tag{2}$$

Let T_m be the tasks that execute on machine m, where task $t_m \in T_m$. The total energy consumed by the system, denoted E, is found by

$$E = \sum_{\forall m \in M} \sum_{\forall t_m \in T_m} EEC[\Phi(t_m), \Omega(m)]. \tag{3}$$

C. Generating Solutions for a Bi-Objective Optimization Problem

When a problem contains multiple objectives, it is challenging to determine a single global optimal solution. Often, there is instead a *set* of optimal solutions. This set of optimal

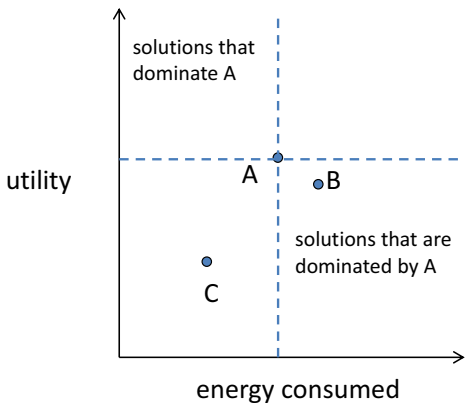

Figure 2. Illustration of solution dominance for three solutions: A, B, and C. Solution A dominates solution B because A has lower energy consumption as well as it earns more utility. Neither solution A nor C dominate each other because C uses less energy, while A earns more utility.

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 [23]. The Pareto optimal set can be used to construct a Pareto front that can illustrate the trade-offs between the two objectives.

Not all solutions we find can exist within the Pareto optimal set. 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. Figure 2 illustrates the notion of solution dominance. In this figure we show three potential solutions: A, B, and C. The objectives we are optimizing for are minimizing total energy consumption along the x-axis and maximizing total utility earned along the y axis. A "good" solution would be one that consumes small amounts of energy and earns large amounts of utility (upper left corner).

If we consider solutions A and B, we can say that B is dominated by A. This is because A uses less energy as well as earns more utility than B. The same is true for any solution located within the lower right region. Now, when we examine solutions A and C we cannot make any claims on the dominance of one solution over the other. This is because C uses less energy than A, but C does not earn as much utility, thus the two solutions can not be directly compared to each other, which means they are both located on the Pareto front. Finally, solution A will only be dominated by solutions that exist in the upper left region.

D. Nondominated Sorting Genetic Algorithm II

The Nondominated Sorting Genetic Algorithm II (NSGA-II) was designed for solving bi-objective optimization problems [5]. We adapt a version of the NSGA-II that we used in [3] for a different, simpler resource allocation problem. We will briefly describe the basic NSGA-II algorithm and how we have modified it for our specific problem.

The NSGA-II is a popular multi-objective genetic algorithm that utilizes solution dominance to allow the populations of chromosomes to evolve into better solutions over time. For our problem domain, a single population represents a set of possible resource allocations (solutions). To use the NSGA-II for our problem we needed improve upon [3] and encode the algorithm so that it could solve bi-objective resource allocation problems where the two objectives are maximizing utility earned and minimizing energy consumed. To do this, the algorithm must be able to accurately track and calculate the individual utility earned by each task. This meant we needed to create our own genes, chromosomes, crossover operator, and mutation operator.

Genes represent the basic data structure of the genetic algorithm. For our problem, a gene represents a task. Each gene contains: the machine the gene will operate on, the arrival time of the task, and the global scheduling order of the task. The global scheduling order is a number from 1 to the number of tasks in the chromosome and it controls the order that tasks execute on the individual machines. The lower the number, the sooner the task will execute. The global scheduling order is independent of the task arrival times.

A chromosome represents a complete solution, i.e., resource allocation. Each chromosome is comprised of T genes, where T is the number of tasks that are present in the trace being studied. The i^{th} gene in every chromosome corresponds to the same task, in particular, the i^{th} task ordered based on task arrival times. To examine dominance relationships amongst the chromosomes in the population, each chromosome is individually evaluated with respect to utility earned and energy consumed. Because the global scheduling order is independent of task arrival times, we must ensure that any task's start time is greater than or equal to its arrival time. If this is not the case, the machine sits idle until this condition is met.

