Energy and Makespan Tradeoffs in Heterogeneous Computing Systems using Efficient Linear Programming Techniques

Kyle M. Tarplee, *Senior Member, IEEE*, Ryan Friese, *Member, IEEE*, Anthony A. Maciejewski, *Fellow, IEEE*, Howard Jay Siegel, *Fellow, IEEE*, and Edwin K. P. Chong, *Fellow, IEEE*

Abstract—Resource management for large-scale high performance computing systems pose difficult challenges to system administrators. The extreme scale of these modern systems require task scheduling algorithms that are capable of handling at least millions of tasks and thousands of machines. These large computing systems consume vast amounts of electricity leading to high operating costs. System administrators try to simultaneously reduce operating costs and offer state-of-the-art performance; however, these are often conflicting objectives. Highly scalable algorithms are necessary to schedule tasks efficiently and to help system administrators gain insight into energy/performance trade-offs of the system. System administrators can examine this trade-off space to quantify how much a difference in the performance level will cost in electricity, or analyze how much performance can be expected within an energy budget. In this study, we design a novel linear programming based resource allocation algorithm for a heterogeneous computing system to efficiently compute high quality solutions for simultaneously minimizing energy and makespan. These solutions are used to bound the Pareto front to easily trade-off energy and performance. The new algorithms are highly scalable in both solution quality and computation time compared to existing algorithms, especially as the problem size increases.

Index Terms—High performance computing, scheduling, resource management, bag-of-tasks, energy-aware, heterogeneous computing, vector optimization, linear programming

1 Introduction

Today's high performance computing (HPC) systems often have hundreds of thousands of cores, processors, and/or machines. The need for these extremely large HPC systems is driven by increasingly large HPC workloads comprising potentially millions of tasks. The increase in computational capability of HPC systems also results in a significant increase in its energy consumption. Therefore, there is a growing need for computationally efficient algorithms for energy-aware scheduling of tasks to machines in such large-scale environments.

HPC systems have seen dramatic increases in their power consumption [1], [2]. This increase in power consumption can increase electricity costs for the operators, cause degradation in the electronic components, and create additional stress on the electrical infrastructure that supports these facilities [3]. Additionally, the goals of HPC users often conflict with the goals of HPC operators. The user's goal is to finish their workload as quickly as possible. Often, this is in conflict with the goal of the system operator to consume less energy, and typically such a situation requires the sacrifice of one of the goals to satisfy the other. To balance the performance and energy costs of the system it is important to provide the system administrator with a tool that provides a set of solutions that trade-off these objectives.

In this study, a set of efficient and scalable algorithms are proposed that can help system administrators quickly gain insight into the energy and performance trade-offs of their HPC system through the use of intelligent resource allocation. The algorithms proposed have very fast run times, good asymptotic algorithm complexity, and produce schedules that are closer to optimal as the problem size increases. This approach is therefore very well suited to large-scale HPC environments.

Our work considers a common scheduling model where users submit a set of independent tasks known as a *bag-of-tasks* [4]. We assume that the full bag-of-tasks is known a priori [5] (i.e., *static scheduling*), a task executes on only one machine, and a machine may only process one task at a time. We study HPC environments that have highly heterogeneous tasks and machines, known as heterogeneous computing (HC) systems [6].

HC systems often have some special-purpose machines that can perform specific tasks quickly, while other tasks might not be able to run on them. Another cause of heterogeneity is differing computational capability, input/output

- *K.M. Tarplee, R. Friese, and A.A. Maciejewski are with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523. E-mail: {kyle.tarplee, ryan.friese, aam}@colostate.edu.*
- *H.J. Siegel is with the Department of Electrical and Computer Engineering, and the Department of Computer Science, Colorado State University, Fort Collins, CO 80523. E-mail: hj@colostate.edu.*
- *E.K.P. Chong is with the Department of Electrical and Computer Engineering, and the Department of Mathematics Colorado State University Fort Collins, CO 80523. E-mail: edwin.chong@colostate.edu.*

Manuscript received 13 Sept. 2014; revised 14 June 2015; accepted 7 July 2015. Date of publication 12 July 2015; date of current version 18 May 2016. Recommended for acceptance by U. V. Catalyurek.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2015.2456020

bottlenecks, or memory limitations. The machines may further differ in the average power consumed for each task type. Machines can have different architectures, leading to vastly different power consumption characteristics. For instance, a task that runs on a GPU might consume more power but execute much faster, therefore consuming less energy to execute than the same task run on a general-purpose machine. The heterogeneity in execution time of the tasks provides the scheduler degrees of freedom to greatly improve the performance as compared to a naïve scheduling algorithm. Similarly the heterogeneity in the power consumption allows the scheduler to decrease the energy consumption.

In this study, we consider optimizing two conflicting objectives. The first is to minimize the *makespan*, that is, the maximum finishing time of all tasks. The second is to minimize the total energy consumption of all machines in the HPC system. We design a novel technique that utilizes a unique relaxation of this scheduling problem then solves it using, in part, linear programming (LP) for generating a set of high-quality solutions that represent the tradeoff space between makespan and energy consumption (i.e., Pareto front).

In summary the contributions of this paper are:

1) the formulation and evaluation of algorithms that:
   a) efficiently compute tight lower bounds on the energy and makespan using LP,
   b) generate a set of high quality bi-objective solutions (i.e., Pareto front), and
   c) improve upon the Pareto front approximation via convex filling,
2) the addition of idle power consumption to the formulation of the energy/makespan problem in [4],
3) a comparison to other Pareto front generation algorithms, and
4) the design and evaluation of a quantitative measure for comparing the quality of bounds on the Pareto front.

The rest of this paper is organized as follows: First an algorithm for minimum makespan and energy scheduling is presented in Section 2. Vector optimization background is given as a tool to solve the bi-objective energy and makespan scheduling problem in Section 3. Section 4 describes an algorithm to generate Pareto fronts and the convex fill algorithm to further improve the Pareto fronts. Section 5 presents the results by comparing the Pareto fronts to an implementation of the non-dominated sorting genetic Algorithm 2 (NSGA-II) for various HPC environments. The Algorithm's complexity is given in Section 6 along with experimental execution time results. We discuss related work in Section 7 and Section 8 concludes this study and presents some ideas for future work.

2 Algorithm Design

2.1 Approach

The fundamental approach of this paper is to apply divisible load theory (DLT) [7] to ease the computational requirements of calculating solutions for the makespan and energy scheduling problem. The technique has two major steps. The first step uses DLT, where we assume a single task is allowed to be divided and scheduled onto any number of machines, to calculate the lower-bound solution. After the lower-bound solution is computed, a two-phase algorithm is used to recover a feasible solution from the infeasible lower-bound solution. The feasible solution will be shown empirically to be a tight upper bound on the optimal solution.

HC systems often have groups of machines, typically purchased at the same time, that have identical or very similar performance and power characteristics. This allows one to view these similar machines (only for the purposes of analysis) as a unique machine type. Machines belonging to a *machine type* have virtually indistinguishable performance and power properties with respect to the workload. Machines of the same type may differ vastly in feature sets so long as the performance and power consumption of the tasks under consideration are not affected. Tasks often exhibit natural groupings as well. Tasks of the same *task type* are often submitted many times to perform statistical simulations and other repetitive jobs. Having groupings for tasks and groupings for machines permits less profiling effort to estimate the run time and power consumption for each task on each machine.[1]

Traditionally the static scheduling problem is posed as assigning all tasks to all machines. This formulation is not well suited for recovering a high quality feasible solution from a relaxation of the problem. The decision variables in the classic formulation are binary valued (a task is assigned or not assigned to a machine), and rounding a real value from the lower bound to a binary value can change the objective significantly. Complicated rounding schemes are necessary to iteratively compute a suitable solution. Rather than addressing the problem of assigning all tasks to all machines, we pose the problem as determining the number of tasks of each type to assign to machines of each type. With this modification, decision variables will be large integers $\gg 1$, resulting in only a small error to the objective function when rounding to the nearest integer. This approximation is most accurate when the number of tasks assigned to each machine type is large. In addition to easing the recovery of the integer solution, another benefit of this formulation is that it is significantly less computationally intensive due to solving the higher level assignment of tasks types to machine types with DLT, before solving the fine-grain assignment of individual tasks to machines. As such, this approach can be thought of as a hierarchical solution to the static scheduling problem. Furthermore, for the size of problems considered in this paper, the classical relaxation is not solvable in reasonable run time with current computing capabilities.

2.2 Lower Bound

The lower bound on the makespan and energy is given by the solution to an linear programming (LP) problem and is formulated as follows. Let there be T task types and M machine types. Let T_i be the number of tasks of type i and M_j be the number of machines of type j. Let μ_{ij} be the

1. This model applies directly to multicore systems where a task executes on a single core, however, the nomenclature changes. Multicore systems typically have homogeneous cores. Identical cores, from all machines, can be considered *machines* belonging to a single *machine type*. Different types of cores would belong to different *machine types*.

number of tasks of type i assigned to machine type j, where $\mu_{ij} \in \mathbb{R}$ is the primary decision variable in the optimization problem. Let **ETC** be a $T \times M$ matrix where ETC_{ij} is the *estimated time to compute* a task of type i on a machine of type j. Similarly, let **APC** be a $T \times M$ matrix where APC_{ij} is the *average power consumption* for executing a task of type i on a machine of type j. These matrices are frequently used in scheduling algorithms (e.g., [5], [8], [9], [10], [11], [12]). **ETC** and **APC** are generally obtained from historical data in real environments.

