Dynamic Pricing For Efficient Workload Colocation

VATCHE ISHAKIAN RAYMOND SWEHA AZER BESTAVROS JONATHAN APPAVOO
visahak@bu.edu remos@bu.edu best@bu.edu jappavoo@bu.edu

Computer Science Department, Boston University
Boston, MA 02215, USA

Abstract—**Pricing models for virtualized (cloud) resources are meant to reflect the operational costs and profit margins for providers to deliver specific resources or services to customers subject to an underlying Service Level Agreements (SLAs). While the operational costs incurred by cloud providers are dynamic – they vary over time, depending on factors such as energy cost, cooling strategies, and overall utilization – the pricing models extended to customers are typically fixed – they are static over time and independent of aggregate demand. This disconnect between the cost incurred by a provider and the price paid by a customer results in an inefficient marketplace. In particular, it does not provide incentives for customers to express workload scheduling flexibilities that may benefit them as well as cloud providers. In this paper, we propose a new dynamic pricing model that aims to address this marketplace inefficiency by giving customers the opportunity and incentive to take advantage of any tolerances they may have regarding the scheduling of their workloads. We present the architecture and algorithmic blueprints of a framework for workload colocation, which provides customers with the ability to formally express workload scheduling flexibilities using Directed Acyclic Graphs (DAGs), optimizes the use of cloud resources to collocate clients' workloads, and utilizes Shapley valuation to rationally – and thus fairly in a game-theoretic sense – attribute costs to customer workloads. In a thorough experimental evaluation we show the practical utility of our dynamic pricing mechanism and the efficacy of the resulting marketplace in terms of cost savings.**

I. INTRODUCTION

Motivation: Cloud computing in general and Infrastructure as a Service (IaaS) in particular have emerged as compelling paradigms for the deployment of distributed applications and services on the Internet due in large to the maturity and wide adoption of virtualization. By relying on virtualized resources, customers are able to easily deploy, scale up or down their applications seamlessly across computing resources offered by one or more infrastructure providers [1]. Cloud Providers incur a significant capital investment as part of creating and providing these services. These investments can be divided into two major categories: capital expenditures and operational expenditures. Capital expenditures are the initial fixed costs for setting up the data center infrastructure, including servers, network devices, transformers, cooling systems, *etc.* Operational expenditures, on the other hand, are the recurring costs throughout the lifetime of the data center, including power, maintenance, and labor costs. A data center's return on investment (ROI) relies heavily on decreasing its overall cost through efficient cooling and energy conservation [2], [3], while increasing its overall utilization as customers' adoption of cloud services increases.

Minimizing the overall cost involves a non-trivial optimization that depends on many factors, including time and location dependent factors. For example, in some cities, the cost of energy is variable depending on time of day [4], [5], while the cost of cooling might be higher during summer than winter, or during peak utilization times. The physical location of allocated resources in the data center can also be a crucial factor in cost reduction. An efficient allocation can lead to powering down different server and network resources [2], or in decreased cost of cooling [6]. These approaches are but examples of what cloud providers must consider in order to decrease their overall operational costs.

Despite these complexities, the pricing models extended to cloud customers are typically fixed – they are static over time and independent of aggregate demand. For example, the pricing model of IaaS providers such as Amazon and Rackspace [7], [8] for leasing resources is in the form of *fixed-price SLAs*, which do not vary with resource availability, seasonal peak demand, and fluctuating energy costs. From the customers' perspective, fixed pricing has its advantages due to its simplicity and the fact that it provides a sense of predictability. That said, fixed pricing has many disadvantages for customers and providers alike due to the fact that it does not allow *both* of them to capitalize on customer-side elasticity.

Under a fixed pricing model, customers do not have any incentive to expose (and do not have any means to capitalize on) the elasticity of their workloads. By workload elasticity, we refer to scheduling flexibilities that customers may be able to tolerate. This customer-side *demand elasticity* could be seen as an asset that may benefit *both* customers and providers, in the same way that provider-side *supply elasticity* proved to be instrumental in the development of an entire industry. From the provider's perspective, demand elasticity could be seen as an additional lever in the aforementioned optimization of operational costs, whereas from the customer's perspective, demand elasticity could be seen as a feature of their workloads that should translate to cost savings. Fixed pricing models do not enable demand elasticity to play a role in the marketplace, effectively resulting in an inefficient marketplace.

Leveraging customer-side demand elasticity requires the development of dynamic (as opposed to fixed) pricing mech-

This research was supported in part by NSF awards #0720604, #0735974, #0820138, #0952145, and #1012798.

anisms and associated flexible SLA models that provide customers with proper incentives and assurances. In particular, the pricing mechanism must provably reward (and certainly never mistreat) customers for expressing the scheduling flexibilities in their workloads.

Scope and Contribution: In this paper, we propose a framework (a flexible SLA model and an associated dynamic pricing mechanism) that achieves the above-stated goals by giving customers both the means and incentive to express any tolerances they may have regarding the scheduling of their workloads. Our framework improves IaaS pricing transparency, enhances credibility of IaaS providers, and increases customer confidence in the IaaS marketplace. It does so by ensuring that resources are allocated to customers who value them the most. Our pricing model consists of two major components: a fixed component, reflecting the fixed costs associated with the acquisition and maintenance of the infrastructure, and a time-variant amortized component, reflecting variable operational costs. Our framework incorporates a resource specification language that provides customers not only the ability to state the cloud resources they require, but also to express their scheduling flexibilities. Realizing that more efficient resource utilization could be achieved by appropriately *colocating* applications from multiple IaaS customers on the same set of resources, we develop techniques that optimize the colocation of customer workloads to maximize the efficient use of IaaS resources. Our techniques are based on a pricing mechanism that not only attributes accrued costs rationally – and thus fairly in a game-theoretic sense – across customers, but also provides incentives for customers to declare their flexibilities by guaranteeing that they will not be mistreated as a consequence. Our framework could be incorporated into an IaaS offering by providers; it could be implemented as a value-added proposition by IaaS resellers; or it could be directly leveraged in a peer-to-peer fashion by IaaS customers. We note that our framework is generic enough to be applicable to any market-based resource allocation environments, with IaaS being a specific instantiation. Results from extensive simulations using synthetically generated workloads, selected from a set of representative real workload models, highlight the practical utility of our dynamic pricing mechanism, the efficacy of our algorithm in colocating workloads, and the rationally fair distribution of costs among customers.

