Robust Performance-Based Resource Provisioning Using a Steady-State Model for Multi-Objective Stochastic Programming

Kyle M. Tarplee, *Senior Member, IEEE*, Anthony A. Maciejewski, *Fellow, IEEE*, and Howard Jay Siegel, *Fellow, IEEE*

Abstract—Cloud computing has enabled entirely new business models for high-performance computing. Having a dedicated local high-performance computer is still an option for some, but more are turning to cloud computing resources to fulfill their high-performance computing needs. With cloud computing it is possible to tailor your computing infrastructure to perform best for your particular type of workload by selecting the correct number of machines of each type. This paper presents an efficient algorithm to find the best set of computing resources to allocate to the workload. This research is applicable to users provisioning cloud computing resources and to data center owners making purchasing decisions about physical hardware. Studies have shown that cloud computing machines have measurable variability in their performance. Some of the causes of performance variability include small changes in architecture, location within the datacenter, and neighboring applications consuming shared network resources. The proposed algorithm models the uncertainty in the computing resources and the variability in the tasks in a many-task computing environment to find a robust number of machines of each type necessary to process the workload. In addition, reward rate, cost, failure rate, and power consumption can be optimized, as desired, to compute Pareto fronts.

Index Terms—Planning, stochastic programming, vector optimization, high performance computing, scheduling, resource management, bag-of-tasks, many-task computing, energy-aware, heterogeneous computing, linear programming

1 Introduction

Some high-performance computing (HPC) users are turning to cloud providers to complete their work due to the potential cost effectiveness and/or ease of use of cloud computing. The ability to provision hardware on-demand from a predefined set of different machine types, known as *instance types*, is very powerful. In fact, a proof of concept cluster was built by Amazon Web Services from their high performance instance types composed of over 26,000 cores with nodes connected via 10 G Ethernet. This cluster ranked 101 on the Top 500 list for November 2014 [1]. The cloud has been successfully employed to process HPC jobs for actual scientific experiments [2]. Recent studies have shown that the performance of small and medium virtual clusters can compete with physical hardware [3]. Cloud infrastructure as a service (IaaS) providers [4] charge for the amount of time a virtual machine, known as an *instance*, is allocated (idle or active). This means that it is advantageous to terminate some or all instances once the workload has been processed. Leaving instances idle in the cloud is usually not cost effective. Once a new set of work needs to be processed, the decision of what instance types to start can be reevaluated each time, considering the size and composition of the workload and the current prices of the available instance types. Selecting the ideal number of instances of each instance type a user needs is challenging.

The approach to provisioning computational resources given in this paper not only applies to cloud resource provisioning but also to selecting physical machines to purchase for use within HPC systems. The goal for provisioning HPC systems is to determine how to originally select or upgrade a system in such a way that maximizes the performance of the resultant system while meeting specific requirements that often include a budget constraint.

The instance types available in the cloud have widely varying capabilities, by design, so that users can choose the resources that best match their workload and in doing so minimize the cost. For example, there is no need to provision high memory instance types if the workload does not require large amounts of memory. The cost for the smaller memory instance type will often be significantly less and provide nearly identical performance assuming all else is equal. Within a single IaaS provider, instance types vary in the amount of memory, number and type of CPUs, disk capacity and performance, and network performance. All of these properties of instance types affects the performance of the workload executing on the instances [5]. Due to the

- *K.M. Tarplee is with the Department of Physical Sciences and Engineering, Anderson University, Anderson, IN 46012 and the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523. E-mail: kyle.tarplee@colostate.edu.*
- *A.A. Maciejewski is with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523. E-mail: 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.*

Manuscript received 9 Feb. 2015; revised 9 May 2016; accepted 27 Aug. 2016. Date of publication 12 Sept. 2016; date of current version 29 Nov. 2019. Recommended for acceptance by I. Raicu, I. Foster, Y. Zhao, and J. Wozniak. Digital Object Identifier no. 10.1109/TCC.2016.2608345

availability of heterogeneous resources, IaaS is inherently a heterogeneous computing (HC) system.

This paper focuses on bag-of-tasks or many-task computing (MTC) workloads composed of a large number of many independent tasks. Each task is processed on a single machine. Bag-of-tasks workloads are commonly run on MTC systems [6].

In MTC and high-throughput computing (HTC), the usual goal is to maximize the number of completed tasks or jobs per unit time. In this research, a steady-state model of MTC is presented and used to formulate a linear optimization problem that can be used to optimize the number of tasks completed per unit time. In this work, types of tasks are assigned different rewards for completing. The *reward rate* is the reward earned per unit time by the system. In some situations, maximizing solely the reward rate is not desirable. Sometimes a conflicting objective such as the upgrade cost should be optimized along with the reward rate. The optimization problem we pose has multiple objectives from which any subset can be chosen, or new objectives added, as needed. The four objectives in the multi-objective optimization problem are (1) the amount of reward earned per unit time, called the reward rate, (2) the net cost to upgrade the system, called the upgrade cost, (3) the number of machine failures per unit time, called the failure rate, and (4) the total machine power consumption.

When making a purchasing decision not all the information is necessarily available nor is the information perfectly accurate. For example, arrival rates of tasks or the performance of the machines might not be known perfectly. In fact, studies have shown that instances of the same instance type can vary significantly in performance as discussed in Section 2. Often only the distributional assumption can be made. That is, the probability distribution of key parameters is known but the actual value of the parameter is unknown. This uncertainty is incorporated in our steady-state model. A multi-objective stochastic programming problem formulation is presented that incorporates the uncertainty in the parameters. Stochastic programming techniques are applied to this provisioning problem to handle uncertainty.

In summary, the contributions of this paper are:

1) the formulation of an energy-aware steady-state model for MTC,
2) the design of a linear optimization problem for resource provisioning (in the cloud or physical hardware procurement),
3) a model of uncertainty and a procedure for fitting this model,
4) a stochastic programming formulation that combines the steady-state model and the uncertainty model,
5) an orthogonal weighted sum algorithm for generating Pareto fronts that illustrates potential trade-offs between conflicting objectives, and
6) a performance evaluation of the stochastic programming formulation.

The rest of this paper is organized as follows. First some related work is given in Section 2. The steady-state model of MTC is in Section 3. Section 4 presents the model of uncertainty and Section 5 details the stochastic programming formulation. Quantitative measures for comparing the stochastic programming solution to other approaches is given in Section 6. Section 7 describes the heuristics implemented for comparison purposes. A technique to solve multi-objective stochastic programs is in Section 8. In Section 9, the simulation results are presented. Section 10 concludes this study and presents some ideas for future work.

2 Related Work

Scheduling resources on the cloud is not new. An interesting technique was presented in [7] that uses ordinal optimization to approximately solve a multi-objective optimization problem. The approach schedules tasks to virtual clusters in the cloud. Scheduling preemptable tasks on cloud resources has been studied recently as well [8]. Their approach uses information from the actual execution times to improve the subsequent resource allocations. The HARNESS project is currently designing algorithms and implementing software to provision resources on the cloud that benefit from highly heterogenous resources such as hardware accelerators and SSDs [9]. The uncertainty-aware scheduling approach presented here could one day be incorporated into such tools to improve the provisioning of resources.

A case study of using the cloud for HPC applications is in [10]. They show that the performance degradation due to virtualization is low but the networking performance can become a performance bottleneck if one is not careful.

The issue of HPC cluster reliability is addressed in [11]. This paper forms a bi-objective optimization problem to schedule tasks onto the cluster that has machines that vary in reliability. The author tries to minimize the maximum of all task completion times, known as the *makespan*, and maximize the reliability. The reliability measure described in Section 3 is similar to this measure.

We consider, in part, minimizing power consumption due to the explosive growth of power consumption in data centers and HPC systems over recent years [12]. Energy usage is becoming a major operating cost that requires algorithms to consider this from the start.

A technique to automatically scale the number of resources up or down based on the dynamic arrival of workloads is presented in [13]. Their technique utilizes the inherit heterogeneity in the cloud offerings to select instance types. In a closely related paper [14], automatic scaling is addressed. The paper models the uncertainty in the execution times of the tasks but does not consider the heterogeneous aspects of IaaS.

Task scheduling is NP-hard so reasonable approximations to model the problem, and scalable algorithms for its solution, are sought [15]. To design a scalable algorithm we use a steady-state scheduling algorithm within the resource provisioning problem. A motivation for using steady-state scheduling is given in [16]. Our steady-state model is inspired by the Linear Programming Affinity Scheduling (LPAS) algorithm [17].

The performance variation within the cloud, from instance to instance, is high compared to traditional hardware as studied in [18], [19], [20]. In [18], the measured coefficient of variation (CoV) was up to 24 % for Amazon Web Service's EC2 instances. They also showed that the distribution of instance performance can be bi-modal. Co-location of instances on the same physical hardware can cause variation in performance of up to 2.5 times [20]. Another surprising finding is that IO contention between VMs can cause upto a factor of 5 in

performance degradation [19]. In [21], some uncertainty in the performance of an instance was taken into account when determining both a set of instance types and task schedule via a particle swarm optimization problem.

Due to this inherit uncertainty in cloud instance performance and task arrivals, the stochastic nature of the problem cannot be ignored. Stochastic programming is the approach we take to rigorously account for the stochastic nature of the problem [22]. Stochastic programming has been used for capacity expansion in telecommunications systems [23] which is a similar application to computational resource provisioning.