The lower bound on the finishing time of the machines of a machine type is found by allowing tasks assigned to a machine type to be divided among all machines to ensure the minimal finishing time. With this conservative approximation, all tasks in machine type j finish at the same time. The finishing time of any machine of type j, denoted by F_j, is given by

$$F_j = \frac{1}{M_j} \sum_i \mu_{ij} ETC_{ij} \ . \tag{1}$$

Throughout this work, sums over i always go from 1 to T and sums over j always go from 1 to M, thus the ranges are omitted. Given that F_j is a lower bound on the finishing time for a machine type, the tightest lower bound on the makespan, denoted by MS_{LB}, is

$$MS_{LB} = \max_j F_j \ . \tag{2}$$

Without considering idle machines, the energy consumed by a bag-of-tasks is given by $\sum_i \sum_j \mu_{ij} \, APC_{ij} ETC_{ij}$. In HPC environments the machines are often powered off or put into a hibernate state to reduce the static power consumption. To incorporate all of these modes of operation, define the idle power consumption $APC_{\emptyset j}$ to be that part of APC_{ij} that occurs when no task is executing on a machine of type j. Not all machines will finish executing tasks at the same time. The makespan is used to compute the energy consumed by idle machines. The equation for the lower bound on the energy consumed while incorporating idle power, denoted by E_{LB}, is given by

$$\begin{aligned} E_{LB} &= \sum_i \sum_j \mu_{ij} \, APC_{ij} ETC_{ij} \\ &\quad + \sum_j M_j \, APC_{\emptyset j} (MS_{LB} - F_j) \\ &= \sum_i \sum_j \mu_{ij} \, ETC_{ij} \big(APC_{ij} - APC_{\emptyset j} \big) \\ &\quad + \sum_j M_j \, APC_{\emptyset j} MS_{LB} \end{aligned} \tag{3}$$

where the second term in the first equation accounts for the idle power. The second equation in (3) breaks the energy into dynamic power and idle power consumption terms. Due to the idle power model, the energy consumption depends directly on the makespan.

The bi-objective optimization problem for the lower bound is:

$$\begin{aligned} \underset{\boldsymbol{\mu}, MS_{LB}}{\text{minimize}} \quad & \begin{pmatrix} E_{LB} \\ MS_{LB} \end{pmatrix} \\ \text{subject to :} \quad & \forall i \qquad \sum_j \mu_{ij} = T_i \\ & \forall j \qquad F_j \leq MS_{LB} \\ & \forall i, j \quad \mu_{ij} \geq 0 \end{aligned} \tag{4}$$

The objective of (4) is to minimize E_{LB} and MS_{LB}, where $\boldsymbol{\mu}$ is the primary decision variable. MS_{LB} is an auxiliary decision variable necessary to model the objective function in (2). The first constraint ensures that all tasks in the bag are assigned to some machine type(s). The second constraint is the makespan constraint. Because the objective is to minimize makespan, the MS_{LB} variable will be equal to the maximum finishing time of all the machine types. The third constraint ensures that there are no negative assignments in the solutions.

This vector optimization problem can be solved to find a collection of optimal solutions. It is often solved by weighting the objective functions to form an LP problem. Methods to find a collection of solutions are presented in Section 4.

Ideally, this LP problem would be solved optimally with $\mu_{ij} \in \mathbb{Z}_{\geq 0}$. However, for practical scheduling problems, finding the optimal integer solution is often not possible due to the high computational cost. Fortunately, efficient algorithms exist that produce high quality sub-optimal feasible solutions. The next few sections describe how we take an infeasible real-valued solution from the linear program and build a complete feasible allocation.

2.3 Recovery Algorithm

2.3.1 Overview

An algorithm is necessary to recover a feasible solution (i.e., full resource allocation) from the infeasible solution obtained from the lower bound in (4). Numerous approaches have been proposed in the literature for solving integer LP problems by first relaxing them to real-valued LP problems [13]. Our approach here follows this common technique except using computationally inexpensive algorithms tailored to this particular optimization problem. The recovery algorithm is decomposed into two phases. The first phase rounds the solution to the nearest solution while taking care to maintain feasibility of (4). The second phase, called local assignment, assigns tasks to actual machines to build the full resource allocation. The details of the two phases of the recovery algorithm are detailed in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2015.2456020.

3 Linear Vector Optimization

3.1 Introduction

Multi-objective optimization is challenging because there is usually no single solution that is superior to all others. Instead, there is a set of superior feasible solutions that are referred to as the *non-dominated* solutions [14]. When all

objectives are to be minimized, a feasible solution x dominates a feasible solution y when

$$\begin{aligned} &\forall i \quad f_i(x) \leq f_i(y) \\ &\exists i \quad f_i(x) < f_i(y) \end{aligned} \tag{5}$$

where $f_i(\cdot)$ is the ith objective function. Feasible solutions that are dominated are generally of little interest because one can always find a better solution in some or all objectives by selecting a solution from the non-dominated set. The non-dominated solutions, also known as *outcomes* and *efficient points*, compose the Pareto front.

The optimization problem in (4) is used to compute the lower bound to a bi-objective linear convex optimization problem with convex constraints. The results to follow in this section apply only to this lower-bound scheduling algorithm. These results do not apply after the solution has been rounded or locally assigned because those are non-linear operations. In this section, the term Pareto front will be used to denote the Pareto front of the linear vector optimization problem (lower bound).

Let $C \in \mathbb{R}^{m \times n}$ be the linear mapping from the schedule to the objective space. For our scheduling problem this is a two-dimensional space consisting of energy and makespan; however, these results apply to larger dimensional objective spaces as well. Let $\mathcal{X} \subset \mathbb{R}^n$ be the convex set of constraints, thus it has the property

$$\forall x_a, x_b \in \mathcal{X} \Longrightarrow \forall \lambda \in [0,1] : \lambda x_a + (1-\lambda) x_b \in \mathcal{X} \ . \tag{6}$$

The decision variable, x, is contained within $\mathcal{X}$. For the lower-bound optimization problem x is a vector that contains the schedule, $\boldsymbol{\mu}$, and the auxiliary decision variable, makespan.

Using the above notation, the linear convex vector optimization problem is

$$\underset{x \in \mathcal{X}}{\text{minimize}} \quad y = Cx \ . \tag{7}$$

The lower-bound optimization problem in (4) can be easily converted to this form.

Let the objective space, spanned by y, be given by $\mathcal{Y} \subset \mathbb{R}^m$ and its non-dominated subspace given by $\mathcal{Y}_{ND} \subset \mathcal{Y}$. The Pareto front is given by all the $y \in \mathcal{Y}_{ND}$. This Pareto front is convex and will be proven below. Fig. 1 is an illustration of the proof. It shows the decision space $\mathcal{X}$ and the objective space $\mathcal{Y}$. Given two points y_a and y_b along the Pareto front,

$$\begin{aligned} y_a &= Cx_a \in \mathcal{Y}_{ND} \\ y_b &= Cx_b \in \mathcal{Y}_{ND} \ , \end{aligned} \tag{8}$$

a point in-between can be found. For any $\lambda \in [0,1]$ let y_c be on the line between y_a and y_b, such that

$$\begin{aligned} y_c &= \lambda y_a + (1-\lambda) y_b \\ y_c &= \lambda C x_a + (1-\lambda) C x_b \\ y_c &= C(\lambda x_a + (1-\lambda) x_b) \\ y_c &= C x_c \\ y_c &\in \mathcal{Y} \ . \end{aligned} \tag{9}$$

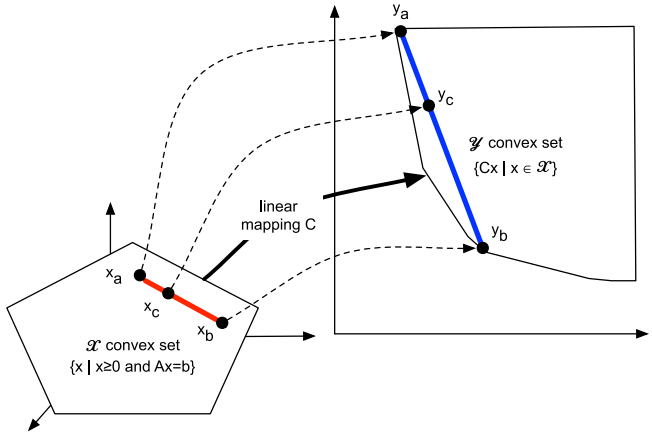


Fig. 1. Illustration of the proof of convexity: Showing the linear mapping from the convex set in the decision space to the convex set in the objective space.

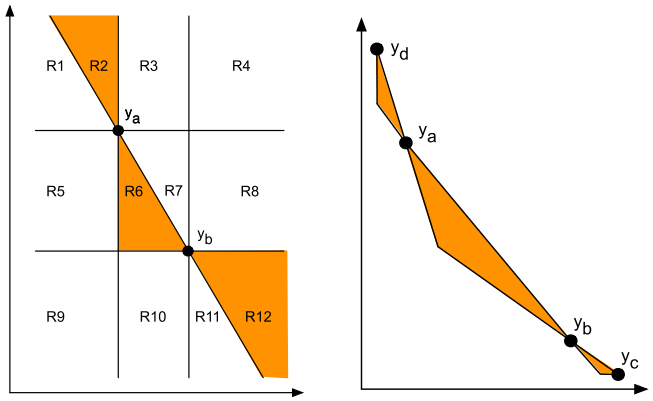
Therefore y_c is feasible and it is on the line between y_a and y_b so the Pareto front cannot have any concave regions. If there were any concave regions of the Pareto front then for some λ the point y_c would not be in the feasible region. It is important that x_c is a convex combination of x_a and x_b. This fact will be used to help fill gaps in the Pareto front in the convex fill algorithm described in Section 4.4. A more general version of this proof is available in [15].