For populations and chromosomes to evolve from one generation to the next, the following crossover and mutation operations were implemented. For crossover, we first select two chromosomes uniformly at random from the population. Next, the indices of two genes are selected uniformly at random from within the chromosomes. Finally, we swap all the genes between these two indices, from one chromosome to the other. In this operation, the machines the tasks execute on, and the global scheduling orders of the tasks are all swapped. The goal of this operation is to allow chromosomes

making good scheduling decisions to pass on favorable traits to other chromosomes.

For the mutation operation, we randomly select a chromosome from the population. We then select a random gene from within that chromosome. We then mutate the gene by selecting a random machine that that task can execute on. Additionally, we select another random gene within the chromosomes and then swap the global scheduling order between the two genes.

Within a given population, a solution is ranked based on how many other solutions dominate it. If no other solutions dominate a solution, it is said to be nondominated and is given a rank of 1. A solution's rank can be found by taking *1 + the number of solutions that dominate it*. All solutions of rank 1 within a given population represent the current Pareto optimal set. The process of sorting the solutions is performed using the nondominating sorting algorithm, and is one step of the NSGA-II algorithm. A basic outline of the NSGA-II is taken from [3] and shown in Algorithm 1.

Algorithm 1 NSGA-II algorithm from [3]

1: create initial population of N chromosomes
2: **while** termination criterion is not met **do**
3: create offspring population of size N
4: perform crossover operation
5: perform mutation operation
6: combine offspring and parent populations into a single meta-population of size $2N$
7: sort solutions in meta-population using nondominated sorting algorithm
8: take all of the rank 1, rank 2, etc. solutions until we have at least N solutions to be used in the parent population for the next generation
9: **if** more than N solutions **then**
10: for solutions from the highest rank number used, take a subset based on crowding distance [5]
11: **end if**
12: **end while**
13: the final population is the Pareto front used to show the trade-offs between the two objectives

An important step to understand in the algorithm is the creation of offspring populations (step 3). We start with a parent population of size *N*. We then perform $N/2$ crossover operations (each crossover operation produces two offspring) to produce an initial offspring population also of size *N*. Next, the mutation operation is then performed with a probability (selected by experimentation) on each offspring within the population. If a mutation occurs, only the mutated offspring remains in the population.

We can now combine the parent and offspring population into a single meta-population of size 2*N* (step 6). This combining of populations allows elitism to be present within the algorithm, that is the algorithm keeps the best chromosomes from the previous generation and allows them to be evaluated in the current generation. From this new meta-population, we then select the next generation's parent population (steps 7, 8, 9, and 10). To create the new parent population, we need to select the best *N* chromosomes from the meta-population.

To illustrate this procedure, we provide the following example. Assume we currently have a parent population and spring population, each with 100 chromosomes, which combines to make a meta-population with 200 chromosomes. We then perform the nondominated sorting algorithm from step 7. We find that we have 60 chromosomes of rank 1, 30 chromosomes of rank 2, 20 chromosomes of rank 3, and 90 chromosomes that have a rank higher than 3. We need to create a new parent population with only 100 chromosomes in it. First we take the 60 chromosomes that are rank 1 and place them into the new population. This leaves room for 40 more chromosomes. We then place the rank 2 chromosomes in the new population. There is space in the population for 10 additional chromosomes. To select only 10 out of the 20 rank 3 chromosomes we use the method of crowding distance [5] to arrive at our full 100 chromosome population. Crowding distance is a metric that penalizes chromosomes that are densely packed together, and rewards chromosomes that are in remote sections of the solutions space. This operation creates a more equally spaced Pareto front.

V. Simulation Setup

A. Datasets and Experiments

To illustrate how our analysis framework allows system administrators to analyze the trade-offs between utility earned and energy consumed of a given system, we have conducted numerous simulations using three different sets of data.

The first set consists of the real historical ETC and EPC data gathered from the online benchmarks (nine machine types and five task types) [20]. This set only allotted one machine to each machine type and simulated 250 tasks comprised of the five task types arriving over a period of 15 minutes. We are performing a post-mortem static resource allocation as we are using a trace of a (simulated) system, thus the arrival times of all tasks are known *a priori*. This real data set is used as the basis for the second and third data sets, as described is Subsection III-D2.