3 Steady-State Model

Often there are millions of tasks and thousands of machines in large scale scheduling problems. To build scheduling and planning algorithms that have manageable run times for large problems, we use a steady-state formulation of the problem. This formulation focuses on the behavior of the system on average and avoids considering the scheduling of each task onto a particular machine. The steady-state model allows our algorithm's computational complexity to be independent of the number of tasks and machines in the problem. To build a compact steady-state model, we assume that the tasks of the workload can be grouped into a relatively small number of task types. All tasks belonging to a task type have similar run time and power consumption properties. This is often the case in scientific workloads. For instance, Monte Carlo simulations consist of a large number of tasks in the workload that can usually all be considered a single task type. Machines (or instances in the cloud) have a similar natural grouping called machine types (or instance types in the cloud). Task types and machine types will be used to reduce the computational complexity and enable a steady-state formulation of the problem.

The following steady-state model can be used for either determining which instance types to launch for cloud resource provisioning or determining which physical machines to purchase. Let there be T task types and M machine types. Let ETC_{ij} be the estimated time to compute (ETC) a task of type i running on a machine of type j. Likewise let APC_{ij} be the average dynamic power consumption of a task of type i running on a machine of type j. The static power consumption of a machine of type j is given by $APC_{\emptyset j}$. The **ETC** and **APC** matrices are commonly used in scheduling applications for HC systems. These matrices are often found by benchmarking the tasks. To model machines being turned off when not in use let $APC_{\emptyset j} = 0$.

This steady-state model can be used for either cloud provisioning or physical hardware purchasing by correctly defining the *cost* of the machines. Let β_j^B be the buying price or *cost* of a machine of type j. For the physical hardware purchasing problem, the *cost* is the total cost of ownership of the hardware (likely on a depreciation schedule) including support and maintenance. The *cost* for a cloud instance is its cost per unit time that a user pays for running an instance of type j. Let β_j^S be the selling price of a machine of type j. The selling price is only applicable to the purchasing of physical hardware. Typically $\beta_j^B > \beta_j^S \geq 0$.

Cloud IaaS providers often limit the number of instances a user can have without prior approval. Let M_j^{cur}, $M_j^{\min}$, and $M_j^{\max}$ be the current, minimum, and maximum number of machines of type j, respectively. Let $M^{\min}$ and $M^{\max}$ be the overall minimum and maximum number of machines, respectively. These parameters can be used to require the solution to adhere to those type of restrictions. When purchasing physical hardware these parameters may map to restrictions on the number of rack units available. Let M_j^B and M_j^S be the number of machines of type j to buy and sell, respectively. The total number of machines of type j is then $M_j = M_j^{\text{cur}} + M_j^B - M_j^S$. Due to the buying price being higher than the selling price, it makes no sense to buy and sell the same machine type.

Each task that is completed earns a reward based on the task type. Let r_i be the reward earned for completing a task of type i. The number of tasks of type i arriving per unit time is given by λ_i.

To compute the reward rate, failure rate, and power consumption, one must have a schedule that maps tasks to machines. In the steady state, it is sufficient to know the fraction of time a task of type i is running on a machine of type j. Let p_{ij} be this fraction of time. The matrix $\mathbf{p}$ is referred to as the schedule as it denotes the fraction of time each task type will be running on each machine type.

Sometimes it is useful to control the system failure rate. Machine failures when not executing a task are ignored as they have little consequence. Let ν_j be the failure rate of a machine of type j then the overall system failure rate is $\sum_i \sum_j \nu_j M_j p_{ij}$.

The optimization problem to determine the number of machines to buy and sell (i.e., $\mathbf{M^B}$ and $\mathbf{M^S}$) and the resultant schedule (i.e., $\mathbf{p}$) when maximizing the reward rate is given by:

$$\underset{\mathbf{M^B},\mathbf{M^S},\mathbf{p}}{\text{maximize}} \quad \sum_i r_i \sum_j \frac{1}{ETC_{ij}} M_j p_{ij} \tag{1a}$$

$$\text{subject to :} \quad \forall j \quad M_j^B \geq 0,\ M_j^S \geq 0 \tag{1b}$$

$$\forall i \quad \sum_j \frac{1}{ETC_{ij}} M_j p_{ij} \leq \lambda_i \tag{1c}$$

$$\forall j \quad \sum_i p_{ij} \leq 1 \tag{1d}$$

$$\forall i, j \quad p_{ij} \geq 0 \tag{1e}$$

The optimization problem in (1) will maximize the reward earned per unit time, namely the *reward rate*. The number of machines to buy and sell is required to be nonnegative, (1b). The arrival rate constraint is given by (1c). The machine utilization constraint, (1d), ensures that machines work no more then 100 % of the time on processing tasks. The last constraint, (1e), ensures that the fraction of time a machine is processing tasks is nonnegative.

The optimization problem in (1) has a non-linear objective (1a) and a non-linear constraint (1c) due to the terms that contain $M_j p_{ij}$. Both M_j and p_{ij} are decision variables in

the optimization problem. Solving non-linear optimization problems is considerably more computationally expensive than linear optimization problems. Fortunately, this non-linear problem can be transformed into an equivalent linear optimization problem. One can replace $M_j p_{ij}$ with a single variable, $\tilde{p}_{ij}$. The variable $\tilde{p}_{ij}$ can be interpreted as the effective number of machines of type j that are running tasks of type i. The constraint, (1d), can be rewritten as

$$\forall_j \sum_i \tilde{p}_{ij} \leq M_j, \tag{2}$$

because $M_j \geq 0$ and $p_{ij} \geq 0$.

After adding the objectives (e.g., cost, failure rate, and power) and constraints, and converting the non-linear problem in (1) to a linear optimization problem, the steady-state model based optimization problem becomes

$$\underset{\mathbf{M^B}, \mathbf{M^S}, \tilde{\mathbf{p}}}{\text{minimize}} \begin{pmatrix} -\sum_i r_i \sum_j \frac{1}{ETC_{ij}} \tilde{p}_{ij} \\ \sum_j M_j^B \beta_j^B - \sum_j M_j^S \beta_j^S \\ \sum_i \sum_j \nu_j \tilde{p}_{ij} \\ \sum_i \sum_j APC_{ij} \tilde{p}_{ij} + \sum_j APC_{\emptyset j} M_j \end{pmatrix} \tag{3a}$$

$$\text{subject to :}$$

$$\forall j \quad M_j^{\min} \leq M_j \leq M_j^{\max} \tag{3b}$$

$$M^{\min} \leq \sum_j M_j \leq M^{\max} \tag{3c}$$

$$\forall j \quad M_j^B \geq 0,\ M_j^S \geq 0 \tag{3d}$$

$$\forall i \quad \sum_j \frac{1}{ETC_{ij}} \tilde{p}_{ij} \leq \lambda_i \tag{3e}$$

$$\forall j \quad \sum_i \tilde{p}_{ij} \leq M_j \tag{3f}$$

$$\forall i, j \quad \tilde{p}_{ij} \geq 0 \tag{3g}$$

$$\sum_j M_j^B \beta_j^B - \sum_j M_j^S \beta_j^S \leq \beta \tag{3h}$$

$$\sum_i \sum_j \nu_j \tilde{p}_{ij} \leq \nu_{\max} \tag{3i}$$

$$\sum_i \sum_j APC_{ij} \tilde{p}_{ij} + \sum_j APC_{\emptyset j} M_j \leq P_{\max} \tag{3j}$$

In (3), all the objectives are to be minimized. The first objective in (3a) is the negative of the reward rate. If $\forall_i\ r_i = 1$, the reward rate objective function is the number of tasks completed per unit time, also known as the throughput of the system. The second objective is the upgrade cost. This is the cost of purchasing machines minus the cost of selling machines. The third objective is the system failure rate. The last objective is the power consumption of the system including static power consumption.

The constraints (3b) and (3c) limit the number of machines of each type and total, respectively. The constraints in the original problem correspond to (3d), (3e), (3f), and (3g). It is often beneficial to include bounds on the objective functions in multi-objective optimization problems. The linear optimization problem (3) has three additional constraints corresponding to a bound on the upgrade cost with budget β, a failure rate bound $\nu_{\max}$, and a power consumption bound $P_{\max}$ as constraints (3h), (3i), and (3j).

The optimization problem in (3) is a linear programming problem [24]. Single objectives of this problem can be quickly solved with either the simplex algorithm or interior point algorithm. A discussion of how to solve the multi-objective problem is presented in Section 8.

The computational complexity of optimally solving a single objective of the linear programming problem in (3) is the square of the number of constraints multiplied by the number of decision variables [24]. The number of non-trivial constraints is $4M + 4$ and the number of decision variables is $M + M + TM$. Therefore, solving the linear program via the simplex algorithm would have a computational complexity of $\mathcal{O}\Big((4M+4)^2(TM+2M)\Big)$. The computational complexity does not depend on the number of machines or tasks. Instead the complexity depends only on the number of types of machines and types of tasks used in the model.

4 Parameter Uncertainty Model

4.1 Overview

Uncertainty is a fact of life. Few parameters are known perfectly, so employing (3) is difficult because the effect of the unknown parameters on the solution is hard to ascertain. A model of the uncertainty in the parameters is needed to rigorously find and evaluate the solution to (3). A major source of uncertainty is the execution time and power consumption of the tasks running on the various machines. Section 4.2 describes a model that decomposes **ETC** and **APC** into parts. These parts are used in Section 4.3 to model the randomness in the system due to uncertainty in execution time.

4.2 Execution Time and Power Consumption Models

This model decomposes **ETC** into a linear combination of abstract computational operations. As we will see, these abstract operations need not map to physical (machine) instructions.

Let the number of abstract operation types be L, which is as small as possible to permit a sufficiently accurate model. Let η_{il} be the number of abstract operations of type l necessary to complete a task of type i. Let τ_{lj} be the time to complete one abstract operation of type l on a machine of type j. Then the $ETC_{ij} = \sum_l \eta_{il} \tau_{lj}$. In matrix form this is

$$\mathbf{ETC} = \boldsymbol{\eta}\boldsymbol{\tau}\ . \tag{4}$$

This model can represent task heterogeneity and machine heterogeneity within the **ETC** matrix. For $L > 1$, the model allows for arbitrary task machine affinity [25]. This minimal model is a mixture model that has the necessary properties for characterizing the execution time characteristics of an HC system. The model splits the **ETC** into two components. The first component is $\boldsymbol{\eta}$ that represents the size of the tasks in terms of operations and is only dependent on the task types' properties. The second component $\boldsymbol{\tau}$ is a property of only the machine types. This decomposition is useful in representing the components of **ETC** as correlated random

variables. In practice an accurate model can be found with a small value for L.