The Pareto front for a linear objective function and convex constraint set is also connected [15]. This means that given one point in the Pareto front $\mathcal{Y}_{ND}$ all other points in the Pareto front can be found by taking infinitesimal steps along the Pareto front while never leaving the Pareto front. This is important because if one can find points along the Pareto front then it is possible to connect those points to form an approximation to the Pareto front.

3.2 Multiple Non-Dominated Solutions

It is desirable to tightly bound the Pareto front using algorithms that are computationally efficient and scale well as the problem size increases. Non-dominated solutions help to restrict the size of the regions where the remaining Pareto front may exist. Given any optimal non-dominated solution, the Pareto front does not exist in the region to the top right nor to the bottom left of the non-dominated solution. When given any two non-dominated solutions there is more information about the Pareto front that can be extracted when considering them jointly than when considering each individually. Fig. 2a shows an example of two non-dominated solutions y_a and y_b. The orange regions in Fig. 2a show where the Pareto front can reside. The Pareto front cannot be in any of the unshaded areas. Regions 3, 4, and 8 are dominated by y_a and/or y_b so they cannot be in the Pareto front. Regions 5, 9, and 10 would dominate y_a and/or y_b but y_a and y_b are in the Pareto front so these regions also cannot contain the Pareto front. If the Pareto front were in regions 1, 7, or 11 then the Pareto front would not be convex thus they are excluded as well. The orange regions 2, 6, and 12 are the only regions where the Pareto front can reside.

With four non-dominated solutions, the region where the Pareto front can reside is reduced even further. Fig. 2b shows four non-dominated solutions. The orange regions



(a) two points (b) four points

Fig. 2. Given y_a, y_b, y_c, and y_d in the Pareto front only the orange shaded regions may contain the Pareto front. Considering two points together provide much more information than considering them independently. Four points provide much more information than considering only two points due to the convexity of the Pareto front.

show where the Pareto front can reside. For instance, the region between y_a and y_b is reduced due to the convexity requirement imposed by y_d and y_c. It can be shown that adding a fifth non-dominated solution outside of y_d and y_c would not reduce the region between y_a and y_b any further due to convexity of the Pareto front.

3.3 Inner and Outer Approximations

In Section 4, multiple Pareto front approximation schemes will be discussed. Some of these approximations form an inner approximation while others form an outer approximation. Fig. 3 shows an example of an optimal Pareto front along with inner and outer approximations for the linear vector optimization problem. The outer approximation is a polytope that encloses $\mathcal{Y}$. Some solutions in an outer approximation may not be feasible but it will encapsulate all the solutions. An inner approximation is a polytope that is fully enclosed by $\mathcal{Y}$. All solutions in an inner approximation are feasible solutions. The Pareto solutions, $\mathcal{Y}_{ND}$, only exist between the inner and outer approximations. Also shown in Fig. 3 are the nadir and utopia points that form the bounds on the objective space region of interest. To find the nadir and utopia points one must first solve the optimization problem for each objective individually. The nadir point is then found by selecting the maximum value of each objective. Likewise, the utopia point is found by selecting the minimum value of each objective. The nadir and utopia points will be used in the weighted sum and convex fill algorithms.

4 Pareto Front Generation

4.1 Introduction

Finding the Pareto front can be computationally expensive because it involves solving numerous variations of the optimization problem to find many optimal solutions. Most algorithms use scalarization techniques to convert the multi-objective problem into a set of scalar optimization problems. Major approaches of scalarization include the hybrid method [15], elastic constraint method [15], Benson's algorithm [16], [17], and Pascoletti-Serafini scalarization [18]. Pascoletti-Serafini scalarization is a generalization of

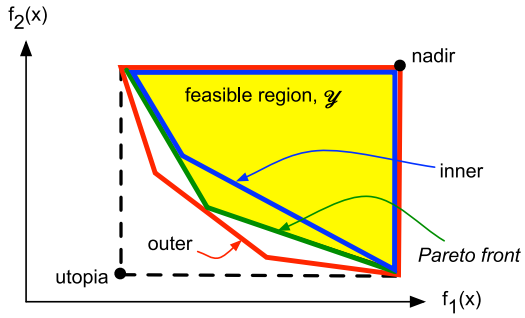


Fig. 3. Inner and outer approximation of the Pareto front: The feasible region is shown in yellow. The Pareto front is in the region between the inner and outer approximation polygons.

many common approaches such as normal boundary intersection, ϵ-constraint, and weighted sum. We will use the weighted sum algorithm in this work. The weighted sum algorithm can find all the non-dominated solutions for problems with a convex constraint set and convex objective functions, when enough weights are chosen [18]. Weighted sum is used for the linear convex problem in (4) to find all non-dominated solutions. A known issue with the weighted sum algorithm is that it does not uniformly distribute the solutions along the Pareto front. The clustering of solutions from weighted sum is mostly overcome by using the algorithm in Section 4.4.

Finding the optimal schedule for makespan alone is NP-Hard in general [19], thus finding the optimal (true) Pareto front is also NP-Hard. However, computing tight upper and lower bounds on the Pareto front is still possible. Specifically, a lower bound on a Pareto front is a set of solutions for which no feasible solution dominates any of the solutions in this set. An upper bound on the Pareto front is a set of feasible solutions that do not dominate any Pareto optimal solutions. The true Pareto front only exists between the lower-bound curve, an outer approximation, and the upper-bound curve, an inner approximation.

4.2 Weighted Sum

The weighted sum algorithm forms the convex combination of the objectives and sweeps the weights to generate the Pareto front. The first step is to compute the lower-bound solution for energy and makespan independently of each other. This is used to find the nadir and utopia points, y^{nadir} and y^{utopia} respectively. This is illustrated in Fig. 3. The next step is to compute the maximum change in each dimension as:

$$y^{\text{nadir}} - y^{\text{utopia}} = \begin{pmatrix} \Delta E_{LB} \\ \Delta MS_{LB} \end{pmatrix} . \tag{10}$$

The scalarized objective for the energy and makespan scheduling problem is then given by:

$$\underset{\boldsymbol{\mu},\, MS_{LB}}{\text{minimize}} \left(\frac{\alpha}{\Delta E_{LB}} E_{LB} + \frac{1-\alpha}{\Delta MS_{LB}} MS_{LB} \right) . \tag{11}$$

A lower bound on the Pareto front can be generated by using several values of $\alpha \in [0, 1]$. As the weights are changed, the objective function changes but the constraints all remain the same. This means that the optimal solution to

the LP in the prior step is still feasible in the new problem however possibly sub-optimal. To decrease the run-time the prior solution and the corresponding basis can be used to warm start the primal simplex algorithm [13]. In practice, this leads to significant savings in algorithm run time. Weighted sums will produce duplicate solutions (i.e., μ is identical for neighboring values of α). Duplicate solutions are removed to increase the efficiency of the subsequent algorithms. Each solution is rounded to generate an intermediate Pareto front. Rounding often introduces many duplicates that can be safely removed. Each integer solution is converted to a full allocation with the local assignment algorithm to create the upper bound on the Pareto front.

4.3 Non-Dominated Sorting Genetic Algorithm II

The NSGA2 [20] is an adaptation of the genetic algorithm (GA) optimized to find the Pareto front of a multi-objective optimization problem. Similar to all GAs, the NSGA-II uses mutation and crossover operations to evolve a population of chromosomes (solutions). Ideally, this population improves from one generation to the next. Chromosomes with a low fitness are removed from the population. The NSGA-II algorithm modifies the fitness function to work well for discovering the Pareto front. In prior work [4], the mutation and crossover operations were defined for this problem. The NSGA-II algorithm will be seeded in two ways in the following results. The first seeding method uses the minimum energy solution (only minimal energy when there is no idle energy), sub-optimal minimum makespan solution (from the min-min [5] algorithm), and a random population as the initial population. This is the original seeding method used in [4]. The second seeding method uses the full allocations from the local assignment algorithm as the initial population for the NSGA-II.

4.4 Convex Fill Algorithm

The weighted sum algorithm finds lower-bound solutions that are on the vertices of the objective space convex set $\mathcal{Y}$. As such, the weighted sum algorithm's solutions tend to be clustered because vertices of the polytope $\mathcal{Y}$ tend to be non-uniformly distributed in the objective space. This leaves large gaps between solutions in the Pareto front. Recall that Fig. 2 shows that as the distance between the known points along the Pareto front increase so does the size of the allowable region for the Pareto front. To better contain or bound the Pareto front, solutions are needed to help fill the gaps between the weighted sum solutions. The convex fill algorithm developed next is a very fast way to find these desired missing solutions.

Fig. 4 shows an example of the lower-bound curve as a thick purple line. Overlaid on the figure are the weighted sum algorithm's solutions as red circles. The white star shaped solution was not found by sweeping the weights for the weighted sum algorithm due to a fixed number of weights. Convex fill's solutions are shown as green triangles. These solutions fill in gaps between weighted sum solutions.

Recall from Section 3 that the convex combination of solutions is also a solution. The convex fill algorithm populates the gaps in the objective space by using this convexity property on the decision space. The convex fill algorithm uses all the unique lower-bound solutions from the weighted sum algorithm.