Paper Overview: The remainder of this paper is organized as follows. In Section II, we present a resource cost model. In Section III, we present the basic concepts underlying our colocation framework, along with the set of components required for efficient optimization of resources and fair allocation of cost among customers. In Section IV, we present experimental results that demonstrate the promise from using our colocation framework to manage the colocation of different types of workloads. In Section V, we review relevant related work. We conclude in Section VI with closing remarks and future research directions.

II. IaaS Resource Cost Model

As we alluded before, fixed resource pricing does not reflect the time-variant expenses incurred by providers and fails to capitalize on the scheduling flexibilities of customers. Expenses incurred by providers are affected by different criteria such as datacenter utilization, efficient cooling strategies, ambient temperature, total energy consumption, and energy costs. Indeed, studies indicate that the amortized cost of energy and physical resources account for 30% and 45% of the cost of datacenters, respectively [1], [9]. In addition, it is becoming a norm for datacenters to be charged a variable hourly rate for electricity [4], or for peak usage [9]. Accordingly, in this paper, we consider two factors to be the primary determinants of the costs incurred by providers: (1) the variable cost of electricity as a function of the time of the day, and (2) the level of utilization of resources, and hence the power consumption, at each point in time.

In order to pursue this notion further, we need an accurate model of resource energy consumption. Recent work on energy [3], [10], [11] suggest that a physical machine's power consumption increases linearly with the system load, with a base idle power draw of 60%. Under this simple model one can already observe a generic notion of fixed and variable costs. In addition, Ranganathan et al. [12] suggest a linear relationship between watts consumed for powering and watts consumed for cooling. Using this knowledge, it is reasonable to assume that the total expense of operating a resource j during time t is:

$$P_j + f(t, U_j(t))$$

where P_j reflects the fixed cost of the resource j. The function $f(t, U_j(t))$ is the energy cost consumed by resource j at time t under utilization $U_j(t)$. we define $f(t, U_j(t))$ as follows:

$$f(t, U_j(t)) = \alpha(t)(v_0 + (1 - v_0)U_j(t) * R_j)$$

where $\alpha(t)$ is a coefficient reflecting the energy cost at time t, and v_0 is the energy fraction consumed by the resource when idle,[1] and R_j is the fixed capacity of resource j which is generic enough to reflect a single host, a single rack, or an entire datacenter.[2] Note that $f(t, U_j(t))$ has also a fixed part reflecting the cost of operating the resource while idle.

III. Colocation: Framework

We assume a general setting consisting of any number of possibly heterogeneous resources, (e.g. physical machines). Each resource is characterized by a number of dimensions (e.g., CPU, network, memory, and disk space) which constitute dimensions of the resource capacity vector. To achieve an efficient scheduling of resources, we observe that the major underlying property to exploit is how different applications are assigned to a resource in the datacenter. This assignment, has implications on the aggregate expense of the provider, and consequently the overall customer cost. Our approach is

[1] Throughout this paper, we take v_0 to be 60% [3], [10], [11].

[2] Although we take energy as an example of time variant cost, our model could apply to any other time variant cost.

to design a framework which has three major components: (1) a workload specification language that enables customers to specify their workloads as a Directed Acyclic Graph (DAG), where each node represents a multi-dimensional utilization vector,[3] (2) a colocation strategy based on optimization theory that generates an efficient colocation of workloads, and (3) a cost distribution component which allocates the total cost of resources among customers in a rationally fair manner.

A. Colocation: Workload Specification

We propose an expressive model for customer workloads, which allows them to declare their quantitative resource requirements as well as any associated temporal flexibilities. A workload is represented as a DAG. A node in the graph represents a single task to be mapped to a resource, and consumes some of the resource dimensions.[4] A task has two attributes: The total number d of time slots during which the task must remain on the same resource, and a quantitative resource request matrix $V^{m \times d}$, consisting of d vectors, where m represents the different dimensions required during each period. The directed edges in the graph represent the temporal dependencies between tasks. An edge between node k and o dictates that task k needs to finish execution before task o starts execution. The weight on an edge $w \geq 0$ designates the maximum delay a customer can tolerate between releasing a resource by task k and acquiring a resource for the execution of task o. In addition, a customer i specifies an execution window (T_i^s, T_i^e), where T_i^s is the workload earliest start time, and T_i^e is a deadline for the completion of the workload. This formally declared temporal flexibility by a customer will be exploited by our framework to achieve better colocation.

This model is expressive enough for various types of applications. Figure 1 (a) shows a sample specification for a batch workload. Such a workload is representative of bulk data transfer or backup applications. The workload consists of six tasks with different utilization levels and durations. The tasks are not temporally dependent, thus there are no edges between them, implying that they may be satisfied in any order within the exeuction window. Specifying a web server, which requires the workload to execute on the same resource would result in representing the workload as one node with a duration equal to 24 and volume $V_{m \times 24}$ that varies accordingly. Figure 1 (b) illustrates a pipelined workload with 24 nodes, where tasks need to execute in sequence throughout an entire day with different utilizations, and the delay between the execution of two consecutive tasks is zero. These two examples illustrate scenarios in which the customer has no scheduling flexibilities. Figure 1 (c) illustrates a typical MapReduce workload, where a scheduling task needs to execute, followed by a set of uncorrelated *map* tasks, and finishing with a *reduce* task. Figure 1 (d) is a constrained version of the MapReduce workload, where some communicating tasks need to run concurrently. We introduce a *marker* node, (in red), that has a duration of zero and a utilization of zero; it forces a number of tasks to run concurrently once the marker node is scheduled.