The **APC** matrix can be decomposed similarly. Let ψ_{lj} be the dynamic energy to execute an abstract operation of type l on a machine of type j. The energy of type l abstract operations is then $\eta_{il}\psi_{lj}$. The total energy can be computed as $\sum_l \eta_{il}\psi_{lj}$. The total average dynamic power consumption is

$$APC_{ij} = \frac{\sum_l \eta_{il}\psi_{lj}}{\sum_l \eta_{il}\tau_{lj}} \quad . \tag{5}$$

Equation (5) can be represented in matrix form, where division is element-wise, as

$$\mathbf{APC} = \frac{\boldsymbol{\eta}\boldsymbol{\psi}}{\boldsymbol{\eta}\boldsymbol{\tau}} \quad . \tag{6}$$

Generally only **ETC** and **APC** are available so the above model parameters, namely $\boldsymbol{\eta}$, $\boldsymbol{\tau}$, and $\boldsymbol{\psi}$, must be derived. These three parameter matrices have all nonnegative elements. The first step is then to compute the nonnegative matrix factorization (NNMF) of **ETC** to find $\boldsymbol{\eta}$ and $\boldsymbol{\tau}$ [26]. The NNMF is similar to the singular value decomposition but the NNMF produces nonnegative matrices for the decomposition. The energy can be written as $\mathbf{APC} * \mathbf{ETC} = \boldsymbol{\eta}\boldsymbol{\psi}$ so we can approximate $\boldsymbol{\psi}$ via least squares. In the E3 environment, described in Section 9.4, the relative errors from this approach for **ETC** and **APC** are 0.8 and 1.5 %, respectively.

Recall that for a task of type i running on a machine type j the run time is $\sum_l \eta_{il}\tau_{lj}$. Each term in the summation is the time that a particular abstract operation l is running. These abstract computational operations are found by decomposing the **ETC** matrix with a given number of abstract operation types L. These abstract operations represent resource bound operations within a computing system including CPU operations, disk and/or network I/O operations, and memory. The algorithm provides no mapping from abstract operation to an actual operation or a set of operations, however the algorithm does provide the correlation among task type run times.

4.3 Parameter Distributions

The dominant sources of uncertainty are generally from the arrival rates $\boldsymbol{\lambda}$, **ETC**, and **APC** parameters. If the probability distributions of the **ETC** and **APC** model are known then they can be used directly. If instead only the CoV of **ETC** is known then the following procedure can be used to estimate the variances of the elements of $\boldsymbol{\tau}$. Here we assume that all the uncertainty is in the machines and not in the tasks as was discussed in [18].

The mean and variance of the **ETC** entries are

$$\mathrm{E}[ETC_{ij}] = \sum_l \eta_{il}\mathrm{E}[\tau_{lj}] \tag{7}$$

$$\mathrm{Var}[ETC_{ij}] = \sum_l \eta_{il}^2\mathrm{Var}[\tau_{lj}] \tag{8}$$

assuming all the elements of $\boldsymbol{\tau}$ are independent. Using the definition of CoV, then substituting from (8), and finally squaring both sides we have the following three equations:

$$\sqrt{\mathrm{Var}[ETC_{ij}]} = \mathrm{CoV}_{ij}\mathrm{E}[ETC_{ij}] \tag{9}$$

$$\sqrt{\sum_l \eta_{il}^2\mathrm{Var}[\tau_{lj}]} = \mathrm{CoV}_{ij}\mathrm{E}[ETC_{ij}] \tag{10}$$

$$\sum_l \eta_{il}^2\mathrm{Var}[\tau_{lj}] = \mathrm{CoV}_{ij}^2\mathrm{E}^2[ETC_{ij}] \quad . \tag{11}$$

Converted to matrix form this is

$$\boldsymbol{\eta}^2\boldsymbol{\sigma}^2 = \mathbf{CoV}^2 * \mathbf{ETC}^2, \tag{12}$$

where the squares are element-wise. One can then compute $\boldsymbol{\sigma}^2$ via least squares. For the E3 environment described in Section 9.4 the relative error of this approach is 0.8 % for the variance of $\boldsymbol{\tau}$.

Nearly any probability distribution can be used within the stochastic programming formulation to model uncertainty. Even multi-modal distributions can be used. The parameters in the steady-state model are averages of execution times and arrival rates. There are no parameters that model temporal changes in arrival rate as it is a steady-state model. Thus, for our simulations we used uniform distributions for all uncertain parameters. Each uniform distribution is specified by a mean and a variance. To ensure the mean is preserved and that parameters only takes on positive values, the variance was capped as necessary.

5 Stochastic Programming

Stochastic linear programming is an extension of linear programming, where some of the coefficients in the objective and the constraints are random variables. For more information on stochastic programming see [27].

The particular stochastic program we use is the *recourse problem (RP)* given in standard form as

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \mathbf{c}^T\mathbf{x} + \mathrm{E}_{\boldsymbol{\xi}}[Q(\mathbf{x},\boldsymbol{\xi})] \\ \text{subject to :} \quad & \mathbf{A}\mathbf{x} = \mathbf{b} \\ & \mathbf{x} \geq \mathbf{0} \\ \text{where :} \quad & Q(\mathbf{x},\boldsymbol{\xi}) = \min_{\mathbf{y}} \ \mathbf{q}(\boldsymbol{\xi})^T\mathbf{y}(\boldsymbol{\xi}) \\ \text{such that :} \quad & \mathbf{T}(\boldsymbol{\xi})\mathbf{x} + \mathbf{W}(\boldsymbol{\xi})\mathbf{y}(\boldsymbol{\xi}) = \mathbf{h}(\boldsymbol{\xi}) \\ & \mathbf{y}(\boldsymbol{\xi}) \geq \mathbf{0}, \end{aligned} \tag{13}$$

where $\boldsymbol{\xi}$ is a random vector representing the uncertain parameters.

For the RP in (13), the first stage decision variable, $\mathbf{x}$ is a flattened version of $\mathbf{M^B}$ and $\mathbf{M^S}$. The second stage decisions, $\mathbf{y}$ are flattened versions of the schedule $\mathbf{p}$. The constraints of the steady-state model that contain random parameters or elements of $\mathbf{p}$, namely (3e), (3f), (3g), (3i), and (3j) are represented by $\mathbf{T}$, $\mathbf{W}$, and $\mathbf{h}$. The constraints without any random variables and thus no dependence on the scenarios, (3b), (3c), (3d), and (3h), define $\mathbf{A}$ and $\mathbf{b}$ in (13). The objective coefficients are separated in a similar way. The coefficients that are deterministic are incorporated into $\mathbf{c}$ and the coefficients that are random are incorporated into $\mathbf{q}$.

This linear RP is similar to a linear program except for the expectation of the value function, $Q(\mathbf{x},\boldsymbol{\xi})$, in the

objective. The RP in (13) is known as a *two-stage RP*. The optimization problem finds the optimal $\mathbf{x}$ that minimizes the sum of the linear function $\mathbf{c}^T\mathbf{x}$ and the expectation of $Q(\mathbf{x}, \xi)$.[1] The second stage optimization problem finds optimal $\mathbf{y}$ given a fixed realization of ξ. The random vector $\mathbf{y}$ is known as the recourse decision vector. This RP finds a robust solution for $\mathbf{x}$ in the sense that the objective value will on average be minimal when the optimal value of $\mathbf{x}$ is used. The solution $\mathbf{x}$ is robust to unknown values of the parameters ξ. The vector $\mathbf{x}$ is often referred to as a strategy of the RP.

There are many ways to solve stochastic programs. We will only describe a rather versatile approach to solving large scale stochastic programs that utilizes sample average approximation (SAA) to build the deterministic equivalent program (DEP).[2] The primary issue with stochastic programming is accurately computing the expectation in the objective function. The SAA approach uses many samples of ξ to compute the expectation as a sample average. Realizations of ξ are called scenarios. The process of creating scenarios is known as *scenario generation*. When using SAA, generating a reasonably small set of representative scenarios is important. Let there be K scenarios. For scenario k, let the probability of occurring be given by p_k. With SAA, the scenarios are generated by randomly sampling ξ according to its distribution, thus all the samples are equally probable. The expectation operator is linear and is applied to a linear function, namely $q^T\mathbf{y}(\xi)$. This lends itself to a linear program that is much larger, yet is equivalent to the stochastic program (in the limit as $K \to \infty$) given in (13). The DEP is given by

$$\begin{array}{lrcccl} \underset{\mathbf{x},\mathbf{y}_k}{\text{minimize}} & \mathbf{c}^T\mathbf{x} & + p_1\mathbf{q}_1^T\mathbf{y}_1 & \cdots & +p_K\mathbf{q}_K^T\mathbf{y}_K & \\ \text{subject to :} & \mathbf{A}\mathbf{x} & & & & = \mathbf{b} \\ & \mathbf{T}_1\mathbf{x} & + \mathbf{W}_1\mathbf{y}_1 & & & = \mathbf{h}_1 \\ & \vdots & & \ddots & & \vdots \\ & \mathbf{T}_K\mathbf{x} & & & + \mathbf{W}_K\mathbf{y}_K & = \mathbf{h}_K \\ & \mathbf{x}, & \mathbf{y}_1, & \cdots & \mathbf{y}_K & \geq 0 \end{array} \tag{14}$$

Each scenario (i.e., realization of the ξ) defines a set of matrices $\mathbf{T}_k$ and $\mathbf{W}_k$, and vectors $\mathbf{q}_k$ and $\mathbf{h}_k$. Each scenario also introduces a new vector of decision variables y_k into the problem.