The second and third sets consist of the ETC and EPC data manufactured from the real data (Subsection III-D2). For these sets, we created four special-purpose machine types for a total of 13 machine types and 25 additional task types for a total of 30 task types. Additionally, for both sets there were 30 machines across the 13 machine types. The break up of those machines can be seen in Table III. Data sets 2 and 3 differ from one another by the number of tasks each

Table III
BREAKUP OF MACHINES TO MACHINE TYPES

machine type	number of machines
Special-purpose machine A	1
Special-purpose machine B	1
Special-purpose machine C	1
Special-purpose machine D	1
AMD A8-3870k	2
AMD FX-8159	3
Intel Core i3 2120	3
Intel Core i5 2400S	3
Intel Core i5 2500K	2
Intel Core i7 3960X	4
Intel Core i7 3960X @ 4.2 GHz	2
Intel Core i7 3770K	5
Intel Core i7 3770K @ 4.3 GHz	2

set simulates. Set 2 simulates 1000 tasks arriving within the span of 15 minutes, while set 3 simulates 4000 tasks arriving within the span of an hour.

We conducted experiments on each data set. For the first group of experiments, we let the genetic algorithm execute for a given number of generations and then we analyze the trade-offs between utility earned and energy consumed. The second group of experiments compare 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.

B. Seeding Heuristics

Seeding heuristics provide the genetic algorithm 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 execution times of the greedy heuristics are negligible compared to the NSGA-II, making solutions found by these heuristics plausible to use. To use a seed within a population, we generate a new chromosome from one of the following heuristics. We place this chromosome into the population and create the rest of the chromosomes for that population randomly.

1) Min Energy: Min Energy is a single stage greedy heuristic that maps tasks to machines that minimize energy consumption. This heuristic maps tasks according to their arrival time. For each task the heuristic maps it to the machine that consumes the least amount of energy to execute the task. This heuristic will create a solution with the minimum possible energy consumption. Solutions may exist that consume the same amount of energy while earning more utility.

2) Max Utility: Max Utility is also a single stage greedy heuristic similar to the min energy heuristic except that it maps tasks to the machines that maximizes utility earned [4]. This heuristic must consider the completion time of the machine queues when making mapping decisions. There is no guarantee this heuristic will create a solution with the maximum obtainable utility.

3) Max Utility-per-Energy: Max Utility-per-Energy tries to combine aspects of the previous two heuristics. Instead of making mapping decisions based on either energy consumption or utility earned independently, this heuristic maps a given task to the machine that will provide the most utility earned per unit of energy consumed.

4) Min-Min Completion Time: Min-min completion time is a two-stage greedy heuristic that maps tasks to the machines that provide the minimum completion time [24, 25, 26]. During the first stage, the heuristic finds for every task the machine that minimizes that task's completion time. In the second stage, the heuristic selects from among all the task-machine pairs (from the first stage) the pair that provides the overall minimum completion time. That task is then mapped to that machine, and the heuristic repeats this operation until there are no more tasks to map.

VI. RESULTS

In Figures 3, 4, and 6 we show the location of numerous Pareto fronts. In each figure we show different fronts corresponding to different initial populations, for each of our three data sets respectively. Each of the four subplots show the Pareto fronts through a specific number of NSGA-II iterations. The x-axis is the number of megajoules consumed by the system, and the y-axis is the amount of utility earned by the system. Each marker within the subplots represents a complete resource allocation. Each marker style represents a different population. The diamond marker represents the population that contained the "Min Energy" seed, the square marker represents the population with the "Min-Min completion time" seed, the circle marker represents the population with the "Max Utility" seed, the triangle maker represents the population with the "Max Utility-per-Energy" seed, and finally the star marker represents the population with a completely random initial population. We also considered an initial population that contained all four of the seeding heuristics, but we found that this population performed similarly to the min-energy seeded population, and thus did not include it in our results.

Figure 3 shows the Pareto fronts for the real historical data set. For this data set we evaluated the evolution of the Pareto fronts through four different number of iterations; 100, 1000, 10,000, and 100,000 iterations. First, in the top left subplot we see that after 100 iterations the populations have formed distinct Pareto fronts covering various parts of the solution space. This occurs because of the use of the different seeds within each population. The presence of the seed within a population allows that population to initially explore the solution space close to where the seed originated. As the number of iterations increase though, the presence of the

