Analyzing the Trade-offs Between Minimizing Makespan and Minimizing Energy Consumption in a Heterogeneous Resource Allocation Problem

Ryan Friese*, Tyler Brinks*†, Curt Oliver*, Howard Jay Siegel*†, and Anthony A. Maciejewski*
*Department of Electrical and Computer Engineering
†Department of Computer Science
Colorado State University
Fort Collins, CO, 80523
Email: Ryan.Friese@rams.colostate.edu, tbrinks@engr.colostate.edu, wcoliver@rams.colostate.edu,
HJ@colostate.edu, aam@colostate.edu

***Abstract*—The energy consumption of data centers has been increasing rapidly over the past decade. In some cases, data centers may be physically limited by the amount of power available for consumption. Both the rising cost and physical limitations of available power are increasing the need for energy efficient computing. Data centers must be able to lower their energy consumption while maintaining a high level of performance. Minimizing energy consumption while maximizing performance can be modeled as a bi-objective optimization problem. In this paper, we develop a method to create different resource allocations that illustrate the trade-offs between minimizing energy consumed and minimizing the makespan of a system. By adapting a popular multi-objective genetic algorithm we are able to construct Pareto fronts (via simulation) consisting of Pareto-efficient resource allocations. We analyze different solutions from within the fronts to further understand the relationships between energy consumption and makespan. This information can allow system managers to make intelligent scheduling decisions based on the energy and performance needs of their system.**

***Keywords- bi-objective optimization*; *energy-aware*; *makespan*; *heterogeneous computing*; *resource allocation*.**

I. INTRODUCTION

Over the past decade, the need for energy efficient computing has become increasingly important. As the performance of high performance computing (HPC) systems, such as servers and datacenters, have increased, so has the amount of energy needed to run these systems. According to the Environmental Protection Agency (EPA) [1], it was estimated that between the years 2000 and 2006 the amount of power consumed by HPC systems more than doubled. An estimated 61 billion kWh was consumed by servers and data centers in 2006, approximately equal to 1.5% of the total U.S. electricity consumption for that year. This is equivalent to the electricity consumption of 5.8 million average U.S. households, and amounts to $4.5 billion in electricity costs [1].

In addition to the rising costs of using so much energy, some data centers are now unable to increase their computing performance due to physical limitations on the availability of energy. A survey conducted in 2008 showed that 31% of the data centers surveyed identified energy availability as a key factor limiting new server deployment [2]. Another example to emphasize this point: Morgan Stanley, a global financial services firm based in New York, is physically unable to draw the energy needed to run a new data center in Manhattan [3].

To battle the rising costs of energy consumption, it is essential for HPC systems to be energy-efficient. This focus on energy-efficiency must have as little impact to performance as possible. Unfortunately, the goals of saving energy and achieving high performance often conflict with each other. To understand and illustrate the trade-offs between energy consumption and computing performance, we model this dilemma as a bi-objective optimization problem. When a problem has multiple objectives, it is often the case that there is not just one single optimal solution, but rather a set of optimal solutions. With our research, we provide a method for developing a set of "Pareto"optimal solutions that not only illustrate the trade-offs between energy consumption and performance for a specific computing system, but also allows the system manager to select a solution that fits the system needs and goals.

In this research, we study how different ways of allocating resources to tasks impact the performance and energy consumption of a computing system. We are modeling a data center consisting of a set of heterogeneous machines that must execute a batch of independent tasks. By heterogeneous, we mean that tasks may have different execution and power consumption characteristics when executed on different machines. All the tasks in a given batch are known *a priori* and are all available for scheduling at the beginning of the simulation, making this a static resource allocation problem. We define a resource allocation to be a complete scheduling of tasks to machines. We perform this research in a *static* and *offline* environment, so that, we can evaluate the resource allocations and analyze the trade-offs between the two objectives. The knowledge gained from studies such as these for a particular system can be used to set the parameters needed for designing dynamic, online, allocation

 ISBN: 978-1-61208-226-4

heuristics.

To measure the performance of the system, we examine the makespan of a batch of tasks for a given resource allocation. Makespan is the total amount of time it takes for all the tasks in the batch to finish executing across all the machines. Energy is measured in the number of joules consumed by that same batch of tasks for a given resource allocation. An optimal resource allocation would be one that minimizes both makespan and energy consumed. By adapting the Nondominated Sorting Genetic Algorithm II (NSGA-II) [4] to handle scheduling problems, we are able to create resource allocations that have different makespan and energy consumption values. This set of solutions will then be one basis to analyze the energy and performance trade-offs of the system.