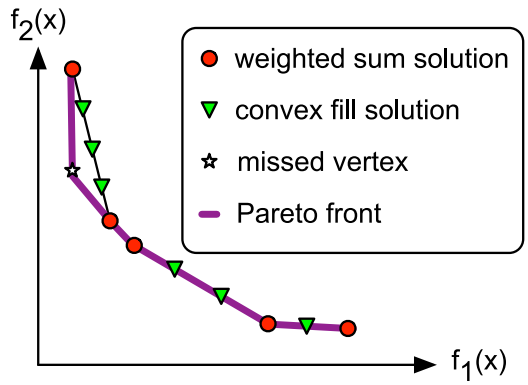


Fig. 4. Example solutions from the weighted sum and convex fill algorithms: Weighted sum's solutions are red circles and convex fill's additional solutions are green triangles. The Pareto front is the thick purple line. The white star represents a non-dominated solution that the weighted sum algorithm did not discover due to a limited number of weights that causes the neighboring convex fill solutions to not be a lower bound. The convex fill algorithm accurately approximates the solutions within the regions that the weighted sum algorithm missed.

Define the normalized objective value to be

$$\bar{y} = \frac{y - y^{\text{utopia}}}{y^{\text{nadir}} - y^{\text{utopia}}}. \tag{12}$$

For the energy and makespan problem, this becomes

$$\bar{y} = \begin{pmatrix} \frac{E_{LB} - E_{LB}^{\text{utopia}}}{\Delta E_{LB}} \\ \frac{MS_{LB} - MS_{LB}^{\text{utopia}}}{\Delta MS_{LB}} \end{pmatrix}. \tag{13}$$

The convex fill algorithm uses the L_1 norm as the measure of distance between outcomes in the normalized objective space. For the two-dimensional objective space, when N solutions are provided by weighted sum, the total distance is

$$\sum_{t=1}^{N-1} \| \bar{y}_t - \bar{y}_{t+1} \|_1 = 2 \ . \tag{14}$$

Let the parameter s be the maximum desired L_1 norm distance between two adjacent solutions in the normalized objective space. The convex fill algorithm will insert convex solutions when the distance between adjacent points from the weighted sum algorithm is greater than s to ensure the resultant spacing is at most s.

Algorithm 1 provides the pseudocode for our convex fill algorithm. It takes the list of lower-bound solutions X and a maximum desired spacing s and produces a list of solutions Z that has no gaps larger than s. This algorithm only works on vector optimization problems with a two-dimensional objective space. Our convex fill algorithm iterates over *adjacent solutions* in X. Two solutions are said to be adjacent if they are nearest to each other in the objective space. Practically, these adjacent solutions are found by first lexicographically sorting the solutions by their objective vectors. For a sorted set of solutions, the adjacent solutions are those that are consecutive in the list. The solutions from the weighted sum algorithm are already lexicographically sorted if α is

swept from 0 to 1. Let the distance between any two solutions be d, and let n be the number of solutions to be added between y_a and y_b to ensure maximum spacing of s. The convex combination of the pair of solutions and the objective values of the solutions are computed. Lastly, the new solution x is appended to the list Z.

Algorithm 1. Convex fill Algorithm

Require: X be the list of lower-bound solutions from the weighted sum algorithm
Require: s be the maximum desired spacing between solutions
$Z \leftarrow X$
for all adjacent pairs (x_a, x_b) in X **do**
  $d \leftarrow \| \bar{y}_a - \bar{y}_b \|_1$
  $n \leftarrow \lceil d/s \rceil - 1$
  for $t = 1$ **to** n **do**
    $\lambda \leftarrow \frac{t}{n+1}$
    $x \leftarrow (1-\lambda)x_a + \lambda x_b$
    $y \leftarrow (1-\lambda)y_a + \lambda y_b = Cx$
    $Z \leftarrow Z \cup \{x\}$
  end for
end for
return Z

Unlike solutions from the weighted sum lower bound, the solutions from the convex fill algorithm are not guaranteed to be a lower bound. This is because there is no guarantee that all solutions were found when performing the weighted sum sweep. If all vertices or solutions of $\mathcal{Y}$ are found, then convex fill will produce lower-bound solutions. Fig. 4 illustrates how a vertex that is not found by weighted sum, causes the convex fill algorithm's solutions to no longer be on the lower-bound curve. To use the convex fill algorithm for producing lower-bound solutions, an optimal algorithm such as Benson's algorithm is required [16]. Benson's algorithm for (4) is much slower than weighted sum.

To construct the full allocation from the lower bound, the recovery procedure described in the Section 4.2 is used. Results for the convex fill algorithm are presented in Section 5.5.

4.5 Pareto Front Solution Quality

Many approaches to quantitatively and qualitatively measure the quality of a Pareto front have been used in the literature [21], [22], [23]. One approach uses a measure of how well-spaced the solutions are in the objective space by computing the sample variance of the distance between solutions [24]. While this is useful in some cases it is not a good measure of the overall quality of an approximation to the Pareto front. A byproduct of the weighted sum algorithm described in Section 4.2 is that it produces many lower and upper-bound solutions that can be used to constrain the Pareto front to a small region. To quantitatively measure the performance of algorithms, we can compute the area of this region. The true Pareto front becomes more tightly bounded when the area of this region becomes smaller. Our approach is similar to the hyperarea difference in [25] that computes the area between the true Pareto front and the actual feasible solutions. Our approach uses the area between the lower bound (using convexity properties) and the feasible solutions (using dominance).

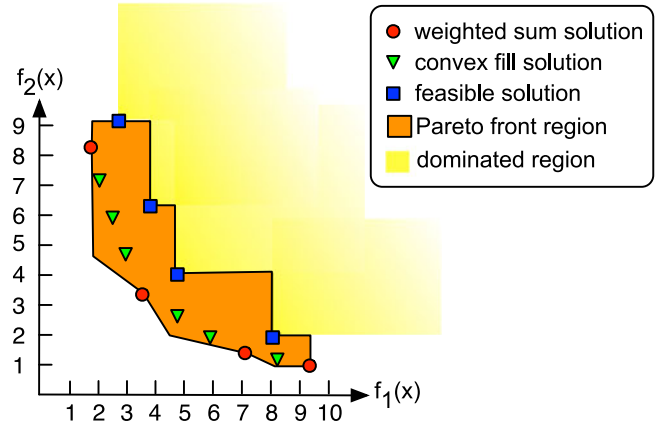


Fig. 5. An example of Pareto front bound area: The area of the orange polygon is a measure of the tightness of the bound on the Pareto front. The red circles are the solutions from the weighted sum algorithm and bound the lower part of the orange polygon using the properties of convexity. The blue squares are the fully feasible solutions and dominate the yellow shaded region to the upper right. The green triangles are the solutions from the convex fill algorithm, but they do not contribute to the orange polygon.

Computing the area of the region where the Pareto front can reside in a consistent manner is important. Fig. 5 shows this region as an orange polygon. The red circles are lower bound solutions that are obtained from the weighted sum algorithm. The lower edges of the polygon are defined by the convexity property as described in Section 3.2 and illustrated in Fig. 2. The blue squares are the fully feasible solutions and form the upper part of the polygon. A feasible solution dominates everything above and to the right that is indicated by the shaded yellow region in the figure. The extra solutions found by convex filling are shown as the green triangles. These solutions are neither part of the lower bound nor part of the upper bound, but they often contribute to unique feasible solutions that can help reduce the area of the orange polygon. For each dimension in the objective space, the maximum lower or upper bound solution is used to limit the polygon. For example, consider the lower bound solution at coordinate (9.5, 1). It has the largest objective value of all solutions in the f_1 direction and is used to limit the extent of the polygon in the f_1 direction. In the f_2 direction, the maximum point is the feasible solution at coordinate (3, 9). This point limits the polygon in the f_2 direction. Once all the vertices of the polygon are found then the area can be trivially computed to provide a measure of the quality of the solution [26]. Fig. 11 in Section 5.6 shows two Pareto front bound polygons computed from solutions of the LP-based algorithm with and without convex filling.

5 Results

5.1 Simulation Setup

ETC and **APC** matrices are needed to evaluate the algorithms. To generate these matrices, a set of five benchmarks executed over nine machine types are used to construct the initial matrices [27]. The benchmarks include raytracing, file compression, a gaming engine, real-time graphics rendering, and source code compilation. Not all the task types represent traditional HPC workloads, but the benchmarks serve as a good baseline of tasks that utilize the servers in

vastly different ways. The servers used include AMD and Intel architectures ranging from four to twelve cores. Five task types is rather small for realistic systems so we added task types that have similar statistical performance characteristics to the existing ones. The method found in [9] was used to construct the larger **ETC** and **APC** matrices. In this environment, there are 1,100 tasks comprised of 30 task types. The number of tasks per task type varies from 11 to 75 and was generated by the method used in [28]. There are nine machine types with four machines of each type for a total of 36 machines. A complete description of the environment and results are available in Appendix D, available in the online supplemental material. This environment will be referred to as the nine machine-type environment.

Some results to follow use an entirely different environment to show that the algorithms are not specific to a single configuration but instead are broadly applicable. This HPC system was previously used in [4] and will be referred to as the ten machine-type environment. Unlike the nine machine-type environment that is based on benchmarks, this environment is synthetically generated. The system has 50 machines selected from ten machine types. There are 1,000 tasks made from 50 task types. The **ETC** and **APC** matrices were generated randomly with the coefficient of variation (CoV) method described in [29].