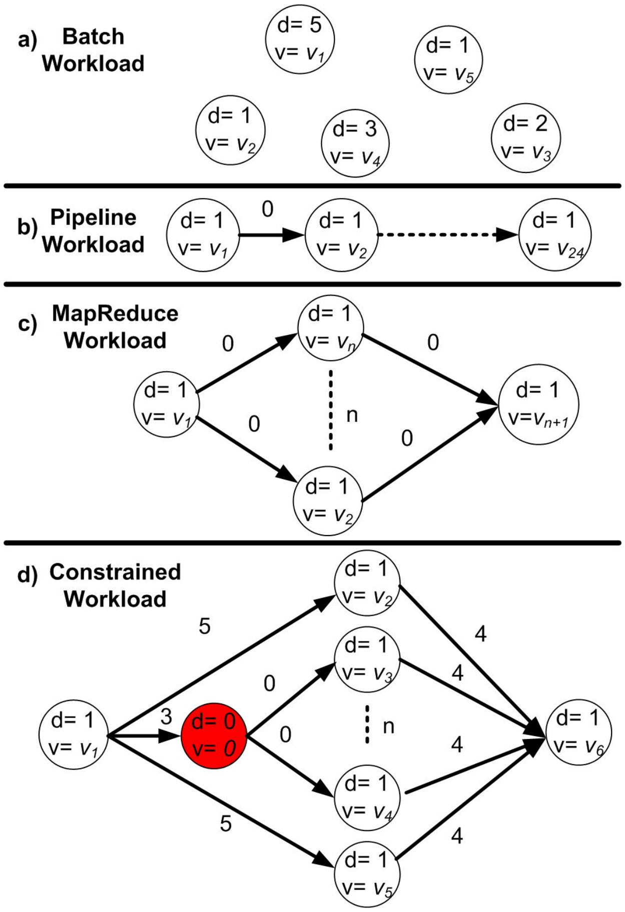

Fig. 1. An example illustrating different workload models.

B. Colocation: Optimization Problem

In the previous section, we presented our workload specification language, which allows IaaS customers to describe their workloads. In this section, we formulate the colocation problem and present a linear programming optimization solution. The objective of the system is to fulfill the requests of all customers while incurring the minimal total cost. The aggregate load on the system can be represented by the graph $G =< V, E >$, representing the union of the DAGs $G_i =< V_i, E_i >$ representing the workloads of all customers $i \in U$ – namely, $V = \bigcup_{\forall i} V_i$ and $E = \bigcup_{\forall i} E_i$.

We define $Y(t, j)$ to be a binary decision variable that equals to one when resource j is in use at time t. We also define $X(j, t, k, l)$ to be a binary decision variable such that

$$X(j,t,k,l) = \begin{cases} 1 & \text{If resource } j \text{ at time } t \text{ is assigned to} \\ & \text{node } k\text{'s duration } l. \\ 0 & \text{Otherwise} \end{cases}$$

[3] We note that our workload specification language allows customers to specify additional dimensions associated with each node (e.g., location, operating system, etc.). Without loss of generality, in this paper, we only consider dimensions related to consumable physical resources.

[4] Conveniently, and without loss of generality, one might view each task as underlying a virtual machine.

We formulate our colocation optimization problem as follows, (verbal description to follow):

$$\min \sum_{\forall t,j} (Y(t,j) \times P_j + Y(t,j) \times \alpha(t) \times v_0 + \alpha(t) \times (1-v_0)U_j(t) \times R_j)$$

Subject to:

$$\sum_{\forall l} X(j,t,k,l) \leq Y(j,t) \quad \forall t,j,k \tag{1}$$

$$\sum_{\forall k, 1 \leq l \leq d_k} X(j,t,k,l) \times u(k,l) \leq R_j \quad \forall j,t \tag{2}$$

$$\sum_{\forall j,t} X(j,t,k,l) = 1 \quad \forall k \in V, 1 \leq l \leq d_k \tag{3}$$

$$X(j,t,k,l) = X(j,t+1,k,l+1) \quad \forall j,t,k \in V, 1 \leq l < d_k \tag{4}$$

$$X(j,t,k,l) = 0 \quad \forall j,k \in V_i, t < T_i^s, 1 \leq l \leq d_k \tag{5}$$

$$X(j,t,k,l) = 0 \quad \forall j,k \in V_i, t > T_i^e, 1 \leq l \leq d_k \tag{6}$$

$$\sum_{j,t<t'} X(j,t,k,d_k) \geq \sum_{j'} X(j',t',o,1) \quad \forall t', (k,o) \in E \tag{7}$$

$$\sum_{j} X(j,t',k,d_k) \leq \sum_{j',t'<t \leq t'+W_e+1} X(j',t,o,1) \quad \forall t', (k,o) \in E \tag{8}$$

where P_j and R_j are the cost and capacity of a specific physical resource j, $U_j(t)$ is the total utilization of resource j at time t, v_0 is the energy consumed by resource j while idle, $\alpha(t)$ is the cost of energy at time t, and $u(k,l)$ is the utilization request of a nodes k's duration l. This formulation is a general enough to model different types of resources. Intuitively, the optimization problem aims to minimize the cost of resources across time. The objective function is the sum of three parts, reflecting the cost of leasing the resource: $Y(t,j) \times P_j$ reflects the fixed cost of leasing the resource, $Y(t,j) \times \alpha(t) \times v_0$ is the initial cost of energy to run the resource at an idle state, and $\alpha(t) \times (1-v_0)U_j(t) \times R_j$ stands for the additional as a consequence for utilizing the resource.

Equation (1) ensures that a resource j is utilized at time t, – $Y(j,t)$ is set to one – if that resource is used to service the requests of any customer. Equation (2) ensures that the utilization of a single resource does not exceed a fixed capacity R_j. Equation (3) guarantees that all periods of each task are fulfilled exactly once. Equation (4) ensures that a task's periods are allocated consecutively on the same resource. Equation (5) and (6) ensure that the time of execution of customer i's tasks are between the start time T_i^s and end time T_i^e specified by the customer. Finally, Equation (7) and (8), guarantee that the allocation of resources respects the client's edge constraints. In particular, Equation (7) constrains the allocation of a request o to follow the resources allocated to request k, while Equation (8) guarantees that such an allocation happens within the specified client's delay W_e on edge (k,o).