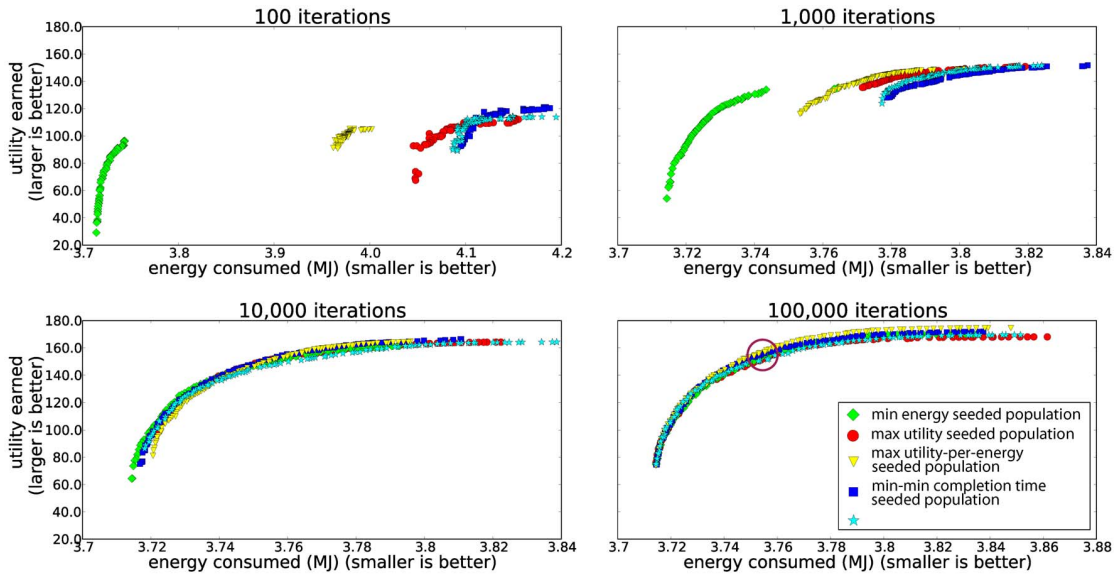

Figure 3. Pareto fronts of total energy consumed vs. total utility earned for the real historical data set (data set 1) for different initial seeded populations through various number of iterations. The circled region represents the solutions that earn the most utility per energy spent. The y-axis values are shared across subplots and while the x-axis values are specific to each subplot.

seed starts to become irrelevant because all the populations, even the all random initial population, start converging to very similar Pareto fronts.

In the 100,000 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. The system administrator may not have energy to reach the circled solution, or may be willing to invest more energy for more utility. 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.

A system administrator can use this bi-objective optimization approach to analyze the utility-energy trade-offs for any system of interest, and then set parameters, such as energy constraints, according to the needs of that system. These energy constraints could then be used in conjunction with a separate online dynamic utility maximization heuristics.

The Pareto fronts in Figure 4 illustrate the trade-offs between total energy consumed and total utility earned for the data set that contains 1000 tasks. These fronts were evaluated at 1000, 10,000, 100,000, and 1,000,000 iterations. We increased the number of iterations executed for this data set compared to the real historical data set because we are simulating a larger and more difficult problem.

Examining the top two subplots, we see the effect the initial seeds have on each of the populations. This provides the chance for system administrators to make some interesting observations about their systems. For example, the "min energy" population typically finds solutions that perform better with respect to energy consumption, while the "min-min completion time" population typically finds solutions that perform better with respect to utility earned. This analysis could provide insight into what type of dynamic heuristics could be used to either maximize utility or minimize energy depending on the needs of a given system. These subplots also show that using smart resource allocation seeds can produce better solutions in a limited number of iterations

In the lower two subplots, we see that all the populations have started to converge towards the same Pareto front. This allows us to find the region containing the solutions that earn the most utility per energy and can provide system administrators with valuable information about the trade-offs between the total energy consumed and the total utility earned of their specific systems. Figure 5 illustrates how the maximum utility per energy region is found. Subplot 5.A shows the final Pareto front for the "max utility-per-energy" seeded population. Subplot 5.B shows a plot of utility earned per energy spent vs. utility while Subplot 5.C shows a plot of utility earned per energy spent vs. energy. By locating the "peaks" in both these plots we can find the utility and energy values, respectively, for which utility earned per energy spent is maximized. We can then translate these values onto the Pareto front to find where this region is located. This is shown using the solid lines for utility and the dashed lines for energy.