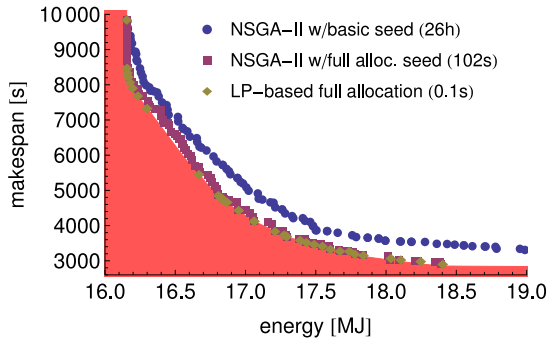
Unless noted otherwise, the simulations were performed on a mid-2009 MacBook Pro with a 2.5 GHz Intel Core 2 Duo processor. All the algorithms were implemented in C++ and optimized using our best effort. The COIN-OR CLP solver was used to solve the LP problems. The third party CLP library is also written in C++ [30]. The hardware being used for running the NSGA-II simulations is a 2013 Dell XPS'15 with an Intel i7-4702HQ 2.2GHz CPU. The NSGA-II code is implemented in C++.

The LP-based Pareto fronts are all generated with 1,000 evenly distributed weights. The weights are used in the weighted sum algorithm to parametrically sweep the Pareto front. Generally this leads to fewer than 100 full allocations depending on the particular problem.

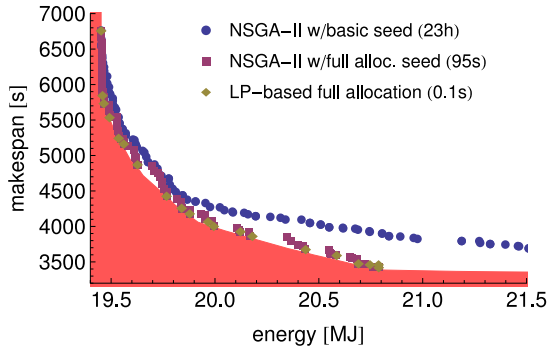
5.2 Pareto Fronts

Fig. 6 shows the lower bound and approximate Pareto fronts for four different environments. The LP-based lower bound is shown by the red shaded region. The figure shows the actual solutions as markers that are connected by lines for the NSGA-II algorithm and the LP-based algorithm. The legend shows the techniques associated with the markers in addition to the total algorithm execution time. All the systems have zero idle power consumption. The NSGA-II algorithm was allowed to run for one million generations when seeded with the basic seed. One thousand generations were used when seeded with the full allocation seed.

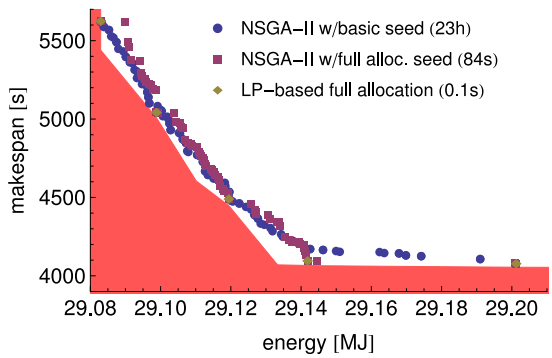
Fig. 6a shows the results for the nine machine-type environment. The lower bound and LP-based full allocation are nearly indistinguishable along the entire Pareto front. This means that the true Pareto front is tightly bounded even though it is unknown. The curve that is dominated (i.e., higher values in both makespan and energy) by all other curves is the set of solutions generated by the NSGA-II using the first seeding method. This means that it took NSGA-II



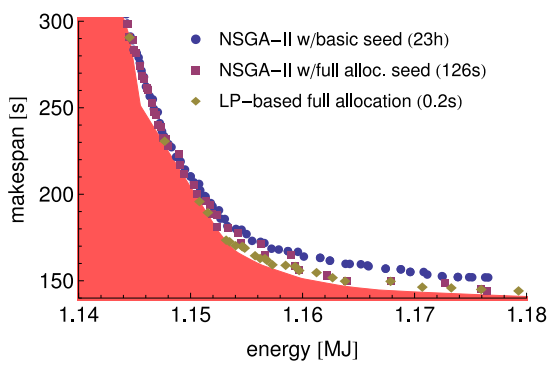
(a) nine machine-type environment



(b) six machine-type environment



(c) two machine-type environment



(d) synthetic ten machine-type environment

Fig. 6. Lower bound and approximate Pareto fronts: The region excluded by the lower bound from the LP is shaded in red and truly bounds the approximate Pareto fronts. The full allocation or upper bound is very near the lower bound so the Pareto front is tightly bounded. The times shown in the parentheses in the legend indicate the total time to compute the solution. Solution quality is rather poor with the NSGA-II using the original seed and expensive to compute, however the NSGA-II seeded with the full allocations produces a reasonable result, close to the full allocation, in much less time, but still is not as good as the full allocation in places.

over a day to find a set of solutions that are of poor quality in comparison to our technique that took 0.1 s. The set of red solutions are those obtained from seeding NSGA-II with the set of solutions produced by our local assignment algorithm. Seeding with the full allocation allows the NSGA-II to both converge to an improved front as well as decrease the run time. The NSGA-II attempts to evenly distribute the solutions along the Pareto front as can be seen in Fig. 6a. All the algorithms seem to perform well at minimizing energy, presumably because computing the optimal minimum energy solution is relatively easy compared to finding the optimal minimum makespan solution. To obtain the minimum energy solution, each task is assigned to the machine that requires the lowest energy to execute that task. Fig. 6a shows that all the algorithms produce good minimum energy solutions; however, for makespan there are significant differences in solution quality. The new LP-based algorithms produce better quality solutions in significantly less time.

A few different systems are used to further demonstrate the applicability of the LP-based Pareto front generation technique. Fig. 6b shows a system composed of just the first six machine types from the previous system, with six machines per type. Fig. 6c shows an even smaller system by taking only the first two machine types, with 18 machines per type. The total number of tasks, task types, and machines is unchanged. These figures show how the lower bound and upper bound still outperform the NSGA-II algorithm even when the number of machines types become small.

The results in Fig. 6d are based on the ten machine-type environment that is an entirely different environment than the nine machine-type environment. Even though this environment is very different from the previous environments, the LP-based algorithm produces a superior Pareto front in significantly less time.

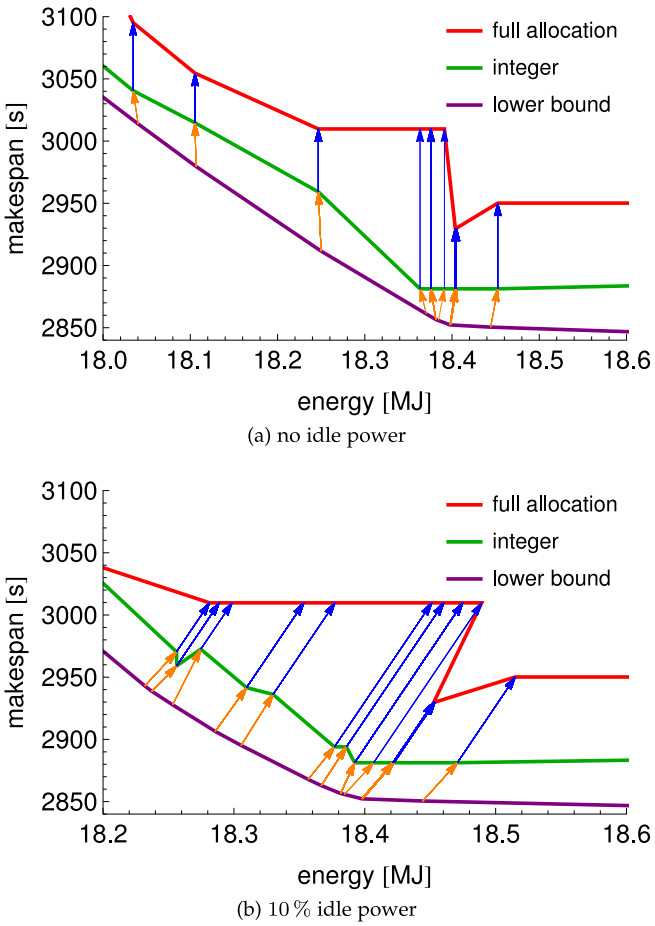


Fig. 7. Progression of solutions from lower bound to integer to upper bound without idle power (a) and with idle power (b). The dominated solutions are not being pruned from the solution set to illustrate how every real-valued solution is transformed to a feasible solution.

5.3 Solution Progression

To further understand the effects of the three phases of the proposed algorithm we can follow a set of solutions as they progress from the lower bound to the upper bound. Fig. 7 illustrates how the solutions progress through the three phases of the algorithm. Fig. 7a shows the progression without considering idle power consumption. This figure is a zoomed in version of a portion of Fig. 6a that details the progression of individual solutions for the nine machine-type environment. The lowest line represents the lower bound on the Pareto front. Each orange arrow represents a solution as it is rounded. In every case, the makespan increases while the energy may increase or decrease. The energy consumption can change during the rounding phase because tasks may become assigned to different machine types that may be more or less efficient compared to the original fractional assignment. As a given solution, μ, is rounded, machines will finish at different times, thus increasing the makespan. Each blue arrow represents a solution that is being fully allocated via the local assignment algorithm. The energy in this case does not change because the local assignment algorithm does not move tasks across machine types, thus the power consumption cannot change. The makespan increases are highly varying and depend on how well tasks in a machine type pack onto individual machines. The full allocation solution second from the right dominates the one on the far right. In this case the solution on the far right is removed from the estimate of the Pareto front in Fig. 6a. It is expected that some of the full allocations (before pruning based on dominance) will be dominated by other full allocations because there is no pruning based on dominance until the end of the algorithm.