To summarize, in this paper, we make the following contributions:

1) Address the concern of energy efficient computing by modeling the resource allocation problem as a bi-objective optimization problem between minimizing energy consumption and maximizing performance (minimizing makespan).
2) Adapt a well-known multi-objective genetic algorithm to the domain of data center task scheduling.
3) Show that by using different resource allocations, one can greatly affect the energy consumption and performance of a heterogeneous computing system.
4) Construct a set of "Pareto"[5] optimal solutions that can be used to illustrate the trade-offs between system performance and energy consumption, as well as allowing data center managers to select appropriate resource allocations to meet the needs of the specific system.

The remainder of the paper is set up as follows. We discuss the related work in Section II. Section III will describe how we define our bi-objective optimization problem using the NSGA-II. In Section IV, we explain our system model. Our simulation setup is detailed within Section V. Section VI contains our simulation results. Finally, Section VII contains our conclusions and future work for this research.

II. Related Work

In Dongarra et al. [6] and Jeannot et al. [7], a heterogeneous task scheduling problem is modeled as a bi-objective optimization problem between makespan and reliability. This differs from our research because they are not minimizing energy consumption.

The study in Abbasi et al. [8] models a resource-constrained project scheduling problem as a bi-objective problem between makespan and robustness. Abbasi et al. solve this problem using a weighted sum simulated annealing heuristic to generate a single solution. They then adjust the weights to produce different solutions. This is different from our work in that we evaluate our two objective functions independently and generate a Pareto front composed of many different solutions in one run of our algorithm.

A Pareto-ant colony optimization approach is presented in Pasia et al. [9] to solve the bi-objective flowshop scheduling problem. Pasia et al. are minimizing makespan and total tardiness. This differs from our work because they are not considering minimizing energy nor are they using a genetic algorithm to create the solutions.

He et al. [10] develop a bi-objective model for job-shop scheduling to minimize both makespan and energy consumption. There are a couple of differences from our work. The first one is that He et al. model a homogeneous set of machines instead of a heterogeneous set of machines. Second, the algorithm used in He et al. produces a single solution while our algorithm produces a set of solutions.

The goal of minimizing makespan with solutions that are robust against errors in computation time estimates is investigated in Sugavanam et al. [11]. This differs from our work in that we do not consider uncertainties in computation time. Also, Sugavanam et al. are not concerned with energy consumption.

Resource to task matching in an energy constrained heterogeneous computing environment is studied in Kim et al. [12]. The problem is to create robust resource allocations that map tasks onto devices limited by battery capacity (energy constrained) in an ad hoc wireless grid. This differs from our paper because our machines are not energy constrained nor are they in an ad hoc wireless environment. Also the heuristics used in this study only create a single resource allocation, whereas ours creates a set of solutions.

The work in Apodaca et al. [13] studies static resource allocation for energy minimization while meeting a makespan robustness constraint. In contrast to our paper, Apodaca et al. only find a single solution, and we do not place constraints on either objective function.

An energy constrained dynamic resource allocation problem is studied in Young et al. [14]. In this work, the resource allocation must try to finish as many tasks as it can while staying within the energy budget of the system. Our work differs because we are modeling a static environment and have no constraints on how much energy our resource allocations can use.

In Pineau et al. [15], the authors are trying to minimize energy consumption while maximizing throughput. Pineau et al. are modeling a heterogeneous system that executes a single bag-of-tasks application where each task is the same size. To solve the problem, Pineau et al. place a constraint on the throughput objective, and then try to minimize energy while meeting the throughput constraint. While similar to our approach, it differs because we are optimizing for makespan, we model tasks that can differ greatly in size, and we do not constrain either of our objectives.

Mapping tasks to computing resources is also an issue in

 ISBN: 978-1-61208-226-4

hardware/software co-design, Teich et al. [16]. 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. Bi-Objective Optimization Using Genetic Algorithms

A. Overview

It is common for many real-world problems to contain multiple goals or objectives. Often, these objectives work against each other, as optimizing for one objective can negatively impact another objective. This is the case in our research, because it is important for HPC systems to be concerned with both lowering energy consumption as well as increasing overall system performance. In general, resource allocations using more energy will allow one to achieve greater performance, while resource allocations trying to conserve energy will cause the system to have slower performance. In Section III-B, we describe how to determine which solutions should be considered when trying to solve a bi-objective optimization problem. We then briefly discuss the genetic algorithm we have adapted to solve our specific problem in Section III-C.