C. Colocation: Polynomial-time Solution

The optimization problem defined in the previous section is a variant of mixed-integer programming, which is known to be NP-hard in general. Therefore, in this section, we propose a polynomial-time greedy algorithm that results in solutions to our colocation problem, which we show to be effective in our experiments. The algorithm starts from an initial valid solution and iterates over several greedy *moves* until it converges.

The initial solution is created by randomly assigning workloads to resources, such that each workload's specific constrains are satisfied. Naturally, the initial solution's total cost is far more expensive than an optimal solution.

At each greedy move, the algorithm chooses a workload which has the highest current-to-optimal cost ratio r among all workloads. Calculating an optimal cost of a workload is not trivial, however, we can calculate the *Utopian* cost, a lower bound on the optimal workload cost efficiently, where the *utopian* cost of a workload reflects only the cost of energy and resources that the workload actually uses. The utopian cost is calculated under the assumption that there is a perfect packing of the workload, with the energy cost being the minimum throughout the customer's specified workload start and end times. The algorithm relocates the workload such that r is minimized. If such a relocation results in a total cost-reducing solution, then the move is accepted, the solution is updated, and the process is repeated. Otherwise, the algorithm chooses the workload with the second highest ratio r and iterates. The algorithm stops when the iteration step fails to find a move for any of the workloads.

D. Fair Pricing Mechanism

The colocation framework is designed to minimize the total aggregate cost of using resources. However, we need a pricing mechanism to apportion (distribute) this total cost across all customers. Such a pricing mechanism needs to ensure that the interests of customers, particularly fairness in terms of costs that customers accrue for the resources they acquire, and provides guarantees of no mistreatment of a customer's elasticity.

There are many ways to apportion the total cost across customers. For instance, one option would be to divide the cost equally among customers. Clearly, this mechanism will not be fair as it does not discriminate between customers with large jobs and customers with small jobs. Another option would be to charge each customer based on the proportional cost of each resource they utilize. As we will show next, such an option is also not fair.

Consider an example of two customers A and B each with a single task workload with 50% resource utilization.

Customer A is constrained to run during the highest energy cost period. Customer B has no such constraint. Let c_l be the cost of running during low energy period, and c_h be the cost of running during high energy period. An optimized solution would colocate customer A and B to run during the highest energy cost period with a total cost of c_h. Assuming the cost of $c_h > 2 \times c_l$. A proportional share pricing mechanism would divide the total cost across both customers, thus forcing customer B to pay more than what he/she would have paid had he/she run by herself at the lowest energy cost period.

A "rationally fair" pricing mechanism allocates the total cost over the customers in accordance with each customer's marginal contribution to the aggregate cost of using the resources. Such mechanism should take into consideration not only the actual customer workload demands, but also the effects of the workload constraints.

To quantify per-customer contribution, we resort to notions from economic game theory. In particular, we adopt the concept of Shapley value [13], which is a well defined concept from coalitional game theory that allows for fair cost sharing characterization among involved players (customers).

Given a set of n customers U, we divide the total cost of the system $C(U)$ by ordering the customers, say $u_1, u_2, \cdots, u_n$, and charging each customer his/her marginal contribution to the total system cost. Thus, u_1 will be charged $C(u_1)$, u_2 will be charged $C(u_1, u_2) - C(u_1)$, etc. Since the ordering of customers affects the amount they will be charged, a fair distribution should take the average marginal cost of each customer over all possible ordering permutations. Then the marginal cost of $\phi(C)$ of each customer u is defined as follows:

$$\phi_u(C) = \frac{1}{N!} \sum_{\pi \in S_N} \left(C(S(\pi, u)) - C(S(\pi, u) \setminus u) \right) \qquad (9)$$

where $S(\pi, i)$ is the set of players arrived in the system not later than u, and π is a permutation of arrival order of those customers. Thus player u is responsible for its marginal contribution $v(S(\pi, u)) - v(S(\pi, u) \setminus u)$ averaged across all $N!$ arrival orders of π.

Looking back at the previous example of two customers A and B, there are two possible ordering: B, A and A, B. For the first, the cost of $B = c_l$ and the cost of $A = c_h - c_l$. For the second, the cost of $A = c_h$, and the cost of $B = 0$. After averaging both costs, we end up with a rationally fair individual cost distribution: $B = \frac{c_l}{2}$ and $A = c_h - \frac{c_l}{2}$.

By adopting Shapley value as a rationally fair mechanism for allocating costs, customers have the incentive to declare the flexibility (if any), because the pricing mechanism guarantees that a customer's cost will not increase because of flexibility. We formalize this notion in the following theorem.

Theorem 1. *The fair pricing mechanism under Shapley value guarantees no mistreatment as a result of customer flexibility, i.e.,* $\phi_i(C) - \phi_i(C)_F \geq 0$, *where* $\phi_i(C)$ *is the cost of customer* i *and* $\phi_i(C)_F$ *is the cost of flexible customer* i *under Shapley value.*

Proof: The proof is by contradiction. Assuming that the opposite is true, i.e., $\phi_i(C) - \phi_i(C)_F < 0$, implies that there exists at least one permutation where $C(S(\pi, i)) - C(S(\pi, i) \setminus i) - C(S(\pi, i))_F - C(S(\pi, i) \setminus i)_F < 0$. Since the configuration of other players did not change, then $C(S(\pi, i) \setminus i)_F = C(S(\pi, i) \setminus i)$. Thus, $C(S(\pi, i)) - C(S(\pi, i))_F < 0$. This implies that the optimization solution $OPT(i)$ resulting in $C(S(\pi, i))$ is better than the optimization solution $OPT(i)_F$ resulting in $C(S(\pi, i))_F$. But if $OPT(i)$ is better than $OPT(i)_F$ then the optimization should have found it, since the flexibility of the customer contains the constrained version as well – a contradiction. ■

While computing the exact cost for each customer using Equation (9) is straightforward for small number of customers, finding the exact cost becomes infeasible as the number of customers increases. Thus, we resort to computing an estimate of the Shapley value.[5] We utilize Castro's [15] polynomial time estimation of Shapley value, which not only achieves a good estimation of the original Shapley value, but also provides bounds on the estimation error.