The SAA is an unbiased estimator of the mean. In practice, it converges to the mean quickly in K. The DEP can have a large number of variables and constraints. For very large problems a technique called the L-method can be used to exploit the block structure of the constraint matrix to distribute the work of solving the linear program to many nodes [28].

The problem is broken into two coupled decisions. The first is what to provision or purchase, namely $\mathbf{M^B}$ and $\mathbf{M^S}$. Then the random variables in the problem are realized and the second decision, known as the recourse decision, can be made. For this work, the random variables are the arrival rates, execution times, and power consumption, but virtually any other parameter in the model can be converted to a random variable. The recourse decision involves selecting the schedule $\mathbf{p}$ that is optimal for the actual arrival rates, execution times, and power consumption of the tasks.

1. This formulation of the value function is risk neutral. Risk seeking and risk averse formulations are also possible.

2. The cloud resource provisioning problem is a two-stage stochastic program with relatively complete recourse [27].

6 Value of Information

To provide insight into different decision making approaches, we will evaluate other approaches besides the standard RP. Let $z(\mathbf{x}, \xi)$ be the objective value using the optimal second stage decision given the first stage decision $\mathbf{x}$ and a particular scenario ξ.

The wait-and-see (WS) solutions are found by waiting until ξ is realized then computing the optimal solution. Formally, the objective value of the WS solution is given by

$$\mathcal{WS} = \mathrm{E}_{\xi}\Big[\min_{\mathbf{x}} z(\mathbf{x}, \xi)\Big]. \tag{15}$$

This objective value is generally unachievable because it requires perfect information about the random variable. Thus, it provides an unachievable lower bound on the problem.

The RP is found by solving (13) or (14). The objective value of the RP is given by

$$\mathcal{RP} = \min_{\mathbf{x}} \mathrm{E}_{\xi}[z(\mathbf{x}, \xi)]. \tag{16}$$

This objective value is achievable and is the optimal strategy or solution to the problem.

An optimization problem that is often used in lieu of the RP is the expected value problem. This problem is also know as the mean value problem (MVP) because it uses only the means of all parameters to pose the optimization problem. Specifically the objective value for the MVP is

$$EV = \min_{\mathbf{x}} z\big(x, \mathrm{E}_{\xi}[\xi]\big). \tag{17}$$

Let $\mathbf{x}_{EV}$ be the optimal solution to the MVP in (17). To compare this solution to the WS and RP solutions one has to take the expected value over ξ of using $\mathbf{x}_{EV}$. Specifically this is

$$EEV = \mathrm{E}_{\xi}[z(\mathbf{x}_{EV}, \xi)]. \tag{18}$$

This equation uses the optimal second stage decision that is using a potentially suboptimal (based on the mean value of the parameters) decision for the first stage.

There are two standard measures used to compare these three standard approaches. The first is the expected value of perfect information (EVPI) defined as $EVPI = \mathcal{RP} - \mathcal{WS} \geq 0$. The second is the value of the stochastic solution (VSS) defined as $VSS = EEV - \mathcal{RP} \geq 0$. The EVPI is the amount the objective value for the RP, on average, would be lowered (i.e., improved) if the random vector ξ is known perfectly. VSS is the expected improvement in the objective value over the MVP if the uncertainty in the parameters is handled properly by using the RP.

7 Traditional Heuristic Strategies

Traditional strategies for purchasing hardware or provisioning cloud resources usually involve selecting the single machine that has the "best" desired property. Often the price and performance are used to select the machine to purchase.

For comparison, we define two common heuristics followed by one heuristic specific to our problem formulation. For each heuristic, the machine type to purchase, j^*, is selected by

$$\begin{aligned} \text{H1}: \quad & j^* = \arg\max_j \sum_i \frac{1}{ETC_{ij}} \\ \text{H2}: \quad & j^* = \arg\max_j \frac{1}{\beta_j^B} \sum_i \frac{1}{ETC_{ij}} \\ \text{H3}: \quad & j^* = \arg\max_j \frac{1}{\beta_j^B} \sum_i \lambda_i r_i \frac{1}{ETC_{ij}}. \end{aligned} \tag{19}$$

The three heuristics will be referred to as H1, H2, and H3. For the selected machine type, the next step is to find the maximum number of machines that do not violate any constraints (such as budget and power). The strategy is simply to let $M_{j^*}^B$ equal the maximum number of machines of that type that is feasible.

The first heuristic, H1 selects the machine that performs the best across all task types. Heuristic H2 uses the price and performance ratio to select the machine to purchase. Heuristic H3 weights the machine performance by the arrival rate and the reward for each task type. There are clear limitations of these heuristics in terms of flexibility and performance. H1 and H2 do not consider workload or differing reward between task types. None of the heuristics incorporate reliability or the option to potentially sell machines. The heuristics only select one machine type to purchase. As the results in Section 9 will show, typically the best solution is found by combining multiple machine types to match the load. None of the heuristics account for uncertainty in the parameters.

These strategies simply define the first stage of the problem. They do not describe how to schedule the tasks after such a purchasing strategy is executed. To provide a fair comparison, we use the optimal schedule from the MVP given that this particular strategy was chosen. Then we compute the expected value of the objective using (18) with $\mathbf{x}_{EV}$ replaced by the purchasing decision made by the heuristic.

8 Multi-Objective Stochastic Programming

8.1 Introduction

The RP derived from the base problem in (3) is a multi-objective stochastic programming problem. In this section, we extend the scalar stochastic programming problem described in Section 5 to the multi-objective case [29], [30].

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 [31]. When all objectives are to be minimized, a feasible solution x_1 dominates a feasible solution x_2 when

$$\begin{aligned} &\forall d \quad f_d(x_1) \leq f_d(x_2) \\ &\exists d \quad f_d(x_1) < f_d(x_2), \end{aligned} \tag{20}$$

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

There are many techniques for solving multi-objective optimization problems. For linear optimization problems, there are two primary approaches. The first is known as Benson's algorithm that iteratively refines the Pareto front [32], [33]. The second is a technique that converts the multi-objective problem into a set of scalar optimization problems through a process called scalarization. There are many scalarization techniques but most are specializations of Pascoletti-Serafini scalarization [34], such as the weighted sum algorithm. We use the weighted sum algorithm, described in Section 8.2, to compute the Pareto front for the multi-objective stochastic program.

8.2 Weighted Sum Algorithm

The weighted sum algorithm forms the positive convex combination of the objectives and then sweeps the weights to generate the Pareto front [35]. The optimization problem in (14) is linear and convex, thus by Theorem 3.15 in [36] the weighted sum algorithm can find all of the non-dominated solutions.

Consider a D-objective optimization problem. Let the weight vector be $\boldsymbol{\omega}$ that is used to combine the objectives. To avoid a degenerate objective function, exclude $\boldsymbol{\omega} = \mathbf{0}$ from the set of weights by imposing a somewhat arbitrary constraint that the $\sum_{i=1}^{D} \boldsymbol{\omega}_i = 1$. The vector $\boldsymbol{\omega}$ is in a $D-1$ dimensional linear subspace of $\mathbb{R}^D$.

The first step in the weighted sum algorithm is to compute the *utopia* and *nadir* points. Let the optimal solution vector for objective d be $\mathbf{q}^d = \arg\min f_d(\cdot)$. The dth element of the utopia and nadir points are then computed as

$$\forall_d \quad y_d^{\text{utopia}} = \min\big(f_d(\mathbf{q}^1), \ldots, f_d(\mathbf{q}^D)\big) = f_d(\mathbf{q}^d) \tag{21}$$

$$\forall_d \quad y_d^{\text{nadir}} = \max\big(f_d(\mathbf{q}^1), \ldots, f_d(\mathbf{q}^D)\big). \tag{22}$$

In other words, the utopia point is the best possible value for all objectives and the nadir point is the worst possible value from optimizing each objective individually. These two vectors are used to normalize the objective functions to better span the space. The utopia and nadir points also remove all units from the objective making the scalarized objective unitless. The normalized objective function is defined as

$$\mathbf{f}(\mathbf{x}) = \frac{\mathbf{f}(\mathbf{x}) - \mathbf{y}^{\text{nadir}}}{\mathbf{y}^{\text{nadir}} - \mathbf{y}^{\text{utopia}}}, \tag{23}$$

where the division is taken to be element-wise.

The second step in the weighted sum algorithm is traditionally performed by recursively combing the objectives while ensuring that the weights sum to one [35]. Let $\boldsymbol{\omega}^1 = 1$, then the recursion is

$$\forall d = \{2, 3, \ldots, D\} \quad \boldsymbol{\omega}^d = \begin{bmatrix} \alpha_{d-1}\boldsymbol{\omega}^{d-1} \\ 1 - \alpha_{d-1} \end{bmatrix}. \tag{24}$$

The final weight vector is $\boldsymbol{\omega}^D \in \mathbb{R}^D$. To sweep the space, each α_d is varied uniformly from 0 to 1. This approach produces duplicate weight vectors and performs non-uniform sampling in the subspace defined by $\sum_{i=1}^{D} \omega_i = 1$.