B. Determining Solutions to a Bi-Objective Optimization Problem

When multiple objectives are present within a problem, it is often the case that there does not exist a single global optimal solution, but rather a *set* of optimal solutions. There is no guarantee one can find the exact solutions within this optimal set, so instead we try to find a set of solutions that are as close to the optimal set as possible. We will call this set of solutions the set of Pareto optimal solutions [5]. This set of Pareto optimal solutions can be used to construct a Pareto front that illustrates the trade-offs between the two objectives.

To understand what it means for a solution to be part of the Pareto optimal set, we illustrate the notion of solution dominance. Dominance is defined as one solution being better than another solution in at least one objective, and better than or equal to in the other objective. To help explain what it means for one solution to dominate another, please refer to Figure 1. Figure 1 shows three potential solutions. The objectives are to minimize energy (along the x-axis), and to minimize makespan (along the y-axis). Let us first examine the relationship between solutions A and B. From the figure we can see that solution B is dominated by A because A uses less energy and has a smaller makespan. Likewise, any solution residing within the upper right (green) shaded region would also be dominated by A. Next, consider solutions A and C. We cannot claim either solution dominates the other because A uses less energy than C, but C has a smaller makespan than A. Thus, for this example, both A and C are solutions in the Pareto optimal set and form the Pareto front. Finally, solution A would be dominated by any solution residing within the lower left (red) shaded area.

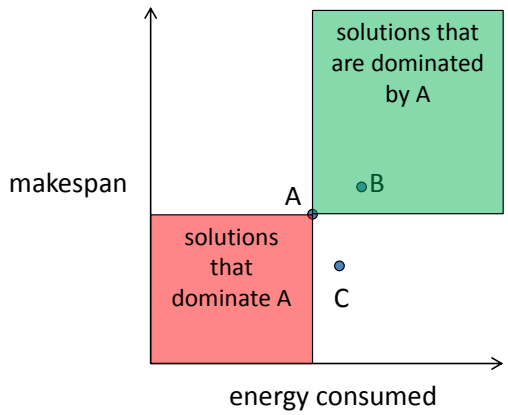

Figure 1. Illustration of solution dominance for three solutions: A, B, and C. Solution A dominates solution B because A has lower energy consumption as well as a lower makespan. Neither solution A nor C dominate each other because A uses less energy, while C has a lower makespan.

C. Nondominated Sorting Genetic Algorithm II Adapted For Resource Allocation

To solve our bi-objective optimization problem, we chose to implement a popular genetic algorithm from the literature, the Nondominated Sorting Genetic Algorithm II (NSGA II) [4]. We will briefly describe the algorithm and how we have adapted it for our use.

The NSGA II is a multi-objective genetic algorithm that uses the idea of solution dominance to create offspring populations, where for our problem domain a population is a set of possible resource allocations. For a given population, the algorithm performs the nondominated sorting algorithm that ranks the solutions within the population based on how many solutions dominate a given solution. Any solution that is not dominated by any other solution is given a rank of one and is part of the current Pareto optimal set. The basic algorithm is outlined in Algorithm 1.

To create the child population in step three, we start with a parent population of size N. From this parent population N crossover operations are performed to create a child population of also of size N. The mutation operation is then performed with a given probability on each chromosome in the child population. If a chromosome is selected for mutation, only the mutated version is kept in the population.

It is important to note the NSGA II is an elitist algorithm as it combined the offspring and parent populations in step six. Elitism means that the algorithm keeps the best chromosomes from the previous generation in consideration for the current generation.

 ISBN: 978-1-61208-226-4

Algorithm 1 NSGA II algorithm

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: take a subset of solutions from the highest rank number used based on crowding distance [4]
11: **end if**
12: **end while**
13: the final population is the Pareto front used to show the trade-offs between the two objectives

To further explain how the next parent population is created in steps eight and nine, assume we have a parent population with 100 chromosomes and a child population with 100 chromosomes for a total of 200 chromosomes. We want to create a new parent population for the next generation that only has 100 chromosomes. Let us assume, that after step seven (where we have ranked the current populations), there are 60 chromosomes of rank one, 30 chromosomes of rank two, 20 chromosomes of rank three, and 90 chromosomes that have a rank higher than three. First, we will place all the rank one chromosomes into the new population, this will leave room for 40 more chromosomes. Next, we place all the rank two chromosomes into the population, leaving room for ten more chromosomes. Since there are 20 rank three chromosomes, but only room left in the new population for ten chromosomes we must select a subset of the rank three chromosomes to place in the population. These ten solutions will be based on the crowding distance [4] and we will have our full 100 chromosome population.