Fig. 7b shows the progression of the solutions when considering idle power. The idle power consumption is set to 10 percent of the mean power for each machine type, specifically $APC_{\emptyset j} = \frac{0.1}{T}\sum_i APC_{ij}$. An idle power consumption of 10 percent is used to model the case when the server is powered off when not in use but the out-of-band management controller is still responsive to remote power on signals. See Appendix C, available in the online supplemental material, for experimental data and further explanation. As the makespan increases, more machines will be idle for longer, so the idle energy increases. The local assignment phase now negatively affects the energy consumption because it will typically have machines idle for some amount of time.

5.4 Idle Power Consumption

Fig. 8 shows the effect of idle power on the Pareto front for the nine machine-type environment. The curves show the lower bound on the optimal Pareto front with different

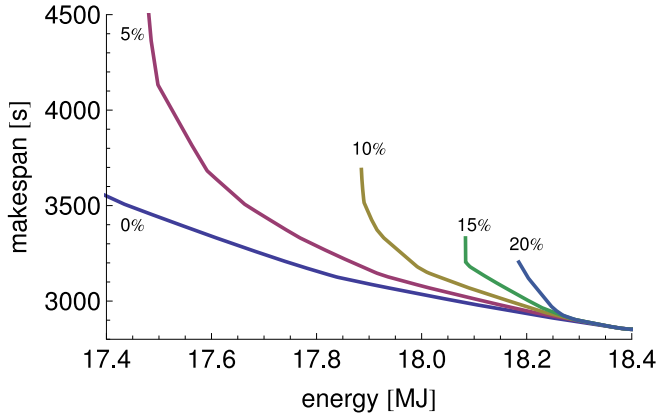


Fig. 8. Pareto front lower bounds when varying idle power: Idle power is increased in 5 percent increments as labelled on the figure. As idle power increases, the reward for minimizing makespan also increases. The curve without idle power is only partially shown.

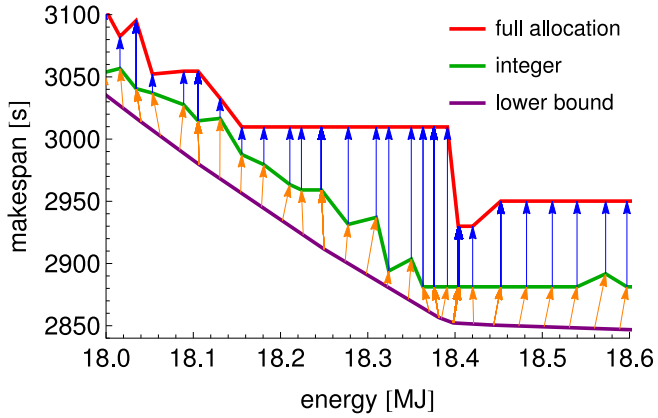


Fig. 10. Progression of solutions from lower bound to upper bound when using the convex fill algorithm: Convex filling produces unique integer and full allocations that tighten the Pareto front bounds compared to without convex filling in Fig. 7a.

percentages of idle power. The penalty for having a large makespan increases as the idle power increases because a large fraction of machines are idle for longer. The optimal energy solutions must now have a shorter makespan to reduce energy usage. This causes the Pareto front to contract in the makespan dimension and shift to the right slightly. As idle power usage approaches 100 percent, the problem degenerates to the single objective minimum makespan scheduling problem.

5.5 Convex Fill

Fig. 9 shows the solution front after convex filling while Fig. 6a is shown without the convex filling. Convex filling increases the run time only slightly, yet produces a much more complete Pareto front compared to using the weighted sum algorithm alone.

Fig. 10 shows how the solutions from the lower bound progress to the full allocation when using the convex fill algorithm with $s = 0.01$. Comparing this figure to Fig. 7a shows that the solutions added by the convex fill algorithm to the lower bound generate many unique integer and full allocation solutions. This allows the upper bound formed by the full allocations to be much tighter, as will be measured quantitatively in Section 5.6. A decision maker also would benefit from having fewer gaps in the Pareto front solutions when selecting an appropriate schedule. The additional run time of generating these extra solutions is negligible compared to the run time of the weighted sum algorithm.

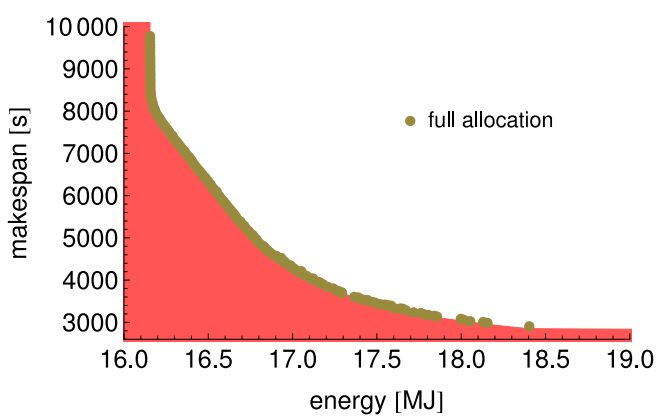
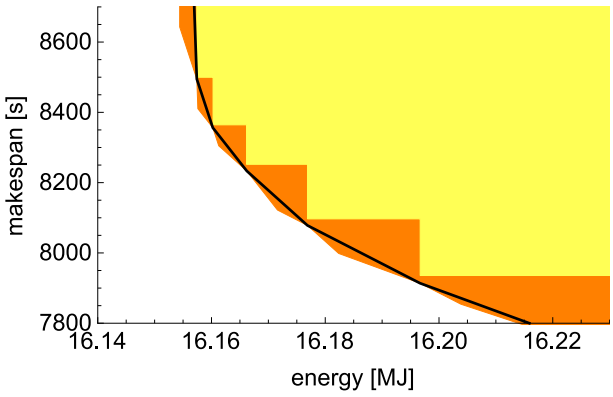


Fig. 9. Solutions after applying the convex fill algorithm: There are no more large spaces between full allocation solutions as compared to Fig. 6a.

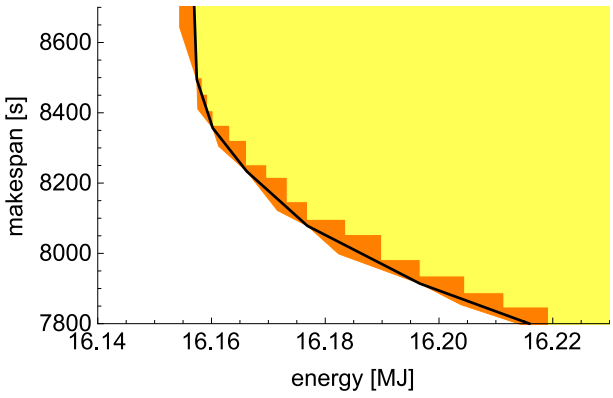
5.6 Area between Pareto Front Bounds

Using the algorithm detailed in Section 4.5 to compute the area between the inner and outer approximations, the quality of the different algorithms that generate bounds on the Pareto front can be quantified. Fig. 11 shows examples of the inner and outer approximations of the Pareto front for the nine machine-type environment. The orange area is the region where the true Pareto front can exist. The yellow region in the upper right is forbidden because full allocations have been found that dominate every solution in that region. The white part of the graph to the bottom left is also forbidden because there are no feasible solutions in that region. This white region is bounded by the outer approximation found from the lower-bound solution. The LP-based algorithm using just weighted sum is shown in Fig. 11a. The same region along the Pareto front after applying the convex fill algorithm is in Fig. 11b. The convex filling does not change the outer approximation but it does add more unique full allocations that greatly increases the area of the inner approximation making the bound on the Pareto front tighter.

Table 1 lists the area (in megajoule-seconds) that is between the inner and outer approximation polygons. When the area is small, the Pareto front is tightly bounded. The area is computed using the method in Section 4.5. Of the Pareto front generation algorithms discussed, only the LP-based algorithm produces an outer approximation or lower bound. The LP-based outer approximation is used for all the results shown in 1. The table shows four different algorithms for computing the inner approximation. The results are shown for the nine, six, two, and ten machine-type environments whose Pareto fronts are shown in Fig. 6. The NSGA-II with the basic seed can only very loosely bound the Pareto front. The LP-based algorithm bounds the Pareto front much more tightly than NSGA-II. However, running the NSGA-II algorithm as a post process to the LP-based algorithm does improve the quality of the bounds.



(a) LP-based



(b) LP-based with convex fill

Fig. 11. Inner and outer approximation polygons with and without convex filling: The orange region is where the Pareto front can exist. The convex fill algorithm greatly reduces the allowable area where the Pareto front can exist.

This is because the NSGA-II will find solutions that are between the seeded full allocations thus filling in the gaps and reducing the area. The convex fill algorithm is an alternative post process to the LP-based algorithm that executes extremely fast. The convex fill algorithm bounds the Pareto front the tightest for all environments considered here.

The lower bound can be tightened even further by using the technique described in Appendix B, available in the online supplemental material, at the cost of greater computation.