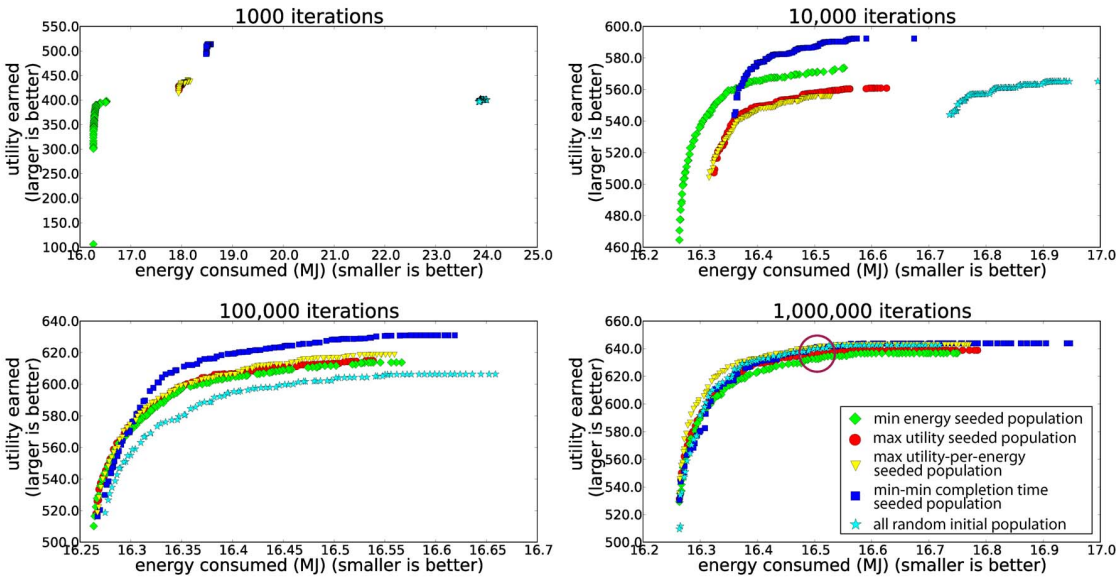

Figure 4. Pareto fronts of total energy consumed vs. total utility earned for the data set containing 1000 tasks (data set 2) for different initial seeded populations through various number of iterations. The circled region represents the solutions that earn the most utility per energy spent. Both the y-axis and x-axis values are specific to each subplot.

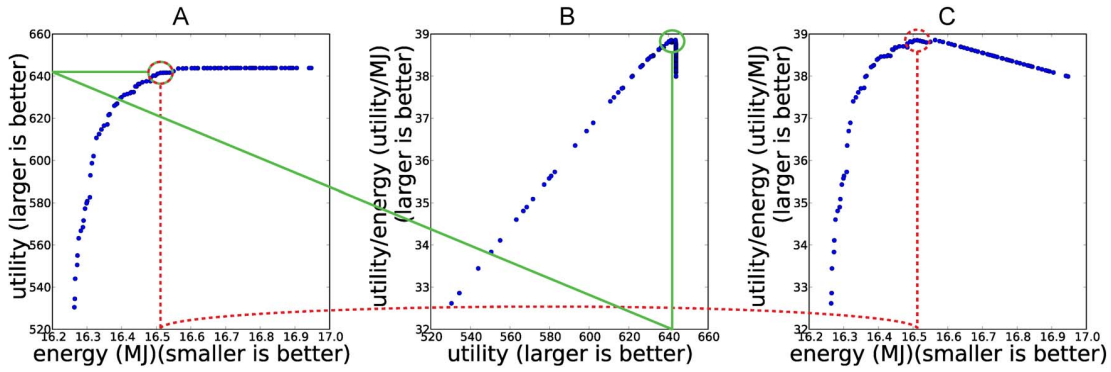

Figure 5. Subplot A shows the Pareto front through 1,000,000 iterations for the "max utility-per-energy" seeded population. The circled region represents the solutions that earn the most utility per energy spent. Subplot B provides the utility value that gives the highest utility earned per energy spent, shown by the solid line. Subplot C provides the energy value that gives the highest utility earned per energy spent, shown by the dashed line.

Finally, Figure 6 contains the Pareto fronts for the largest of our three data sets (4000 tasks). This data set is also evaluated at 1000, 10,000, 100,000, and 1,000,000 iterations. Due to the larger size of this problem, it takes more iterations for the Pareto fronts to start converging. This allows us to see the benefit of using the seeds in the initial populations over the all random initial population. In all cases, our seeded populations are finding solutions that dominate those found by the random 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.