To use the NSGA II, we needed to encode the algorithm so that it could be used to solve resource allocation problems. This meant we needed to create our own genes, chromosomes, crossover operator, and mutation operator. Genes are the basic data structure of the genetic algorithm, and for our problem each gene represents a task. Within each gene there is a single integer number representing the machine on which the task will execute. Chromosomes represent complete solutions, i.e., resource allocations. Each chromosome is comprised of T genes, where T is the number of tasks the system must execute. The i^{th} gene in a chromosome represents the same task in every chromosome. Each chromosome is individually evaluated with respect to makespan and energy consumption, allowing dominance relationships to be found amongst all the chromosomes within a population.

To allow chromosomes and populations to evolve from generation to generation, we implemented the following crossover and mutation operations. For crossover, two chromosomes are selected randomly from the population. Next, the indices of two genes within the chromosomes are selected randomly. We then swap the genes between these two indices from one chromosome to the other. This operation switches the machines on which the tasks will execute. This potentially allows chromosomes making good scheduling decisions to pass on the useful traits to other chromosomes. For mutation we randomly select a chromosome from the population and randomly select a gene within that chromosome. We then randomly select a machine for that task to execute on.

IV. System Model

A. Machines

Our computing system is modeled as a suite of M heterogeneous machines where each node belongs to a specific machine type μ. Machines are assumed to be dedicated, meaning only one task can be executing on the machine at a time, such as the ISTeC Cray located at Colorado State University [17]. Once a task starts executing it runs until it is finished. Machines of the same machine type are identical to one another. Machine types exhibit heterogeneous performance (i.e., machine type A may be faster than machine type B for some tasks, but slower for others) [18]. Machine types are also heterogeneous with respect to energy consumption (i.e., machine type A may use less energy than machine type B for some tasks, but more energy for others). We implement a heterogeneous behavior for both performance and energy consumption to model a computing system that contains a variety of different resources. Real world systems may be *highly* heterogeneous due to having machines of different ages, varying micro-architectures, subsets of machines that have accelerators, and the inclusion of special purpose machines. Differences in machine components such as memory modules, hard disks, and power supplies also cause systems to be heterogeneous.

B. Workload

We assume we have a static collection of T tasks. Each task t is a member of a given task type. Each task type has unique performance and energy consumption characteristics for executing on each of the machine types. To model the performance of the task types, we assume that the estimated time to compute (ETC) a task of type τ on a machine of type μ, ETC(τ,μ), is given. Entries in the ETC matrix represent the estimated amount of time a task type takes

 ISBN: 978-1-61208-226-4

to execute on a given machine type. Research in resource allocation often assumes the availability of ETC information (e.g., [19, 20, 21, 22]). We have provided the analysis framework for system administrators to use ETC information from data collected on their specific systems. This allows for systems of varying size and heterogeneity to be analyzed. For our simulation studies, we have constructed synthetic ETC values modeling real-world systems, but these values can also be taken from various sources of historical data (e.g., [21, 20]).

Similar to the ETC values used for determining compute times, we also assume we have estimated power consumption (EPC) values that tell us the average power a task type consumes while executing on a specific machine type. The EPC values represent the power consumption of a machine as a whole, not just the CPU. Again, we have constructed synthetic EPC values for our simulations, but historical power consumption data could also be used to populate the matrix.

Finally, to obtain the estimated energy consumed (EEC) of a task of type τ on a machine μ we take the product of the execution time and the estimated power consumption, as shown below.

$$EEC[\tau, \mu] = ETC[\tau, \mu] \times EPC[\tau, \mu] \tag{1}$$

C. Objective Functions

1) Makespan: One objective we are trying to optimize is makespan, which is the total amount of time it takes for all the tasks in the batch to finish executing across all machines. When optimizing for makespan the goal is to minimize the makespan. For a given resource allocation, calculating the makespan of the system requires that we first determine the finishing time of each machine.