6 Computational Complexity

6.1 Analysis

A complete analysis of the scaling properties of the single objective minimum makespan scheduling problem are presented in [31]. Those results are summarized below and then extended for the full Pareto front generation problem.

Recall that T and M are the number of task and machine types respectively. The average case complexity of solving a single LP problem with the simplex algorithm is $(T+M)^2(TM+1)$. The complexity of the rounding algorithm is $\mathcal{O}(T(M \log M))$. Let $T_{\text{total}} = \sum_i T_i$ be the total number of tasks and $M_{\text{total}} = \sum_j M_j$ be the total number of machines. Assuming for the sake of analysis that tasks and machines are evenly distributed so $\forall j \ \sum_i \mu_{ij} \approx \frac{T_{\text{total}}}{M}$ and $M_j \approx \frac{M_{\text{total}}}{M}$. The local assignment algorithm has complexity $\mathcal{O}(MT \log T + T_{\text{total}} \log M_{\text{total}} - T_{\text{total}} \log M)$.

TABLE 1
Area between Bounds

	Machine Types in Environment			
Algorithm	Nine	Six	Two	Ten
NSGA-II	2,149	1,351	115	2.655
LP-based	684	339	63	1.011
NSGA-II seeded	436	306	53	0.851
LP with convex fill	231	238	38	0.762

The complexity of the overall algorithm to find both the lower bound and upper bound (full allocation) is driven by either the lower-bound algorithm or the local assignment algorithm. The complexity of the lower bound and rounding algorithms are independent of the number of tasks and machines. Those algorithms depend only on the number of task types and machine types. This is a very important property for large-scale environments. Millions of tasks and machines can be handled easily if the machines can be reasonably placed in a small number of homogeneous machine types and, likewise, tasks can be grouped by a small number of task types. Only the local assignment algorithm's complexity has a dependence on the number of tasks and machines. This phase is only necessary if a full allocation or schedule is required. The lower bound can be used to analyze much of the behavior of the HPC environment at a lower computational cost. Furthermore, the local assignment algorithm can be trivially parallelized because each machine type is scheduled independently.

When generating a Pareto front the lower-bound solutions are generated by re-solving a similar LP many times. The objective space of vector optimization problems are polytopes so they have a finite number of vertices. This means that there is a maximum number of solutions that can be found by the weighted sum algorithm because it is restricted to vertices. Usually there are a large number of duplicate solutions from weighted sum that can safely be removed thus reducing the computational cost of subsequent algorithms such as rounding and local assignment.

6.2 Results

To demonstrate the scaling properties of our Pareto front generation algorithm, a scaled up version of the nine machine-type environment was used to generate the larger environments used in this simulation. The number of machines per type was changed from 4 to 400 so there are now 3,600 machines. Tasks for each trial were generated by sampling the task type distribution with replacement. The mean of 50 trials is shown. For this set of simulations the convex filling algorithm was used to improve the quality of the Pareto front that was computed.

Fig. 12 shows the relative area between the inner and outer approximation polygons as a function of the number of tasks. The quality of the bound improves (i.e., relative area decreases) as the number of tasks to schedule increases. Fig. 13 shows the run time of the three phases of the algorithm as a function of the number of tasks. The Fig. 13a shows the time to run the weighted sum algorithm and solve all the resultant LPs. Corresponding to the analysis in

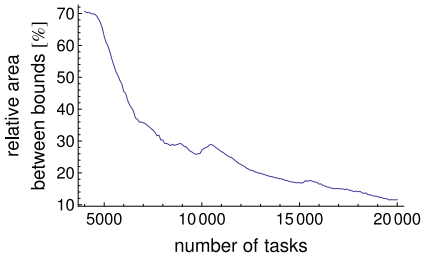


Fig. 12. Relative percent increase in area as a function of the *total number of tasks*: The quality of the solution improves as more tasks are used.

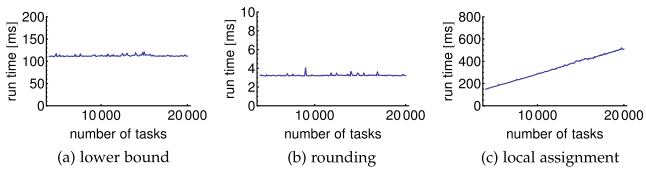


Fig. 13. Algorithm run time versus *total number of tasks*: Both the lower bound and the rounding algorithms are independent of the number of tasks. The local assignment, used to obtain the full allocation, is linearly dependent on the number of tasks.

Section 6.1, the weighted sum algorithm is independent of the number of tasks. The rounding algorithm is shown in Fig. 13b. Its runtime is also approximately independent of the number of tasks. Fig. 13c shows the local assignment and is the only phase of the algorithm that depends on the number of tasks. The dependency is linear which matches the analysis in Section 6.1.

The time required to solve the initial LP problem is on average 12.6 times more expensive than doing a single re-solve of the problem after perturbing the weights. This is because the LP problem changes only slightly in the objective function so only a few primal simplex steps are required to restore optimality.

7 Related Work

Techniques for generating Pareto fronts have been well studied (e.g., [4], [9], [10], [14], [20]). The LP-based approach in this paper achieves huge gains in run time and solution quality over prior methods by exploiting properties that are common to static scheduling problems. Most schedulers assign a single task to a machine at each iteration while our approach first assigns groups of tasks to groups of machines and then efficiently constructs the feasible schedule. Assigning groups of tasks is possible because there usually exist a relatively small number of task and machine types in practical systems. Our work also is focused on very large-scale systems and finding high quality solutions on average, whereas [19], [32] are concerned with worst-case performance of the scheduling algorithms. The energy and makespan problem is a specialization of the classic optimization problem of minimizing makespan and cost [19], [32].

While this paper deals with scheduling tasks to entire machines, the algorithms could also be applied to scheduling tasks to cores within a machine or across cores on many machines. The full allocation recovery algorithm we use is similar in nature to the algorithms presented in [33] that deal with scheduling on a single machine with deadlines to determine the best dynamic voltage and frequency scaling (DVFS) parameters to use to minimize energy as a secondary objective. An algorithm is presented in [34] that minimizes energy while constraining makespan and reliability to provide computationally efficient schedules for DVFS scheduling on identical processors.

In [12], the A^* search algorithm is used to assign tasks to machines considering task dependencies and communication constraints. This algorithm is very expensive for large numbers of tasks because the algorithm's branching factor is on the order of the number of machines and the depth is on the order of the number of tasks.

NSGA-II based approaches to find the energy and makespan Pareto front are in [4], [10] without the use of task and machine types. Other algorithms exist in the literature that may perform differently than NSGA-II such as the strength pareto evolutionary algorithm (SPEA2) algorithm [35].

Makespan and energy bi-objective optimization is also proposed in [36] via a mixed integer linear programming (MILP) formulation using the weighted sum algorithm to find solutions along the Pareto front. They present an adaptive algorithm that fills in the weighted sum solutions by solving additional MILP problems. They assign individual tasks to individual machines so scalability will suffer. An extension of their work that uses vector ordinal optimization to approximate the Pareto front is presented in [37].

The bi-objective problem for makespan and energy is solved in [38] for a homogeneous machines whereas our work models a heterogeneous set of machines. Their algorithm does not produce a Pareto front but rather a single solution trading off the objectives.

8 Conclusions

A highly scalable scheduling algorithm for the energy and makespan bi-objective optimization problem was presented. The complexity of the algorithm to compute the lower bound on the Pareto front was shown to be independent of the number of tasks and machines. Only the algorithm to compute the full allocation, that is computationally inexpensive and trivially parallelizable, is dependent on the number of tasks and machines. The quality of the solution also improves as the size of the problem increases. The LP-based Pareto front was compared to the solution found with the NSGA-II algorithm and shown to be superior in solution quality and algorithm run time for a variety of test environments. A post-process to the LP-based algorithm was developed that fills in solutions quickly using the convexity property of the relaxed problem. This was shown to further increase the quality of the Pareto front with a negligible increase in run time. A new approach for quantifying the quality of the Pareto fronts was developed and used to compare the different algorithms. These properties make this algorithm perfectly suited for very large-scale scheduling problems. This new LP-based Pareto front generation algorithm allows decision makers to more easily trade-off energy and makespan to reduce operating costs and improve efficiency of HPC systems.

This work could be extended by considering alternative scalarization techniques to potentially reduce the time required to compute the lower bound. Many of the LP

problems result in solutions that are identical, thus providing minimal information in forming the Pareto front. It is possible to avoid generating duplicate solutions by utilizing different scalarization techniques. The LP-based scheduling algorithm only takes a fraction of a second to compute a single schedule for a given bag-of-tasks so it is possible to use this scheduler for online batch-mode scheduling. Specifically, this algorithm can be used to schedule tasks as they arrive at the system by computing a schedule for all tasks waiting in the queue (as a batch) and recomputing the schedule when a task completes or a new task arrives.

Acknowledgments

This work was supported by the Sjostrom Family Scholarship, Numerica Corporation, the US National Science Foundation (NSF) under grants CNS-0905399 and CCF-1302693, a NSF Graduate Research Fellowship, and by the Colorado State University George T. Abell Endowment. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. A preliminary version of portions of this work has been presented in [39]. A special thanks to Mark Oxley and Bhavesh Khemka for their valuable comments.

References

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

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

[3] K. Cameron, "Energy oddities, part 2: Why green computing is odd," *Computer*, vol. 46, no. 3, pp. 90–93, Mar. 2013.

[4] 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 *Proc. 2nd Int. Conf. Adv. Commun. Comput.*, 2012, pp. 81–89.