Similar to the first two data sets, we see that Pareto fronts for this data set also exhibit a region where the amount of utility earned per energy spent is maximized. This is the location where the system is operating as efficiently as possible, and can help guide system administrators in making decisions to try and reach that level of efficiency for their systems.

VII. Conclusions and Future Work

Rising costs of energy consumption and the push for greater performance make the need for energy-efficient computing very important as HPC continues to grow. To begin computing in an energy-efficient manner, system administrators must first understand the energy and performance

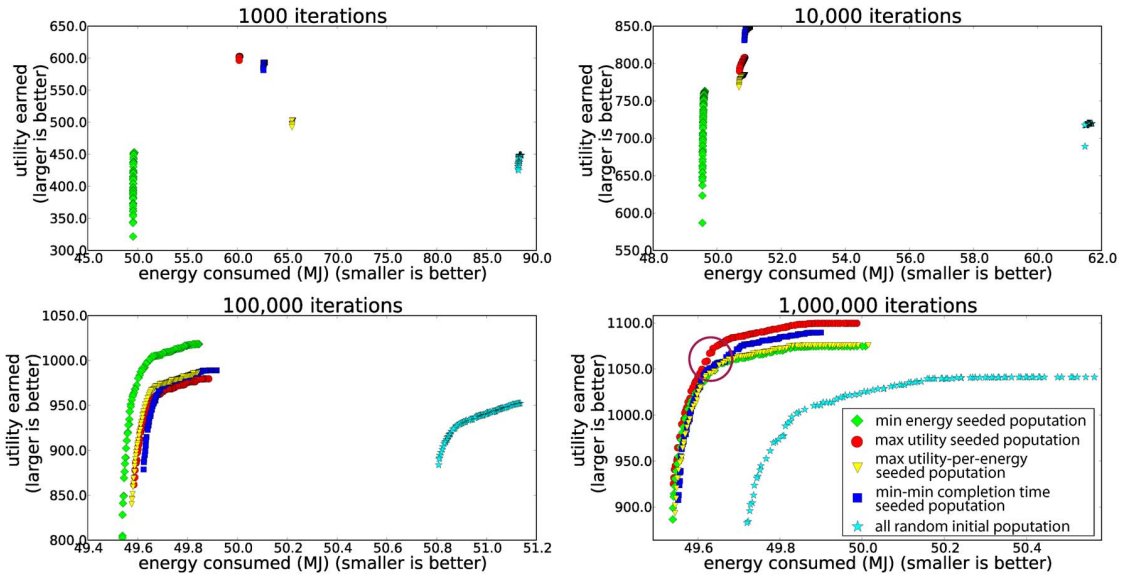

Figure 6. Pareto fronts of total energy consumed vs. total utility earned for the data set containing 4000 tasks (data set 3) for different initial seeded populations through various number of iterations. The circled region represents the solutions that earn the most utility per energy spent. Both the y-axis and x-axis values are specific to each subplot.

characteristics of their systems. In this work, we have provided an analysis framework for investigating the trade-offs between total utility earned and total energy consumed. We have developed a method for creating a synthetic data set that preserves the heterogeneity characteristics of a data set constructed from real historical data.

We showed that by using the NSGA-II we can create well defined Pareto fronts and analyzed how using different seeding heuristics within the initial populations affected the evolution of the Pareto fronts. Finally, we tested our method using three data sets. The first data set contained only real data gathered from online benchmarks. The other two data sets contained synthetic data created from the real data to simulate a larger computing system. These two data sets differed in the number of tasks that were required to execute. Our method successfully illustrates the trade-offs between energy consumption and utility earned for all three data sets.

There are many possible directions for future work. Two are: dropping tasks that will generate negligible utility when they complete, and incorporating dynamic voltage and frequency scaling capabilities of processors.

In summary we have designed an analysis framework that: 1) provides the ability to create synthetic data sets that preserve the heterogeneity measures found from real data sets, 2) provides the ability to take traces from any given system and then use our resource allocation heuristic and simulation infrastructure to plot and analyze the trade-offs between total utility earned and total energy consumed, 3) find the region of the Pareto front where a given system is operating as efficiently as possible, and 4) show the effect different genetic algorithm seeds have for various systems, and how using seeds can create populations that dominate completely random populations.