The orthogonal weighted sum algorithm finds an orthonormal basis (i.e., spanning set) for the null space of $\mathbf{1}_D$ and sweeps independently in the $D-1$ dimensional space

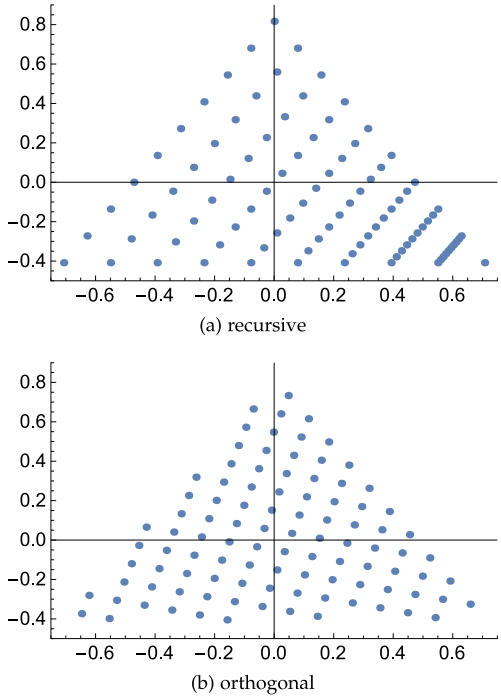


Fig. 1. Comparison of the recursive and the orthogonal weights drawn in the 2D plane that they span: The recursive algorithm unnecessarily puts more samples in the bottom right quadrant while the orthogonal algorithm more uniformly spans the space thus making better use of a limited number of samples.

defined by the basis vectors. Weight vectors with any negative components are dropped. This sweeps the subspace uniformly, therefore, it does not prefer any objective to any other. To ensure the whole space is swept, one must sweep α_d from $-\Delta$ to $+\Delta$ where $\Delta = \sqrt{1 - \frac{1}{D}}$.

The third step in the weighted sum algorithm is to solve the optimization problem for each weight computed in step two. Specifically, for each l solve

$$\min_{\mathbf{x}} \boldsymbol{\omega}_l^T \mathbf{f}(\mathbf{x}), \tag{25}$$

to find the Pareto front.

To compare the recursive and the orthogonal sweeping methods, Fig. 1 shows the weights in the subspace where the weights sum to one. The recursive algorithm in Fig. 1a uses more samples in the bottom right corner than it does elsewhere. That region is no more important than the rest of the space so it is wasted sampling. Using roughly the same number of points, Fig. 1b more uniformly distributes the samples. The orthogonal weighted sum algorithm was used to generate the Pareto fronts in Section 9.

8.3 Confidence Regions

While solving a single stochastic programming problem it is useful to know the quality of the solution. To increase the quality of the solutions, the number of generated scenarios K can be increased but only at the cost of increased run time of the linear programming solver.

Confidence intervals are often used as a measure of the quality of these types of algorithms. For multi-objective optimization, a multi-dimensional confidence region is necessary. By the central limit theorem, the sample mean will converge to the multivariate normal distribution as $K \to \infty$.

TABLE 1
ETC for E1

	M1	M2	M3
T1	1	3	100
T2	100	3	1.1

Let $\bar{\mathbf{y}}$ and $\mathbf{S}$ be the sample mean and unbiased covariance matrix of the samples of $Q(\mathbf{x}, \boldsymbol{\xi})$, respectively. Then the $100(1-\alpha)$ % confidence region of the true mean, $\boldsymbol{\mu}$, is defined by

$$(\bar{\mathbf{y}} - \boldsymbol{\mu})^T \mathbf{S}^{-1} (\bar{\mathbf{y}} - \boldsymbol{\mu}) \leq \frac{(n-1)p}{(n-p)n} F_{p,n-p}(1-\alpha), \tag{26}$$

where $F_{p,n-p}(\cdot)$ is the inverse CDF of the F-ratio distribution with p numerator and $n-p$ denominator degrees of freedom [37]. The boundary of the confidence region is an ellipse in 2D and an ellipsoid in 3D. In 1D, (26) collapses down to regular confidence intervals. Confidence regions will be plotted in Section 9.

9 Results

9.1 Overview

Three very different environments are used to analyze the behavior of the proposed algorithms for resource provisioning. The heuristic-based algorithms H1, H2, and H3, and the RP that uses stochastic programming are compared. The first environment is a small example used to illustrate the behavior of the algorithms. The second environment is a larger environment. The third environment was built based on benchmark data. A complete description of the source code, system parameters, and simulation results are available in the supplementary material to aid in reproducibility. The data is provided as (CSV) and JSON files with further details available in the accompanying README.txt file.

The steady-state model and the scenario generation are written in C++. To generate the DEP in (14) the Coin-OR Stochastic Modeling interface (SMI) is used [38]. The underlying linear programming problem is solved with the Coin-OR Linear Programming (CLP) solver [39]. CLP is a high quality open-source solver written in C++. All the simulations where run on an Apple MacBook Pro Mid 2014, 2.2 GHz Intel Core i7. The solver is single threaded so timing results are for one core.

To compare MVP, RP, and WS against the heuristics described in Section 7, one must be able to compute all the objectives including reward rate. The reward rate is a function of the steady-state schedule. To allow heuristics to perform as best as possible we use the optimal schedule from the steady-state model by solving a linear programming problem where the $\mathbf{M^B}$ is fixed by the heuristic and $\mathbf{M^S} = \mathbf{0}$. When computing the expected objective values the optimal schedule is used for each scenario.

9.2 Highly Heterogeneous Environment (E1)

This environment is composed of two task types and three machine types. The **ETC** matrix is given in Table 1. Machine types 1 and 3 are special purpose machines and machine type 2 is a general purpose machine. Machines of type 1 can execute tasks of type 1 rapidly but are slow to process tasks

TABLE 2
Solutions for E1

	H1	H2	H3	MVP	RP
M1	3.5	3.5	3.5	1.	1.9
M2	0	0	0	0	0
M3	0	0	0	2.5	1.6

of type 2. The reverse is true for other special purpose machine type (i.e., type 3).

The cost for each type of machine is one per unit time (e.g,$1/hour instance on AWS EC2 for a particular instance type). One task of each task type arrives (on average) every time unit. There are no pre-existing machines in the environment. The power and failure rate constraints are inactive. The only uncertainties in this environment are the arrival rates of the tasks. The arrival rate for tasks of type 1 is uniform from 0 to 2. For tasks of type 2 the arrival rate is uniform from $1 - 0.547$ to $1 + 0.547$. The budget is set to 3.5 so a total of 3.5 machines can be purchased.

Table 2 shows the number of machines of each type that the algorithms chose. All three heuristics select machine type 1 to buy because it has the highest machine performance (19). Machine type 3 is the other special purpose machine but it has a slightly lower machine performance, therefore, is not selected by the heuristics. The MVP and RP both select the special purpose machines but they differ on quantity of the machines.

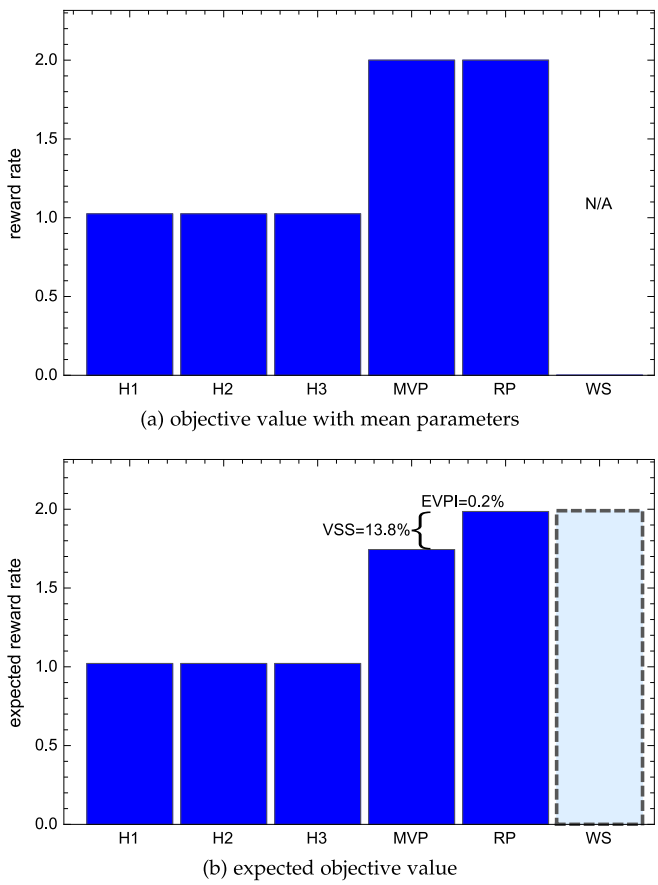


Fig. 2. Reward rates for various solutions for the E1 environment: (a) shows the reward rate computed with the mean of the parameters. (b) shows the expected reward rate over all uncertainty in the parameters.

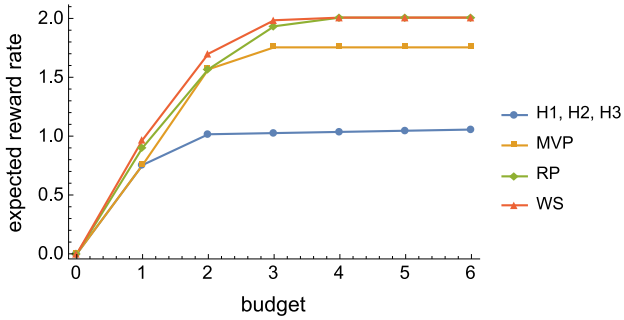


Fig. 3. Expected reward rate for different budgets for the E1 environment: The reward rate asymptotes at 2.0 with RP approaching WS. The other algorithms never reach the maximum reward rate.

Fig. 2 shows the performance of the three heuristics (H1, H2, and H3), MVP, RP, and WS. Fig. 2a shows the reward rates when using the mean value of the parameters for evaluating the reward rate based on the model in Section 3. The WS solution is not available because there is one solution (e.g., buying strategy) for each scenario and not one solution for all scenarios. This is the same reason why the WS algorithm is not realizable but can be used as an upper bound on performance. The heuristics all achieve a reward rate of about 1.0 because they can only efficiently process tasks of type 1, the reward per task is 1.0, and the mean arrival rate is 1.0. The MVP and RP purchase machines of different types and achieve the maximum achievable reward rate of 2.0.