To calculate the finishing time of a machine we let the set T_m represent all the tasks in *T* that were allocated to machine *m*, where $t_m \in T_m$. Let the function $\Upsilon(t_m)$ return the task type that task t_m belongs to, and let the function $\Omega(m)$ return the machine type to which machine m belongs. We then calculate the expected finishing time of machine *m* denoted as F_m, with the following equation

$$F_m = \sum_{\forall t_m \in T_m} ETC(\Upsilon(t_m), \Omega(m)). \tag{2}$$

The makespan for a given resource allocation, denoted ρ, can be found from the machine with the maximum finishing time, and is given as

$$\rho = \max_{\forall m \in M} F_m. \tag{3}$$

2) Energy Consumption: The other objective we will optimize for is energy consumption. For a given resource allocation, the total energy consumed is the sum of the energy consumed by each task to finish executing. Recall that the amount of energy consumed by a task is dependent upon the machine on which that task is executing. Therefore, the total energy consumed for a resource allocation, denoted E, can be found as

$$E = \sum_{\forall t_m \in T_m, \forall m \in M} EEC[\Upsilon(t_m), \Omega(m)]. \tag{4}$$

V. Simulation Setup

A. Simulation Environment Parameters

To construct a Pareto front and illustrate the trade-offs between makespan and energy consumption, we conducted numerous simulation trials. For each trial, the number of tasks to execute was set to 1000, with 50 different task types. The number of machines used throughout the simulations was set to 50, with 10 different machine types. The number of tasks per task type and number of compute nodes per compute node type were randomly assigned, and could change from trial to trial.

The ETC values were obtained using the Coefficient of Variation (COV) method from [18], which allows us to model a heterogeneous set of machine types and task types. For our simulations, the mean execution time for the tasks was 10 seconds, and the variance amongst the tasks was 0.1, while the variance amongst the machines was 0.25. These parameters allowed us to model a heterogeneous set of compute nodes.

The EPC values were constructed in a similar manner as the ETC values. Specifically, the mean power consumption for the tasks was 200 watts, and the variation amongst tasks was 0.1, while the variance amongst the machines was 0.2.

For each trial, the genetic algorithm consisted of 100 chromosomes. In the initial population, we used 98 randomly generated chromosomes, and two chromosomes generated using two heuristics based on approaches taken from literature, as discussed below.

B. Seeding Heuristics

The goal of the seeding heuristics are to provide the genetic algorithm with initial solutions that try to optimize the objectives. These seeds can help guide the genetic algorithm towards better solutions faster than an all-random initial population. We chose to implement two greedy heuristics, min energy and min-min completion time, based on concepts found in [23, 24, 25]. The execution times of the greedy heuristics are negligible compared to the NSGA II. Utilizing these seeds in the initial population does not negatively affect the computation time of the NSGA II.

1) Min Energy: Min energy is a single stage greedy heuristic that maps tasks to machines to minimize energy consumption. The heuristic selects a task from the batch and places that task on to the machine that has the smallest energy consumption. For this heuristic, the order in which tasks are mapped to machines does not matter. This heuristic creates a solution that will have the minimum possible

 ISBN: 978-1-61208-226-4

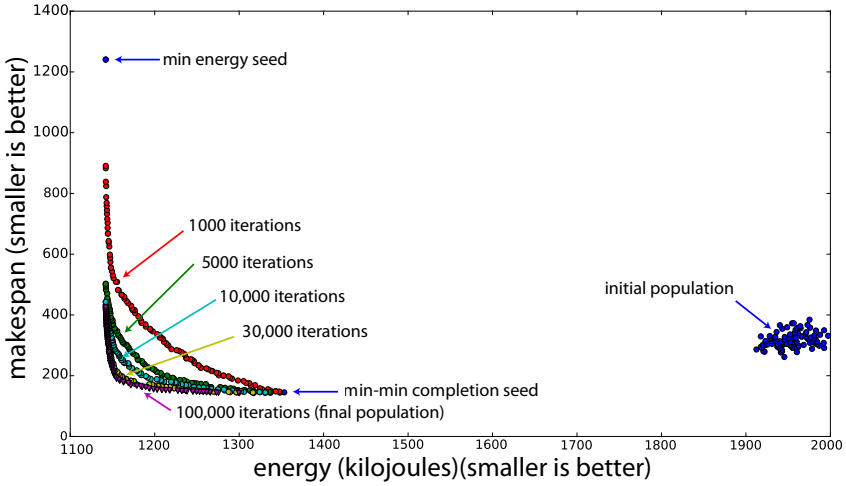

Figure 2. Pareto fronts showing the trade-offs between energy consumption and makespan. Shows the evolution of the solutions through number of iterations completed

energy consumption. For a solution to be more efficient, it must have a smaller makespan.