Let $Sh = (\phi_1(C), \phi_2(C), \cdots \phi_n(C))$ be a vector representing the Shapley value of all customers based on all possible $N!$ permutations; let the estimated Shapley value based on m sample permutations be $S\hat{h} = (\hat{\phi}_1(C), \hat{\phi}_2(C), \cdots, \hat{\phi}_n(C))$. $S\hat{h}$. Using the central limit theorem, Castro's technique calculates the number of permutations m needed such that $P(|\phi_i(C) - \hat{\phi}_i(C)| \leq \epsilon) \geq 1 - \alpha$, where ϵ is the error bound, and α is the confidence factor. Calculating the number of samples m required to achieve the bound $P(|\phi_i(C) - \hat{\phi}_i(C)| \leq \epsilon) \geq 1 - \alpha$ requires knowing the standard deviation σ, which is an unknown value. In our setting, to calculate σ, we first (conservatively) take the standard deviation σ_i of each customer to be $\omega_h - \omega_l$: ω_l reflects the cost incurred by the customer under the assumption that there is an optimal packing of the workload with minimum cost of energy, and ω_h reflects the cost incurred by the customer under the assumption that the workload is the only workload in the system with a maximal cost of energy. A worst case value on σ could be calculated by taking $\sigma = \max(\sigma_1, \sigma_2, \cdots, \sigma_i)$ for all customers i.

Let $\hat{\phi}_i(C)_F$ be the flexibility of a customer using a Shapley value sampling technique. The mistreatment guarantee by the system no longer holds. However, as we show in Theorem 2, we can bound the mistreatment of the customer based on the original Shapley value.

Theorem 2. *The fair pricing mechanism under an estimated Shapley value bounds the mistreatment of a customer as a result of his/her flexibility from the original Shapley value to be* $\leq \epsilon$ *i.e.,* $P(\hat{\phi}_i(C)_F - \phi_i(C) \leq \epsilon) \geq 1 - \frac{\alpha}{2}$, *where* $\hat{\phi}_i(C)_F$ *is the sampled cost of flexible customer* i, $\phi_i(C)$ *is the cost of customer* i *under Shapley value,* ϵ *is the error bound, and* α *is the confidence factor.*

Proof: Using a Shapley value sampling technique, we

[5] Estimating Shapley value has proven to be effective in calculating the contribution of customers to the effective network peak demand [14].

have $P(|\hat{\phi}_i(C)_F - \phi_i(C)_F| \leq \epsilon) \geq 1-\alpha$, thus, $P(\hat{\phi}_i(C)_F - \phi_i(C)_F \leq \epsilon) \geq 1 - \frac{\alpha}{2}$. But we know from Theorem 1 that $\phi_i(C)_F \leq \phi_i(C)$, thus, $P(\hat{\phi}_i(C)_F - \phi_i(C) \leq \epsilon) \geq 1 - \frac{\alpha}{2}$. ■

Since comparison against Shapley valuation is impractical because of it computational inefficiency, which might not provide confidence for customer to be flexible, A further motivation is provided by bounding the flexible Shapley value with the estimated Shapley value.

Theorem 3. *The fair pricing mechanism under estimated Shapley value bounds the mistreatment of a customer as a result of his/her flexibility to be* $\leq \epsilon_1 + \epsilon_2$, *i.e.* $\hat{\phi}_i(C)_F \leq \hat{\phi}_i(C) + \epsilon_1 + \epsilon_2$ *with probability* $(1-\frac{\alpha}{2})^2$, *where* $\hat{\phi}_i(C)_F$ *is the sampled cost of flexible customer i,* $\hat{\phi}_i(C)$ *is the sampled cost of customer i,* ϵ_1 *and* ϵ_2 *are the sample error bounds, and* α *is the confidence factor.*

Proof: Using the Shapley value sampling technique, we have the following results: $|\phi_i(C) - \hat{\phi}_i(C)| \leq \epsilon_1$ and $|\hat{\phi}_i(C)_F - \phi_i(C)_F| \leq \epsilon_2$ with probability $(1-\alpha)$. Thus, $P((\phi_i(C) - \hat{\phi}_i(C)) \leq \epsilon_1) \geq 1 - \frac{\alpha}{2}$ and $P(\hat{\phi}_i(C)_F - \phi_i(C)_F \leq \epsilon_2) \geq 1 - \frac{\alpha}{2}$. Since the sampling process is independent, The probability of $(\phi_i(C) - \hat{\phi}_i(C)) \leq \epsilon_1$ and $\hat{\phi}_i(C)_F - \phi_i(C) \leq \epsilon_2$ is equal to $(1-\frac{\alpha}{2})^2$.

In addition, from Theorem 1, we have $\phi_i(C)_F \leq \phi_i(C)$. Therefore we have $\hat{\phi}_i(C)_F \leq \epsilon_2 + \phi_i(C)_F \leq \epsilon_2 + \hat{\phi}_i(C) + \epsilon_1$ with probability $(1-\frac{\alpha}{2})^2$. ■

Finally, an added property of Shapley and sampled Shapley value is budget balance *i.e.* the total cost of customers is always equal to the total cost of the resources used. This property works as incentive for providers or resellers, since it guarantees that they are going to get a revenue which covers the resources they lease.

IV. Experimental Evaluation

In this section, we present results from extensive experimental evaluations of our colocation framework. Our main purpose is to establish the feasibility of our proposed framework as an underlying mechanism to make effective use of a provider's IaaS and still achieve a fair allocation of costs among customers, by (1) establishing the efficacy of our greedy heuristic by comparing it to optimally packed workloads, (2) evaluating the cost incurred by the customer to use such a system to allocate a workload compared to the utopian cost, and (3) measure the benefit of a customer from flexibility.