Fig. 2b shows the expected value of the reward rate. This is the average reward one could expect if the given algorithm was employed to select the machines to purchase. All the solutions use the full budget that was allotted. The VSS and EVPI are also shown in the figure. The MVP and RP perform much better than the heuristics because the heuristics only choose one special purpose machine. The RP performs 13.8 % better than the MVP indicated by the VSS. This is due to RP selecting more of machine type 1 because task type 1 has a much larger uncertainty in the arrival rate compared to task type 2. In this example the EVPI is very small indicating that having fully realized parameters would not improve the solution any further than what the RP already found.

The budget has a strong influence on the expected reward rate. Fig. 3 shows the expected reward rate for the algorithms for different values of the budget. The results are averaged over ten Monte Carlo trials with 3,000 scenarios each. With no budget, there is no possibility of any reward. As the budget increases the reward increases until the maximum achievable reward rate is reached at 2.0. The heuristics flatten out at around 1.0 because they only process tasks of type 1. As the budget increases the heuristics buy more machines of type 1 but provide no value. The MVP starts hitting a limit on the reward rate once it has enough budget to purchase all the necessary machines to run the average number of tasks. The MVP does not consider the uncertainty in the arrival rates and so has no means of being robust against this uncertainty. The RP, on the other hand, fully considers the uncertainty in the arrival rates to make the best use of the budget.

9.3 Medium Sized Problem (E2)

Environment E2 has ten task types and five machine types. The number of abstract operations is set to two. This environment is a representative system and workload used to

TABLE 3
ETC for E2

	M1	M2	M3	M4	M5
T1	5	10	10	101	30
T2	15	30	30	300	90
T3	50	101	11	1010	303
T4	50	101	100	1010	303
T5	505	1010	1001	10100	3030
T6	15	30	12	300	90
T7	55	110	110	1100	330
T8	17	34	16	340	102
T9	6	11	10	110	33
T10	5	10	10	100	30

show some interesting properties of the different approaches. In this environment, the arrival rates, η, τ, and ψ are all stochastic with known means and a CoV of 100 %. There are ten machines of type 5 in the initial environment that can be retained or sold by the algorithms. The **ETC** matrix is given in Table 3.

To understand how the stochastic programming approach scales, Fig. 4 shows the confidence in the solutions and run time for solving the RP for various numbers of scenarios. The results shown in Fig. 4 are the average of ten Monte Carlo trials. The one-sided confidence interval in Fig. 4a shows that a ± 1.2 % confidence can be obtained after about 8,000 scenarios. It takes only about four minutes to compute that solution. These algorithms are meant to be used offline to aid in purchasing decision so these run times are reasonable.

The solutions from the different algorithms is in Table 4. The heuristics only select a single machine type to buy and retain all 10 machines of type 5. H2 and H3 both decided on the same machine to purchase so they have identical results in Fig. 5. The MVP solution chose mostly machines of type

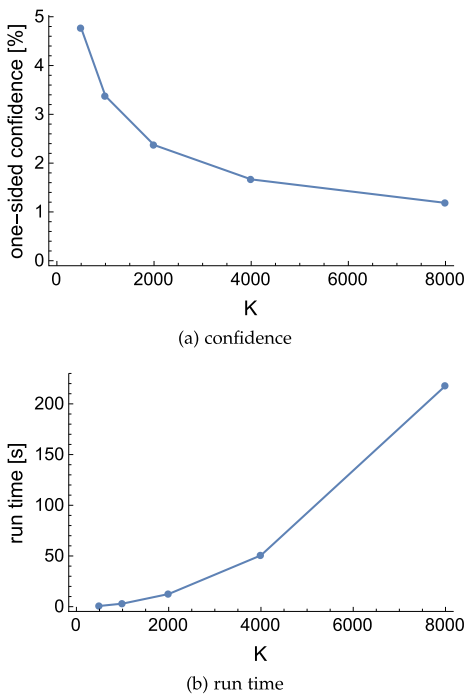


Fig. 4. Confidence interval (a) and algorithm run time (b) versus the number of scenarios for the E2 environment: The quality of the solution is again acceptable after only a relatively small number of scenarios.

TABLE 4
Solutions for E2

	H1	H2	H3	MVP	RP
M1	31.8	0	0	0	10.
M2	0	67.3	67.3	8.5	26.9
M3	0	0	0	32.	11.5
M4	0	0	0	0	0
M5	0	0	0	-10.	-6.7

3, but some type 2 machines were selected. The MVP solution decided to sell all 10 of the existing type 5 machines indicated by the minus sign. The RP picks mostly type 2 machines but some of type 1 and 3. Only 6.7 of the existing type 5 machines where sold.

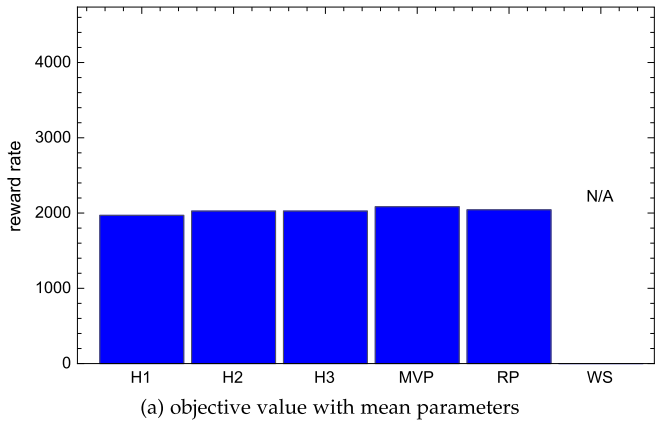
Similar to Fig. 2, the raw objective value and expected objective value are show in Fig. 5 for the E2 environment with 8,000 scenarios. When considering the performance of each algorithm using the mean of the parameters, Fig. 5a, the MVP performs slightly better than all other algorithms. It even appears to outperform the RP solution. This is misleading because the mean of the parameters is not a good measure of the expected performance. It is only one possible realization of the parameters of the problem.[3] Many more realizations or scenarios are possible that are completely ignored in this measure, however this is what is commonly used by practitioners. A better measure is the true expected reward rate shown in Fig. 5b. Due to the steady-state model and the stochastic model presented in Sections 3 and 4, computing the expected reward rate is easily accomplished. The RP's expected reward rate is over 13 % (given by VSS) higher than the MVP and the heuristics. If the parameters could be perfectly known at the time of making the decision then an additional 26.2 % (given by EVPI) improvement can be expected. The diversity in the machine types for the MVP and the RP allowed the resultant systems to better match the workload characteristics. In the case of the RP the workload was matched not only for the mean parameters but for nearly all possible realizations of the parameters. It is not clear how one could modify the heuristic based algorithms to better match the workload.

There are many possible scenarios that are possible in this environment. The RP is guaranteed to be the best performing solution on average but that does not imply it is the best for each possible scenario. Fig. 6 illustrates this by plotting the probability distribution of the relative improvement that RP has over the other algorithms (i.e. H1, H2, H3, MVP). The width of the glyphs represent the normalized probability density of the relative increase in reward rate. The RP can out perform the H1 heuristic by up to 300 %. For some scenarios, RP can also underperform H1 by 40 %. However, on average, RP produces significantly higher reward rates as seen by the positive mean of the distribution and by Fig. 5b.

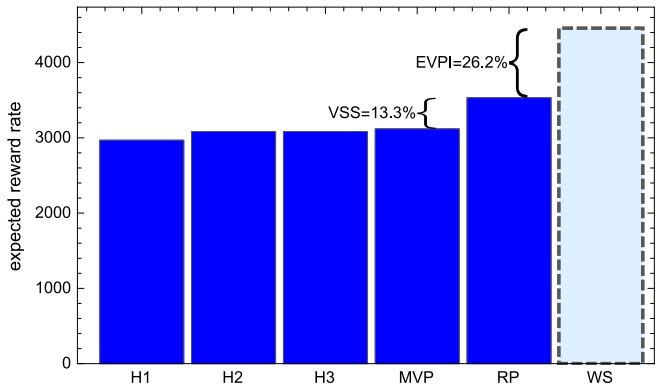
9.4 Benchmark Based Environment (E3)

This environment is based on a set of five benchmark applications that were run on nine different types of hardware.

3. This is assuming the mean parameters have non-zero probability density. For multimodal or discrete distributions the mean value of the parameters might not even be realizable.



(a) objective value with mean parameters



(b) expected objective value

Fig. 5. Reward rates for various solutions for the E2 environment: (a) shows the reward rate computed with the mean of the parameters. (b) shows the expected reward rate over all uncertainty in the parameters.

The execution time and power consumption was recorded for these systems [40]. 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. The method in [41] was then used to increase the number of task types to ten. This environment has nine machine types and ten task types that define the **ETC** and **APC** matrices. The **ETC** matrix is shown in Table 5. The algorithm in Section 4.2 was used to generate $\boldsymbol{\eta}$, $\boldsymbol{\tau}$, and $\boldsymbol{\psi}$. Based on [18], the CoV for the **ETC** elements was taken to be 25 %. The algorithm in Section 4.3 is used to generate the variance of $\boldsymbol{\tau}$. The number of abstract operations, L, is set to three. The arrival rates have a known mean with a CoV of 25 %. There is no power constraint in this

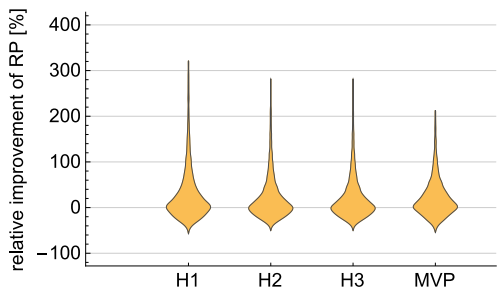


Fig. 6. Distributions of the relative improvement of RP over the other algorithms for the E2 environment: The RP for certain scenarios can perform as much as 300 % better than the heuristics, but RP also performs worse for some scenarios by as much as 40 %.