2) Min-Min Completion Time: Min-min completion time is a two-stage greedy heuristic that maps tasks to machines to minimize the makespan of the system. During each iteration of the heuristic, one task gets mapped to the machine that provides the minimum completion time. One iteration consists of two stages. In the first stage, every unmapped task finds the machine that minimizes completion time. In the second stage, the heuristic selects the task and machine pair from the first stage that has the smallest overall completion time and assigns that task to that machine. This continues until there are no more tasks to map. There is no guarantee that the solution created by this heuristic represents the absolute lowest makespan of the system, so better solutions can potentially improve in both makespan and energy consumption.

VI. Results

Throughout this section, we will only be discussing the results from one simulation trial. We have confirmed that the findings and trends for this trial hold for the other trials we ran. In Figure 2, we show the evolution of the solutions through the number of NSGA-II iterations completed. It is important to note for genetic algorithms, as we increase the number of iterations, the genetic algorithm will in general find new and better solutions; some solutions may remain a member of the Pareto front as we increase the iterations. Each point in Figure 2 represents a complete resource allocation. The set of points corresponding to a given number of iterations form the Pareto front. These points are obtained from the genetic algorithm running through that number of iterations. We see that as the genetic algorithm runs for more iterations, the Pareto fronts are converging towards the lower-left corner. This makes sense because we are minimizing makespan as well as energy consumption. We can also see that for this size problem there is very little improvement to the Pareto front after 30,000 iterations. The size of the problem as well as using the two seeds help the solutions converge in a relatively short number of iterations. Also, observe that both of the seeds provide good starting solutions for the genetic algorithm to evolve from relative to the rest of the initial population.

Although it is useful to see how the solutions evolve over time, the most important information to take away from Figure 2 are the trade-offs between makespan and energy consumed. Figure 3 shows a blown-up plot of the final Pareto front from Figure 2. There are a number of points we can learn from the final Pareto front shown in Figure 3. One such point is circled in red. We can see that around this point there is a definite and visible “knee” in the front. To the left of the knee, small increases in energy consumption result in large decreases in makespan. To the right of the knee we see the opposite, small decreases in makespan result in large increases in energy. Given the information provided in this Pareto front, it is then up to the system manager to select which region of the curve to operate in based on the individual system needs.

To further understand how solutions in the Pareto front differ from one another, we analyzed the individual finishing times and energy consumptions for the 50 machines at five points along the final Pareto front. The five points were the two endpoints of the front, the middle point, and the two points between the middle point and each endpoint, as shown in Figure 4 and Figure 5. The results for machine finishing