VIII. Acknowledgements

This work was supported by Oak Ridge National Laboratory and their Extreme Scale Systems Center, by the National Science Foundation Graduate Research Fellowship. Any opinion, 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 resources of the Extreme Scale Systems Center at Oak Ridge National Laboratory as well as the CSU ISTeC Cray System supported by NSF Grant CNS-0923386.

References

[1] Environmental Protection Agency, "Report to congress on server and data center energy efficency," http://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf, Aug. 2007.

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

[3] R. Friese, T. Brinks, C. Oliver, H. J. Siegel, and A. A. Maciejewski, "Analyzing the trade-offs between minimizing makespan and minimizing energy consumption in a heterogeneous resource allocation problem," in *The 2nd Int'l Conf on Advanced Communications and Computation (INFOCOMP 2012)*, Oct. 2012, p. 9 pp.

[4] L. D. Briceno, 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 system," in *The 20th Heterogeneity in Computing Workshop (HCW 2011)*, May 2011, p. 14.

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

[6] J. J. Dongarra, E. Jeannot, E. Saule, and Z. Shi, "Bi-objective scheduling algorithms for optimizing makespan and reliability on heterogeneous systems," in *The 19th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '07)*, 2007, pp. 280–288.

[7] E. Jeannot, E. Saule, and D. Trystram, "Bi-objective approximation scheme for makespan and reliability optimization on uniform parallel machines," in *The 14th Int'l Euro-Par Conf on Parallel Processing (Euro-Par '08)*, 2008, vol. 5168, pp. 877–886.

[8] B. Abbasi, S. Shadrokh, and J. Arkat, "Bi-objective resource-constrained project scheduling with robustness and makespan criteria," *Applied Mathematics and Computation*, vol. 180, no. 1, pp. 146 – 152, 2006.

[9] J. Pasia, R. Hartl, and K. Doerner, "Solving a bi-objective flowshop scheduling problem by Pareto-ant colony optimization," in *Ant Colony Optimization and Swarm Intelligence*, 2006, vol. 4150, pp. 294–305.

[10] Y. He, F. Liu, H.-j. Cao, and C.-b. Li, "A bi-objective model for job-shop scheduling problem to minimize both energy consumption and makespan," *Journal of Central South University of Technology*, vol. 12, pp. 167–171, Oct. 2005.

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

[12] J. Apodaca, D. Young, L. Briceno, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, S. Bahirat, B. Khemka, A. Ramirez, and Y. Zou, "Stochastically robust static resource allocation for energy minimization with a makespan constraint in a heterogeneous computing environment," in *9th IEEE/ACS Int'l Conf on Computer Systems and Applications (AICCSA '11)*, Dec. 2011, pp. 22–31.

[13] B. D. Young, J. Apodaca, L. D. Briceno, J. Smith, S. Pasricha, A. A. Maciejewski, H. J. Siegel, B. Khemka, S. Bahirat, A. Ramirez, and Y. Zou, "Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environment," *Journal of Supercomputing*, accepted, to appear 2012.

[14] J. Teich, "Hardware/software codesign: The past, the present, and predicting the future," *Proceedings of the IEEE*, 2012.

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

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

[17] A. Khokhar, V. Prasanna, M. Shaaban, and C.-L. Wang, "Heterogeneous computing: challenges and opportunities," *IEEE Computer*, vol. 26, no. 6, pp. 18–27, June 1993.

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

[19] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42–50, Jul.-Sep. 1998.

[20] "Intel core i7 3770k power consumption, thermal," http://openbenchmarking.org/result/1204229-SU-CPUMONITO81#system_table, 2012, accessed: 07/24/2012.

[21] A. M. Al-Qawasmeh, A. A. Maciejewski, H. Wang, J. Smith, H. J. Siegel, and J. Potter, "Statistical measures for quantifying task and machine heterogeneities," *The Journal of Supercomputing*, vol. 57, no. 1, pp. 34–50, Jul. 2011.

[22] M. Kendall, *The Advanced Theory of Statistics*. Charles Griffin and Company Limited, 1945, vol. 1.

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

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

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

[26] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing, Special Issue of Software Support for Distributed Computing*, vol. 59, pp. 107–131, Nov. 1999.