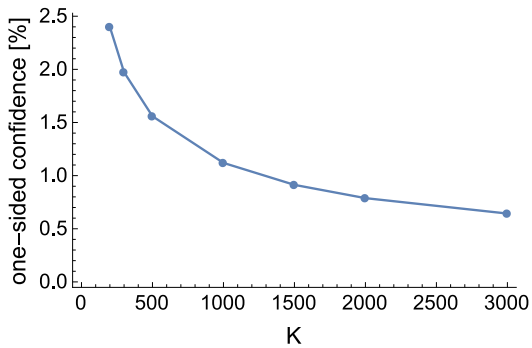
TABLE 5
ETC for E3

	M1	M2	M3	M4	M5	M6	M7	M8	M9
T1	57	28	72	45	41	19	27	28	26
T2	98	50	120	77	70	37	49	49	45
T3	463	303	362	342	314	311	303	290	264
T4	165	113	113	120	111	122	114	108	98
T5	167	91	185	129	118	74	90	88	81
T6	162	87	185	125	114	68	85	84	77
T7	45	22	57	36	33	15	22	22	20
T8	57	28	74	45	41	18	27	27	25
T9	59	36	54	44	41	34	36	35	32
T10	39	22	41	30	27	19	21	21	19

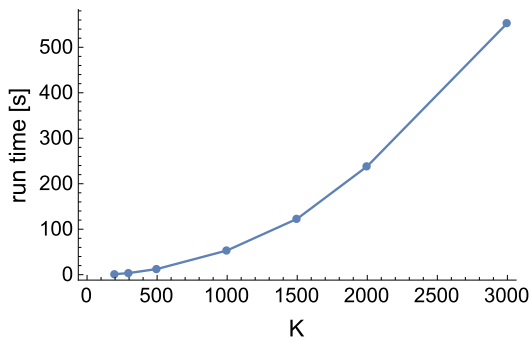
environment. The budget is $40,000 with the average machine costing $3,183, thus hundreds of machines are provisioned by the algorithms.

Fig. 7a shows the one sided confidence interval for different number of scenarios. The results shown in Fig. 7a are the average of ten Monte Carlo trials. At 1,500 scenarios, the error in the reward rate is under ± 1 %. Fig. 7b shows the corresponding run times for solving RP and takes less than a minute in all cases. Due to the (at least) quadratic growth of the run time w.r.t. the number of scenarios, it is important to use as few scenarios as possible. At 3,000 scenarios, the DEP is rather large with 63,011 constraints (i.e., rows) and 270,018 decision variables (i.e., columns). The constraint matrix in the DEP is very sparse so solving this large linear programming problem only consumes about 180 MB of memory.

For these simulations, MVP, RP, and WS are all trying to maximize reward rate but as a secondary objective they are



(a) confidence



(b) run time

Fig. 7. Confidence interval (a) and algorithm run time (b) versus the number of scenarios for the E3 environment: Computing an accurate solution is fast for this larger problem even though the number of machines being provisioned and scheduled is large.

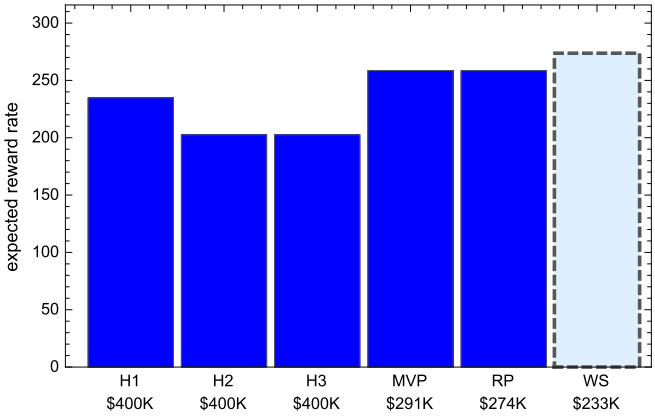


Fig. 8. Expected reward rates for various solutions for the E3 environment: The cost for each solution is indicated at the bottom of the figure. RP ties for the best performance with MVP however with a solution that has a lower cost.

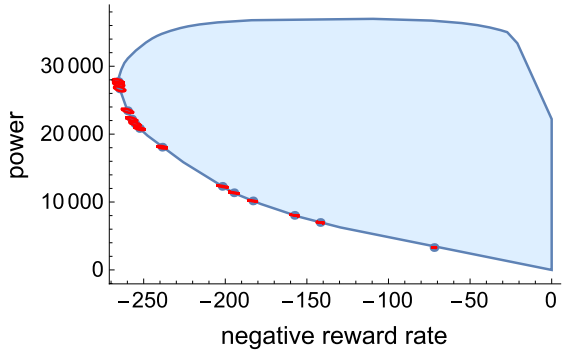


Fig. 9. Reward rate and power consumption Pareto front and feasible objective space for the E3 environment: The relatively small 95 % confidence ellipses are shown in red. The blue shaded region is the feasible region of the objective space. The blue dots are solutions that the weighted sum algorithm found.

also trying to reduce the cost. This is accomplished by weighting the reward rate with 1.0 and the upgrade cost objective with a small positive constant. Fig. 8 gives the expected negative of the reward rate for 3,000 scenarios. The upgrade cost is indicated below the labels at the bottom of the graph.

Table 6 shows the solution for each of the algorithms. All the proposed algorithms can be solved with integer constraints on these variables however at a huge computational cost. Even though the number of machines is fractional, a simple rounding policy can easily be applied to the solutions in Table 6 when integers are required with negligible effect on the reward rate.

All the heuristics spend the entire budget and still perform worse than MVP and RP. The MVP solution does not use all the available budget. In fact, it only used $291 K of the budget. This is because the MVP solution does not provision for the uncertainty in the arrival rates of tasks nor in the machine performance. In the MVP formulation, there is no benefit to buying more machines once the workload can be handled hence it does not use all the available budget. The MVP has extra degrees of freedom to improve the solution but has no practical way of determining how to best select the machines. From the MVP perspective, the algorithm is already achieving maximal reward. The RP takes the uncertainty of the arrival rates and the machines into account and can achieve the same performance as the MVP solution but at a lower cost of $274 K. The RP solution uses many different types of machines to reduce the cost and still achieve the optimal performance. The upgrade cost under the WS solution is the cost one would pay on average if they could select the machines after knowing precisely the execution time of each task type on all machine types and the arrival rates of the tasks.

TABLE 6
Solutions for E3

	H1	H2	H3	MVP	RP
M1	0	0	0	0	0
M2	0	168.8	168.8	0	0
M3	0	0	0	0	4.3
M4	0	0	0	0	0
M5	0	0	0	0	0
M6	121.2	0	0	0	12.7
M7	0	0	0	0	4.5
M8	0	0	0	100.	73.8
M9	0	0	0	0	1.1

9.5 Pareto Fronts

Pareto fronts are useful tools to quantify the trade-off between conflicting objectives. The weighted sum algorithm described in Section 8.2 is used to generate the Pareto front for the reward rate and power consumption objectives. Fig. 9 shows the Pareto front for the E3 environment with 1,000 scenarios. This took only eight minutes to compute. The solutions found by the weighted sum algorithm are blue dots. The 95 % confidence regions are shown as red ellipses centered at the blue dots. The confidence regions are relatively small. The light blue shaded region is the feasible objective space for this problem. Objective values outside this feasible space are not possible due to one or more of the constraints in the problem. The feasible region is computed by solving the RP with

$$\boldsymbol{\omega} = \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix}, \tag{27}$$

for values of θ from 0 to 2π. More details on computing the feasible regions for linear programming problems is available in [42].

10 Conclusion and Future Work

Stochastic programming is a powerful tool that can be applied to make robust decisions in the midst of the inherit uncertainty in computing systems in both IaaS provider clouds and traditional environments. The linear steady-state model and representative stochastic model enables the use of an efficient two-stage stochastic program for solving the machine provisioning problem. The new algorithms were compared to heuristic based algorithms in a few different environments. The heuristic approaches tend to perform poorly when considering their average performance. The RP produces the best quality solution on average compared to the heuristics and the MVP. The inherit uncertainty in the execution times and arrival rates necessitates algorithms that can incorporate random variables and their statistics. The RP can be used to reduce the upgrade cost for a computing system by exploiting the uncertainty in the environment while maintaining the optimal level of

performance. The multi-objective optimization problem can be used to quickly generate Pareto fronts.

In the future, we would like to adapt this model to include aspects of spot pricing and the uncertainty surrounding the price of the instances. We would also like to adapt these concepts to existing cloud provisioning tools and computational models. Risk averse formulations are also possible that minimize the expected reward rate and the variance of the reward rate.