 ISBN: 978-1-61208-226-4

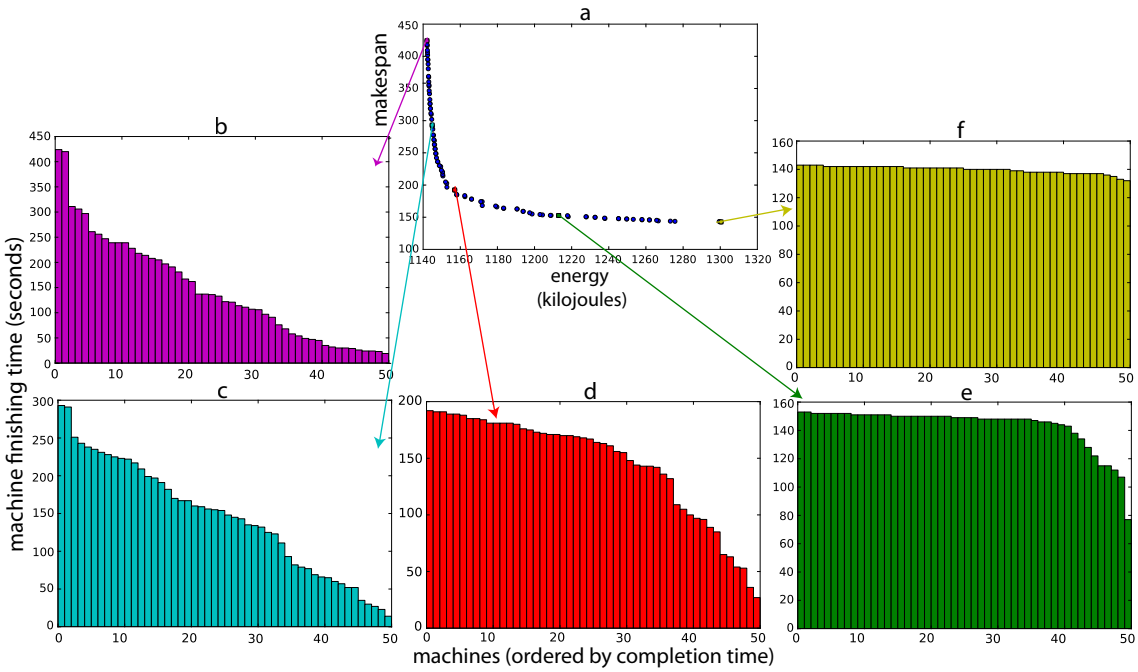

Figure 4. Finishing time of the 50 machines in descending order of finishing time for five solutions from the Pareto front. The y-axis contains different ranges of machine finishing times from plot to plot (to show each plot in greater detail). Subfigure “a” is the same as Figure 3 and has different axis labels from the other subplots. Each subplot b-f has a different ordering of the machines along the x-axis.

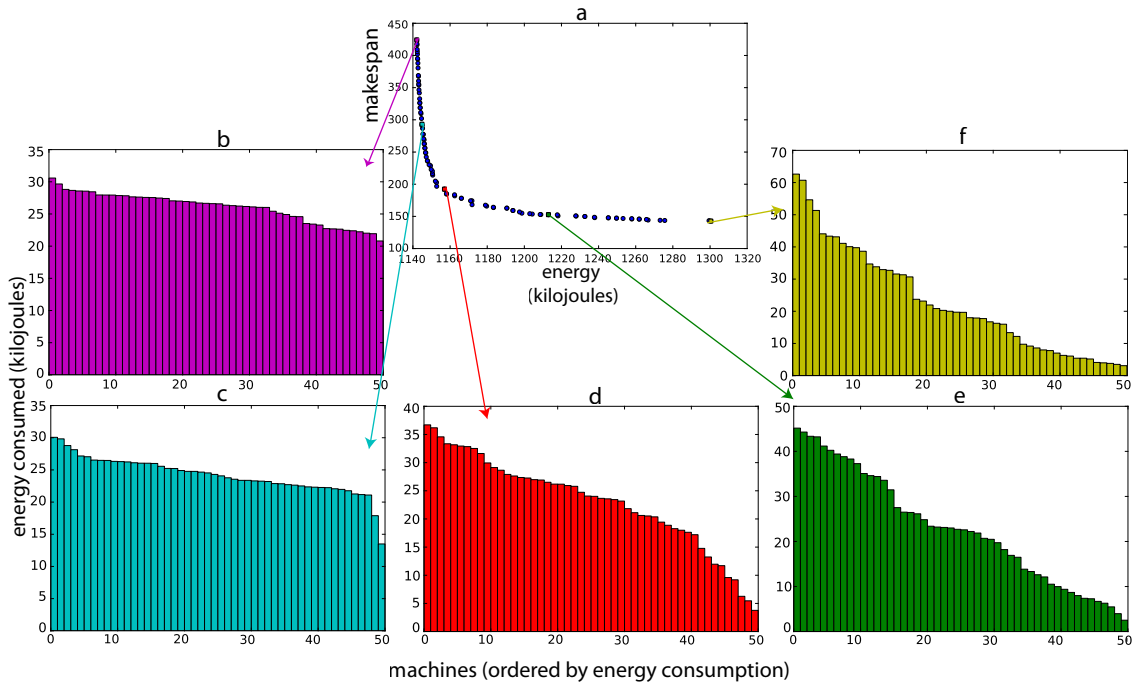

Figure 5. Energy consumption of the 50 machines in descending order of energy consumed for five solutions from the Pareto front. The y-axis contains different ranges of machine energy consumption from plot to plot (to show each plot in greater detail). Subfigure “a” is the same as Figure 3 and has different axis labels from the other subplots.Each subplot b-f has a different ordering of the machines along the x-axis.