[5] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, 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," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, Jun. 2001.

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

[7] V. Bharadwaj, T. G. Robertazzi, and D. Ghose, *Scheduling Divisible Loads in Parallel and Distributed Systems*. Los Alamitos, CA, USA: IEEE Comput. Soc. Press, 1996.

[8] 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," *J. Supercomput.*, vol. 57, no. 1, pp. 34–50, Jul. 2011.

[9] R. Friese, B. Khemka, A. A. Maciejewski, H. J. Siegel, G. A. Koenig, S. Powers, M. Hilton, J. Rambharos, G. Okonski, and S. W. Poole, "An analysis framework for investigating the trade-offs between system performance and energy consumption in a heterogeneous computing environment," in *Proc. 27th Int. Parallel Distrib. Process. Symp. Workshops, Heterogeneity Comput. Workshop*, 2013, pp. 19–30.

[10] R. Friese, T. Brinks, C. Oliver, H. J. Siegel, A. A. Maciejewski, and S. Pasricha, "A machine-by-machine analysis of a bi-objective resource allocation problem," in *Proc. Int. Conf. Parallel Distrib. Process. Technol. Appl.*, 2013, pp. 3–9.

[11] M. K. Dhodhi, I. Ahmad, A. Yatama, and I. Ahmad, "An integrated technique for task matching and scheduling onto distributed heterogeneous computing systems," *J. Parallel Distrib. Comput.*, vol. 62, no. 9, pp. 1338–1361, Sep. 2002.

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

[13] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*, 1st ed. Belmont, MA, USA: Athena Scientific, 1997.

[14] V. Pareto, *Cours d'economie Politique*. Lausanne, Switzerland: F. Rouge, 1896.

[15] M. Ehrgott, *Multicriteria Optimization*. Secaucus, NJ, USA: Springer-Verlag, 2005.

[16] H. Benson, "An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem," *J. Global Optim.*, vol. 13, no. 1, pp. 1–24, 1998.

[17] A. Löhne, *Vector Optimization with Infimum and Supremum* (Vector Optimization). Berlin, Germany: Springer, 2011.

[18] G. Eichfelder, *Adaptive Scalarization Methods in Multiobjective Optimization*. Berlin, Germany: Springer, 2008.

[19] K. Jansen and L. Porkolab, "Improved approximation schemes for scheduling unrelated parallel machines," *Math. Oper. Res.*, vol. 26, no. 2, pp. 324–338, 2001.

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

[21] Y.-W. Leung and Y. Wang, "U-measure: A quality measure for multiobjective programming," *IEEE Trans. Syst., Man Cybern., Part A: Syst. Humans*, vol. 33, no. 3, pp. 337–343, May 2003.

[22] B. Lacevic and E. Amaldi, "On population diversity measures in euclidean space," in *Proc. IEEE Congr. Evol. Comput.*, Jul. 2010, pp. 1–8.

[23] S. L. Faulkenberg and M. M. Wiecek, "On the quality of discrete representations in multiple objective programming," *Optim. Eng.*, vol. 11, no. 3, pp. 423–440, 2010.

[24] I. Y. Kim and O. De Weck, "Adaptive weighted-sum method for bi-objective optimization: Pareto front generation," *Struct. Multidisciplinary Optim.*, vol. 29, no. 2, pp. 149–158, 2005.

[25] J. Wu and S. Azarm, "Metrics for quality assessment of a multiobjective design optimization solution set," *J. Mech. Design*, vol. 123, no. 1, pp. 18–25, 2001.

[26] P. Bourke, C. Fuhrman, and A. Pons, "Calculating the area and centroid of a polygon," 1988, http://local.wasp.uwa.edu.au/~pbourke/geometry/polyarea

[27] (2013, May). Intel core i7 3770k power consumption, thermal [Online]. Available: http://openbenchmarking.org/result/1204229-SU-CPUMONITO81

[28] B. Khemka, R. Friese, L. D. Briceño, H. J. Siegel, A. A. Maciejewski, G. A. Koenig, C. Groer, G. Okonski, M. M. Hilton, R. Rambharos, and S. Poole, "Utility functions and resource management in an oversubscribed heterogeneous computing environment," *IEEE Trans. Comput.*, vol. 64, no. 8, pp. 2394–2407, Sep. 2014.

[29] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang J. Sci. Eng., Special Tamkang Univ. 50th Anniversary Issue, Invited*, vol. 3, no. 3, pp. 195–208, 2000.

[30] (2013, Mar.). Coin-or clp [Online]. Available: https://projects.coin-or.org/Clp

[31] K. M. Tarplee, R. Friese, A. A. Maciejewski, and H. J. Siegel, "Scalable linear programming based resource allocation for makespan minimization in heterogeneous computing systems," 2015, http://www.sciencedirect.com/science/article/pii/S0743731515001203

[32] D. B. Shmoys and E. Tardos, "Scheduling unrelated machines with costs," in *Proc. 4th Annu. ACM-SIAM Symp. Discrete Algorithms*, 1993, pp. 448–454.

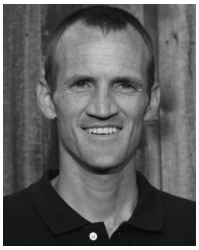
[33] D. Li and J. Wu, "Energy-aware scheduling for Frame-based tasks on heterogeneous multiprocessor platforms," in *Proc. 41st Int. Conf. Parallel Process.*, 2012, pp. 430–439.

[34] G. Aupy, A. Benoit, and Y. Robert, "Energy-aware scheduling under reliability and makespan constraints," in *Proc. 19th Int. Conf. High Perform. Comput.*, Dec. 2012, pp. 1–10.

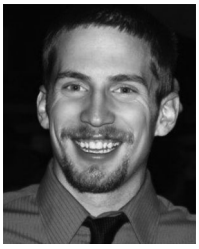
[35] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization," in *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, Eds. Athens, Greece: International Center for Numerical Methods in Engineering, 2001, pp. 95–100.

[36] S. U. Khan and C. Ardil, "A weighted sum technique for the joint optimization of performance and power consumption in data centers," *Int. J. Elect., Comput., Syst. Eng.*, vol. 3, no. 1, pp. 35–40, 2009.

[37] F. Zhang, J. Cao, K. Li, S. U. Khan, and K. Hwang, "Multi-objective scheduling of many tasks in cloud platforms," *Future Generation Comput. Syst.*, vol. 37, pp. 309–320, 2014.
[38] Y. He, F. Liu, H.-J. Cao, and C.-b. Li, "A bi-objective model for job-shop scheduling problem to minimize bag-of-tasksh energy consumption and makespan," *J. Central South Univ. Technol.*, vol. 12, no. 2, pp. 167–171, 2005.
[39] K. M. Tarplee, R. Friese, A. A. Maciejewski, and H. J. Siegel, "Efficient and scalable pareto front generation for energy and makespan in heterogeneous computing systems," in *Proc. Recent Adv. Comput. Optim.: Results Workshop Comput. Optim.*, 2015, pp. 161–180.

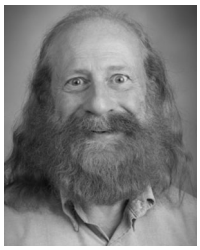


Kyle M. Tarplee received the BSEE and the master's degree from the University of California at San Diego and the PhD degree from Colorado State University in Electrical Engineering. He is a senior engineer at Numerica Corporation developing multi-target tracking algorithms for the Department of Defense. He has been the principal investigator on several projects. He is a senior member of the IEEE.



Ryan Friese received dual BS degrees in computer engineering and computer science from Colorado State University (CSU) in 2011. He received the MS degree in electrical engineering from CSU in the Summer of 2012. He is currently working toward the PhD degree at CSU. He is a United States National Science Foundation graduate research fellow. His research interests include heterogeneous computing as well as energy-aware resource allocation. He is a member of the IEEE.

Anthony A. Maciejewski received the BSEE, MS, and PhD degrees from The Ohio State University in 1982, 1984, and 1987, respectively. From 1988 to 2001, he was a professor of electrical and computer engineering at Purdue University, West Lafayette. He is currently a professor and department head of Electrical and Computer Engineering, Colorado State University. He is a fellow of the IEEE.



Howard Jay Siegel received two BS degrees from MIT, and the PhD degree from Princeton. He was appointed the Abell Endowed Chair Distinguished professor of electrical and computer engineering at Colorado State University in 2001, where he is also a professor of computer science. From 1976 to 2001, he was a professor at Purdue University, West Lafayette. He is a fellow of the IEEE and the ACM.

Edwin K. P. Chong received the BE degree with first class honors from the University of Adelaide, South Australia, in 1987, and the MA and PhD degrees in 1989 and 1991, respectively, both from Princeton University, where he held an IBM Fellowship. He joined the School of Electrical and Computer Engineering, Purdue University in 1991, where he was named a University Faculty Scholar in 1999. Since August 2001, he has been a professor of electrical and computer engineering and professor of mathematics at Colorado State University. His current research interests span the areas of stochastic modeling and control, optimization methods, and communication and sensor networks. He coauthored the best-selling book, *An Introduction to Optimization* (4th Edition, Wiley-Interscience, 2013). He received the US National Science Foundation (NSF) CAREER Award in 1995 and the ASEE Frederick Emmons Terman Award in 1998. He coreceived the 2004 Best Paper Award for a paper in the journal *Computer Networks*. In 2010, he received the IEEE Control Systems Society Distinguished Member Award. He is a fellow of the IEEE.