Nomenclature

Symbol	Description
T	Number of task types
M	Number of machine types
i	Task type
j	Machine type
β_j^B	Buying price (cost) of a machine of type j
β_j^S	Selling price of a machine of type j
M_j^{cur}	Current number of machines of type j
$M_j^{\min}$	Minimum number of machines of type j desired
$M_j^{\max}$	Maximum number of machines of type j desired
M_j^B	Number of machine of type j to buy
M_j^S	Number of machine of type j to sell
M_j	Total number of machines of type j after buying and selling
r_i	Reward for completing a task of type i
λ_i	Arrival rate: Number of tasks arriving of type i per second
p_{ij}	Schedule: Fraction of time tasks of type i run on machines of type j
ν_j	Failure rate: Number of machines of type j that fail per second
L	Number of abstract computational operations
η_{il}	Number of abstract operations of type l in a task of type i
τ_{lj}	Run time [s] per abstract operation of type l on a machine of type j
ETC_{ij}	Run time [s] of a task of type i on a machine of type j
ψ_{lj}	Dynamic energy [J] consumed by an abstract operation of type l on a machine of type j
CoV_{ij}	Coefficient of variation for a task of type i on a machine of type j
$\boldsymbol{\xi}$	Random vector
$\mathbf{x}$	First-stage decision variable vector
$\mathbf{y}$	Second-stage decision variable vector
$Q(\mathbf{x}, \boldsymbol{\xi})$	Value function
$\mathcal{WS}$	Wait-and-see problem
$\mathcal{RP}$	Recourse problem
MVP	Mean value problem
EVPI	Expected value of perfect information
VSS	Value of the stochastic solution

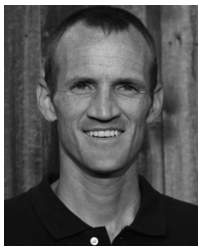
Acknowledgments

This work was supported by Numerica Corporation, the US National Science Foundation (NSF) under grants CNS-0905399 and CCF-1302693, 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 US NSF. A special thanks to Ryan Friese and Bhavesh Khemka for their valuable comments.

References

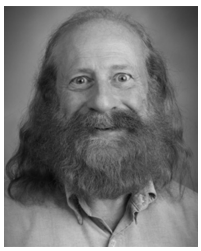
[1] Oct. [Online]. Available: http://www.top500.org/system/178321
[2] I. Zinno, et al., "Cloud computing for earth surface deformation analysis via spaceborne radar imaging: A case study," *IEEE Trans. Cloud Comput.*, vol. 4, no. 1, pp. 104–118, Jan. 2016.
[3] E. Roloff, F. Birck, M. Diener, A. Carissimi, and P. O. A. Navaux, "Evaluating high performance computing on the windows azure platform," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 803–810.
[4] J. Gibson, R. Rondeau, D. Eveleigh, and Q. Tan, "Benefits and challenges of three cloud computing service models," in *Proc. 4th Int. Conf. Computat. Aspects Social Netw*, 2012, pp. 198–205.
[5] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas, "Performance implications of multi-tier application deployments on infrastructure-as-a-service clouds: Towards performance modeling," *Future Generation Comput. Syst.*, vol. 29, no. 5, pp. 1254–1264, 2013.
[6] A. Iosup and D. Epema, "Grid computing workloads," *IEEE Internet Comput.*, vol. 15, no. 2, pp. 19–26, Mar. 2011.
[7] 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.
[8] J. Li, M. Qiu, Z. Ming, G. Quan, X. Qin, and Z. Gu, "Online optimization for scheduling preemptable tasks on IaaS cloud systems," *J. Parallel Distrib. Comput.*, vol. 72, no. 5, pp. 666–677, 2012.
[9] J. G. F. Coutinho, "HARNESS project: Managing heterogeneous computing resources for a cloud platform," in *Reconfigurable Computing: Architectures, Tools, and Applications*. Berlin, Germany: Springer, 2014, pp. 324–329.
[10] Q. He, S. Zhou, B. Kobler, D. Duffy, and T. McGlynn, "Case study for running HPC applications in public clouds," in *Proc. 19th ACM Int. Symp. High Performance Distrib. Comput.*, 2010, pp. 395–401.
[11] E. Jeannot, E. Saule, and D. Trystram, "Optimizing performance and reliability on heterogeneous parallel systems: Approximation algorithms and heuristics," *J. Parallel Distrib. Comput.*, vol. 72, no. 2, pp. 268–280, 2012.
[12] J. Koomey, "Growth in data center electricity use 2005 to 2010," *New York Times*, vol. 49, no. 3, 2011.
[13] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. Int. Conf. High Performance Comput. Netw. Storage Anal.*, Nov. 2011, pp. 1–12.
[14] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proc. Int. Conf. High Performance Comput. Netw. Storage Anal.*, Nov. 2012, pp. 1–11.
[15] K. Jansen and L. Porkolab, "Improved approximation schemes for scheduling unrelated parallel machines," *Math. Operations Res.*, vol. 26, no. 2, pp. 324–338, 2001. [Online]. Available: http://dx.Doi.org/10.1287/moor.26.2.324.10559
[16] O. Beaumont, A. Legrand, L. Marchal, and Y. Robert, "Steady-state scheduling on heterogeneous clusters," *Int. J. Found. Comput. Sci.*, vol. 16, no. 2, pp. 163–194, 2005.
[17] I. Al-Azzoni andD. G. Down, "Linear programming-based affinity scheduling of independent tasks on heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 12, pp. 1671–1682, Dec. 2008.
[18] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," *Proc. VLDB Endowment*, vol. 3, no. 1/2, pp. 460–471, Sep. 2010. [Online]. Available: http://dx.Doi.org/10.14778/1920841.1920902
[19] M. Rehman and M. Sakr, "Initial findings for provisioning variation in cloud computing," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, Nov. 2010, pp. 473–479.
[20] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, and I. Stoica, "Improving MapReduce performance in heterogeneous environments," in *Proc. 8th USENIX Conf. Operating Syst. Des. Implementation*, 2008, pp. 29–42.
[21] M. Rodriguez and R. Buyya, "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, Apr. 2014.

[22] S. Ahmed, A. J. King, and G. Parija, "A multi-stage stochastic integer programming approach for capacity expansion under uncertainty," Stochastic Programming E-Print Series, 2001. [Online]. Available: http://www.speps.org
[23] M. Riis and J. Lodahl, "A bicriteria stochastic programming model for capacity expansion in telecommunications," *Math. Methods Operations Res.*, vol. 56, no. 1, pp. 83–100, 2002.
[24] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*, 1st ed. Belmont, MA, USA: Athena Scientific, 1997.
[25] A. M. Al-Qawasmeh, A. A. Maciejewski, R. G. Roberts, and H. J. Siegel, "Characterizing task-machine affinity in heterogeneous computing environments," in *Proc. Int. Parallel Distrib. Process. Workshops Phd Forum Heterogeneity Comput. Workshop*, May 2011, pp. 34–44.
[26] D. D. Lee and H. S. Seung, "Algorithms for non-negative matrix factorization," in *Advances in Neural Information Processing Systems*, Cambridge, MA, USA: MIT Press, 2001, pp. 556–562.
[27] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*. Berlin, Germany: Springer, 2011.
[28] R. M. Van Slyke andR. Wets, "L-shaped linear programs with applications to optimal control and stochastic programming," *SIAM J. Appl. Math.*, vol. 17, no. 4, pp. 638–663, 1969.
[29] R. B. Franca, E. C. Jones, C. N. Richards, and J. P. Carlson, "Multi-objective stochastic supply chain modeling to evaluate tradeoffs between profit and quality," *Int. J. Prod. Econ.*, vol. 127, no. 2, pp. 292–299, 2010.
[30] J. Teghem, D. Dufrane, M. Thauvoye, and P. Kunsch, "STRANGE: An interactive method for multi-objective linear programming under uncertainty," *Eur. J. Oper. Res.*, vol. 26, pp. 65–82, 1986.
[31] V. Pareto, *Cours d'economie Politique*. Lausanne, Switzerland: F. Rouge, 1896.
[32] 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 Optimization*, vol. 13, no. 1, pp. 1–24, 1998.
[33] A. Löhne, *Vector Optimization with Infimum and Supremum*. Berlin, Heidelberg: Springer, 2011.
[34] G. Eichfelder, *Adaptive Scalarization Methods in Multiobjective Optimization*. Berlin, Germany: Springer, 2008.
[35] I. Y. Kim and O. De Weck, "Adaptive weighted sum method for multiobjective optimization: A new method for Pareto front generation," *Struct. Multidisciplinary Optimization*, vol. 31, no. 2, pp. 105–116, 2006.
[36] M. Ehrgott, *Multicriteria Optimization*. Berlin, Germany: Springer, 2006.
[37] V. Chew, "Confidence, prediction, and tolerance regions for the multivariate normal distribution," *J. Amer. Statistical Assoc.*, vol. 61, no. 315, pp. 605–617, 1966.
[38] Coin-OR SMI, Jan. 2015. [Online]. Available: https://projects.coin-or.org/Smi
[39] Coin-OR CLP, Jan. 2015. [Online]. Available: https://projects.coin-or.org/Clp
[40] Intel core i7 3770k power consumption, thermal, May, 2013. [Online]. Available: http://openbenchmarking.org/result/1204229-SU-CPUMONITO81
[41] R. Friese, et al., "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 Phd Forum Heterogeneity Comput. Workshop*, 2013, pp. 19–30.
[42] T. Huynh, C. Lassez, and J.-L. Lassez, "Practical issues on the projection of polyhedral sets," *Ann. Math. Artif. Intell.*, vol. 6, no. 4, pp. 295–315, 1992.



Kyle M. Tarplee received the BSEE and master's degrees from the University of California, San Diego, and the PhD degree in electrical engineering from Colorado State University. He is an assistant professor of engineering with Anderson University. Previously he has worked in the Department of Defense, Numerica Corporation for ten years developing multi-target tracking algorithms. He has been the principal investigator on several projects.He is a senior member of the IEEE. Further details at: https://www.engr.colostate.edu/∼ktarpleeand http://www.anderson.edu/science-engineering/faculty/engineering/tarplee.

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 with 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. A complete vita is available at: https://www.engr.colostate.edu/∼aam



Howard Jay Siegel received the BS degree from MIT, and the PhD degree from Princeton. He was appointed the Abell Endowed chair distinguished professor of electrical and computer engineering with Colorado State University, in 2001, where he is also a professor of computer science. From 1976 to 2001, he was a professor with Purdue University, West Lafayette. He is a fellow of the IEEE and the ACM. A complete vita is available at: https://www.engr.colostate.edu/∼hj