 ISBN: 978-1-61208-226-4

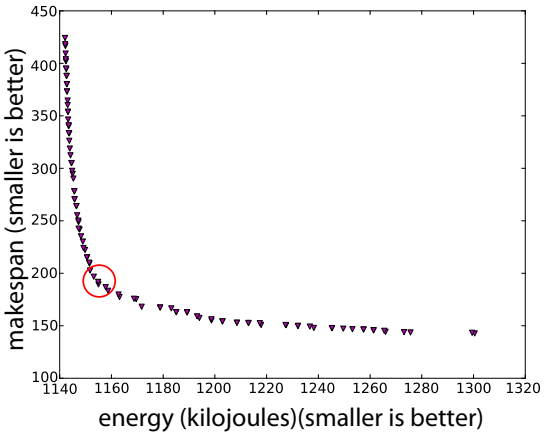

Figure 3. The final Pareto front after 100,000 iterations showing the trade-offs between energy consumed and makespan.

times are shown in Figure 4, while the results for machine energy consumption are shown in Figure 5.

First, we consider Figure 4, which focuses on the finishing times of each machine. Each of the five subplots (b-f) in the figure represents a solution from the Pareto front. On the x-axis of the plots we have the machines sorted by finishing time in descending order, this ordering is different from subplot to subplot. On the y-axis we have the actual finishing time of each machine. Note that each figure (b-f) has different values along the y-axis. In Figure 4.f, we have the solution that provides the lowest makespan. As we can see, the finishing times for all the machines are evenly balanced; this allows the makespan to be small since no one machine is doing a lot more work than the others. As we move left along the Pareto front selected points in Figure 4.a (minimizing energy) we see that the solutions become more and more unbalanced with respect to machine finishing times going from Figure 4.f to Figure 4.e to Figure 4.d, etc. This is because each task type has an affinity for a specific machine type that minimizes that task type's energy consumption.

If we now consider the plots of machine energy consumption in Figure 5 which focus on energy consumption, we see similar trends as before, but in reverse order. This time the machines are ordered in descending order based on energy consumption. Figures 5.b and 5.c are more balanced in terms of energy consumption amongst the machines. This is because this area of the Pareto front focuses on trying to minimize energy and thus makespan is compromised; as we saw in the corresponding makespan plots from Figure 4. By similar reasoning, this is why Figure 5.f is unbalanced. In this region of the Pareto front, makespan is being optimized so tasks are going to have to run on machines that use more energy to lower system makespan.

With the information provided by the Pareto fronts as well as the plots showing the completion time and energy consumption of individual machines, a system manager will be able to analyze the trade-offs between energy consumption and makespan. The system manager can then make a scheduling decision based on the needs of the computing system.

VII. Conclusion and Future Work

As high performance computing systems continue to become more powerful, the energy required to power these systems also increases. In this paper we have developed a bi-objective optimization model that can be used to illustrate the trade-offs between the makespan and energy consumption of a system. Having adapted the nondominated sorting genetic algorithm for use within our domain, we successfully ran simulations that provided us well defined Pareto fronts. We then analyzed five different solutions from the final Pareto front and discussed the differences in their makespan and energy consumption. Given this information a system administrator would be able to pick a specific resource allocation from the Pareto front that meets the energy and performance needs of the system.

There are many possible directions for future work. We would like to enhance our energy consumption model by considering machines that utilize dynamic voltage and frequency scaling techniques to save more energy. We do not currently consider communications within our environment, but the analysis framework we present here could be extended to do this. We would like to try and increase the execution rate and performance of the genetic algorithm by trying numerous parallel techniques. To more accurately model real-world systems, we would like to use probability density functions to model both task execution times and task energy consumption characteristics.

Acknowledgements

The authors thank Bhavesh Khemka and Mark Oxley for their valuable comments on this work. This research was supported by the United State National Science Foundation (NSF) under grant number CNS-0905399, and by the Colorado State University George T. Abell Endowment.

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

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

 ISBN: 978-1-61208-226-4

[4] 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.

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

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

[12] 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.

[13] J. Apodaca, D. Young, L. Briceno, J. Smith, S. Pasricha, A. Maciejewski, H. 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 International Conference on Computer Systems and Applications (AICCSA '11)*, Dec. 2011, pp. 22–31.

[14] 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.

[15] J.-F. Pineau, Y. Robert, and F. Vivien, "Energy-aware scheduling of bag-of-tasks applications on master-worker platforms," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 2, pp. 145–157, 2011.

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

[17] "ISTeC cray high performance computing (HPC) system," http://http://istec.colostate.edu/istec_cray/, Aug. 2012.

[18] S. Ali, H. 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.

[19] "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.

[20] 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.

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

 ISBN: 978-1-61208-226-4