Workload models: To evaluate our experiments, we synthetically generate workloads based on the workload models (shown in Figure 1), such as batch, and mapReduce workloads. We generate two pipeline workload versions: Webserver which has a single node with an execution length equal to the length of the epoch, and a *chain* workload which has a variable number of sequential tasks.[6] In addition, we enrich our set of workloads with two additional High Performance Computing workloads (c.f. Figure 2) for Protein annotation workflow (*PAW*), and Cognitive Neuroscience (*fMRI*) [16]. We believe that this set of workload models is representative for many cloud based applications. We assume homogeneous resources with a fixed cost of 10 cents per hour, a resource capacity equal to one, and an epoch consisting of twenty four hours. To calculate the number of samples m required to estimate a Shapley costs, we take $\epsilon = 0.1$, and $\alpha = 0.05$. Based on available server power consumption measurements provided by Koomey [17], specifically for mid-range server, we assume that a physical resource's power consumption is 500 watts per hour.

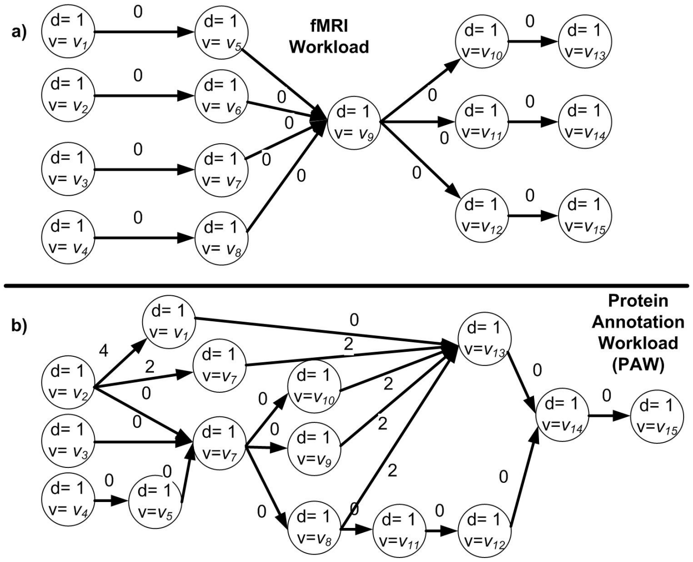

Fig. 2. High Performance Computing Workloads

Energy Cost: To model the energy cost for our colocation framework, we use real energy costs from the Ameren website [5]. Ameren publishes energy costs daily on an hourly basis. We get energy cost for a one month period (from 08/01/11 to 08/31/11) and average them per hour. Figure 3 shows the average price of energy for this period over a 24-hour period. The cost of energy reflects a diurnal pattern – higher during the day and cheaper at night.

Efficacy of our greedy heuristic: In this experiment, we evaluate the performance of our greedy heuristics compared to an optimal colocation of tasks. Since knowing an optimal colocation is difficult (bin packing is NP-hard), we resort to generating workloads for which we know (by construction) that an optimal colocation exists.

We do so by lining up a set of resources for the duration of an epoch, and repeatedly creating fragments to fill a resource to it full capacity. We generate fragments based on a uniform distribution between zero and one, thus the average number of fragments per resources is two.[7] Similar results for resource fragmentation were observed given other distributions but were omitted due to lack of space. We proceed in a round-robin

[6] We vary the length of the chain workload in our experiments.

[7] If the generated fragment is greater than the leftover resource capacity, then we assign the fragment the remaining resource capacity.

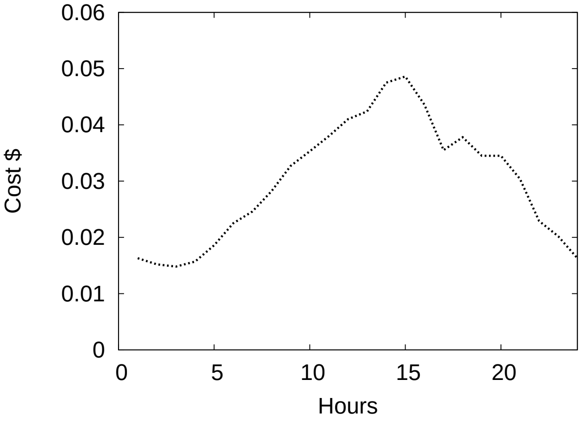

Fig. 3. Energy Cost (KW/H)

fashion over the set of workload models in our disposal (except the batch), and greedily embed each workload over the resource epoch lineup. Once no more workloads can be embedded, we assign the remaining unembedded fragments as part of a batch workload. By construction, we know that a "perfect" colocation exists (with every resource being fully utilized for the entire epoch).

We set the start time and end time of all workloads to be the beginning and end of the epoch, respectively. Next, we place the resulting workloads to be the input to our greedy heuristic. Our purpose from this experiment is to evaluate how far our heuristic is from an optimal colocation. Therefore, we assume that the cost of electricity is fixed (i.e., independent of time).

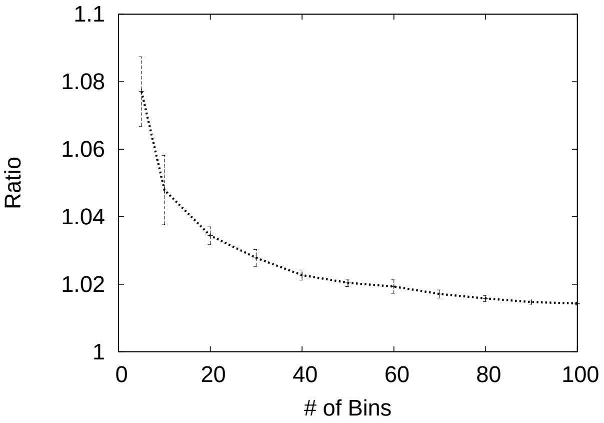

Fig. 4. Packing Ratio (Heuristic/Optimal)

Figure 4 shows the ratio of packing achieved using our algorithm relative to an optimal colocation. The x-axis shows the number of bins used, and the y axis shows the ratio of workload colocation achieved using our heuristic over that of an optimal colocation. The results are reported with 95% confidence. The figure shows that our algorithm's performance is highly comparable to the optimal. Furthermore, as we increase the number of bins, the ratio decreases.

Fair pricing scheme vs. utopian customer cost: Unlike the pervious experiment, which aimed to show the efficacy of our heuristic by comparing its performance to an optimally-colocated set of workloads, the purpose of this experiment is to highlight the fairness of our game-theoretic inspired pricing scheme in comparison to the utopian cost of the customer. As we alluded before, the utopian cost is the (possibly unrealistic) minimal possible cost – reflecting only the cost of the energy and resources the customer actually uses.

To generate workloads, we start by selecting a workload model based on a uniform distribution where each workload type: HPC (fMRI, PAW), WebServer, MapReduce (MR), Chain, and batch get equal percentages (20%) of the total workload population. Once a workload is selected, we generate a start time randomly for the workload to execute, and set the end time of the workload to be the start time plus the length of execution of the workload. This is an easy step since all of the workload except chain have fixed structures. For chain workloads, we generate the number of consecutive resource requests based on an exponential distribution with a mean of six. If the end time is greater than the duration of the epoch, then we exclude that workload, and proceed to generate a new one, otherwise we accept the generated workload as part of the overall workload population.

To model the utilization of the webserver workload, we use a standard method of generating the workloads based on an exponential distribution whose mean is modulated by a Sine function. This is done to model the diurnal pattern of higher web server load during the day, and lower web server load at night. For the remaining workload models, we generate the utilization of requests based on a uniform distribution between 0.2 and 1.

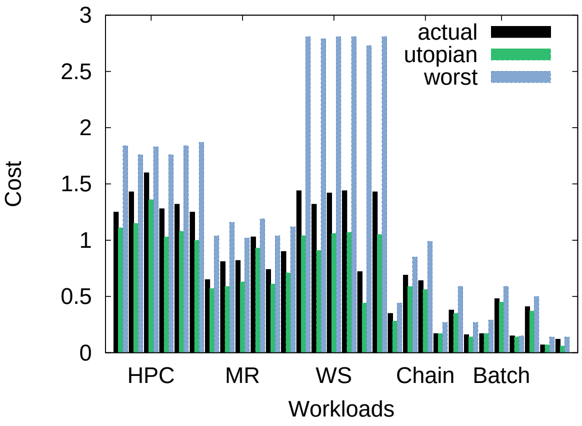

Fig. 5. Per workload cost comparison

Figure 5 shows the distribution of costs based on sampled Shapley value for 30 workloads, where all workload types have equal percentage of workload population (20%). We also show the utopian cost, as well as the cost incurred by the customer had she opted to execute her workload by herself (i.e. no colocation), which we denote as *Worst* cost. As shown, approximate Shapley value is close to the utopian cost. An interesting observation is the ratio between the utopian and approximate cost is highest for webserver workloads, while batch workloads are very close to the utopian. In fact, we also observe that batch workloads can even pay *less* than their utopian. This is due to the fact that batch workloads are the least restrictive workloads in terms of modeling (no edges between tasks), and have complete time flexibility, while webservers have the least flexibility.

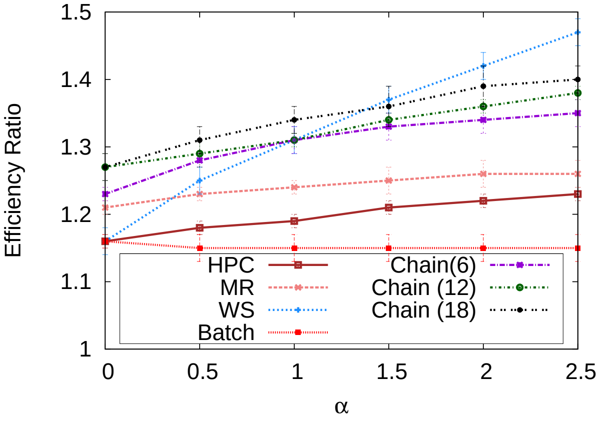

Fig. 6. Effect of energy fluctuation on workload cost

To further investigate this phenomena, we proceed to measure the sensitivity of workload costs to fluctuation in energy costs. To model variability in energy cost, we use the distribution of energy highlighted in Figure 3, and modulate it by multiplying it with α, where α varies between 0 and 2.5. For each workload model, we generate 25 workloads and calculate the cost of colocation using the modulated energy cost. We generate two additional variations of chain workloads with length based on exponential distribution with mean 12 and 18 respectively. We define the efficiency ration as the ration between the actual customer cost over the utopian cost. Figure 6 highlights our results. The x-axis plots the changing values of α. For $\alpha = 1$, the cost of energy reflects the actual cost shown in Figure 3. As highlighted, inflexible workloads, such as the webserver suffer most as a result of increase in energy cost with overall increase of more than 20 percent, while batch workloads do not show any increase.

Given the fluidity (maximal elasticity) of batch workloads, we investigate their effect when colocated with other workload models. We performed experiments using the same settings as the previous experiment: set the value of $\alpha = 1$, and for each workload type, we mix it with different percentages of batch workloads. Figure 7 shows the measured efficiency ratio for different percentages of batch workload mix. We observe that pipeline based workloads like chain and webserver are a better fit for batch workloads than HPC or MR workloads. One reason which is based on observing the actual colocation outcome is due to the existence of parallel branches in MR and HPC models, which provides an additional opportunity for colocation.

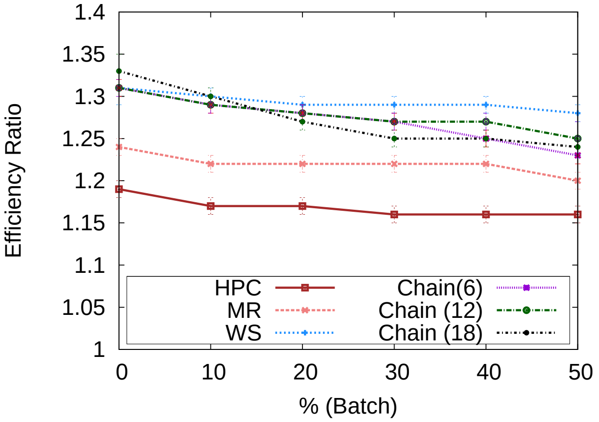

Fig. 7. Workloads with batch mix

Benefit from flexibility: To measure the effect of flexibility on the overall reduction in cost, we performed experiments using the same setting as before, while allowing the extension of start time and end time of workloads by σ, for different values of σ (hours). Figure 8 shows the effect of customer flexibility on workloads.[8] As expected, the more flexible a workload is, the better the efficiency ratio.

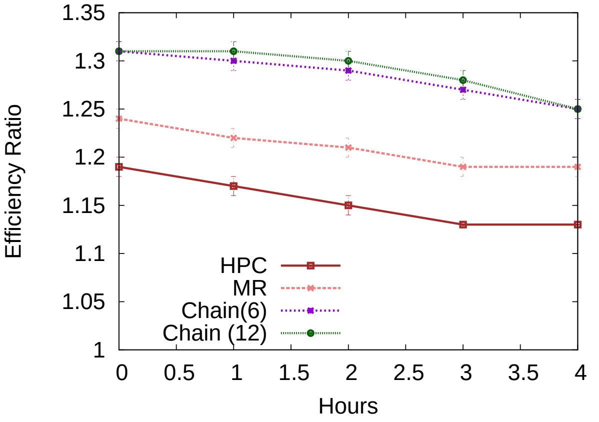

Fig. 8. Effect of Flexibility

V. Related Work

Economic models for resource management: Many resource management techniques have been developed for large-scale computing infrastructures based on various micro-economic models such as auctions, commodity markets, double-auctions, and iterative combinatorial exchange [18]–[21]. Amazon EC2 spot instance is a prime example of one of these mechanisms [22]. Customers are allowed to bid for resources, and will be allocated such resources as long as the bid is higher than the

[8] We do not include models of webserver and chains with average length 18 since they do not allow for much flexibility in a 24-hour epoch.

market price at the time of allocation. Once the demand for resources increases thus increasing the market price, Amazon redistributes these resources to other customers. Unlike EC2 spot instance, in our model, a customer is guaranteed to execute throughout the entire time of his/her allocation. More, importantly, our approach is not subject to collusion.

The work by Londono et al [23] outline a colocation game which allows for task colocation. In that setting, a customer's workload consists of a single task and colocation interactions are driven by the rational behavior of customers, who are free to relocate and choose whatever is in their best interest (minimize their workload's cost). In our Setting, a customer's workload consists of multiple tasks and we optimize the allocation of resources and apportion costs using the game-theoretic-inspired Shapley concept – what we devise is a pricing mechanism and not a game. As a result, each customer ends up paying a marginal cost.

Unlike all of the models referenced above, our framework allows for an explicit consideration of the flexibility of customers (as opposed to having such a flexibility be expressed through the strategic choices of customers).

Data center energy management: Minimizing the operating cost of data centers is a very active research topic. Along these lines, there has been significant breakthroughs in terms of optimizing the use of resources through efficient server power management [3], [11], [24], optimized workload distribution and consolidation which results either in powering off unused resources [2] or better cooling [25]. Qureshi et al [4], suggest migrating service requests from a data center to another such that the computation cost is minimized, without violating delay constraints. In [3], the authors highlight the need for coordination among different energy management approaches since in the absence of coordination, these approaches are likely to interfere with one another in unpredictable (and potentially dangerous) ways. The authors propose and validate a power management solution that utilizes control theory for coordination of different approaches. A common characteristic in the above-referenced, large body of prior work (which we emphasize is not exhaustive) is that the IaaS provider is the one who is doing the optimization, which does not provide any incentive for customers. In our model, we aim to minimize the overall operational cost of the datacenter, and provide the transparency that allows customers to move their requested computations to run during hours where energy cost is minimum.

Service Level Agreements (SLAs): There has been a significant amount of research on various topics related to SLAs. The usage of resource management in grids have been considered in [26], [27]; issues related to specification of SLAs have been considered in [28]; and topics related to the economic aspects of SLAs usage for service provisioning through negotiation between consumers and providers are considered in [29], [30]. An inherent assumption in such systems is that the customer's SLAs are immutable. We break that assumption by allowing the customer to provide multiple yet functionally equivalent forms of SLAs. Our framework utilizes this degree of freedom to achieve significantly better colocation.

Languages and execution environments: Workflow/dataflow languages have been proposed since the sixties, with IBM job control language [31] a prime example. Since then, different languages and execution engines have been developed [32]–[35]. These languages modeled coordination or dependencies among tasks as DAGs. In these models, task dependencies were defined in terms of data dependency. In our model, workloads are defined in terms of resource requests and dependencies are modeled in terms of temporal tolerance or flexibility of the customer.

Lubin et al [36] describe resource requests in the form of a tree based bidding language (TBBL), where resources are mapped to the leaves of the tree, and inner nodes model logical operations among customer requests. Although we believe that our model can be described using TBBL, such description would be inefficient due to the exponential increase in the number of nodes as a result of a customer's variability.

Workflow scheduling: Different workflow management and scheduling tools have been proposed that focus on scheduling DAGs with the purpose of optimizing the makespan and consider QoS properties like deadlines and/or budget constraints [16], [37]–[39]. Henzinger et al [40] provide a static scheduling framework that is based on small state abstractions of large workloads. Similarly to previous work, Our model aims to minimize the overall operational cost of the datacenter. However, our scheduling framework utilizes customer flexibility to achieve a better optimization of workloads. In addition, we provide a fair pricing mechanism which distributes the cost of leasing resource over customers and provides them with the incentive to declare their flexibility.

VI. Conclusion

In this work, we proposed a new pricing model for cloud resources that better reflects the costs incurred by IaaS providers, and gives cloud customers the opportunity and incentive to take advantage of any scheduling flexibilities they might have. We presented the architecture and algorithmic blueprints of a generic framework for colocation of customer workloads. Our framework provides (1) a resource specification language that allows customers to formally express their flexibility, (2) an algorithm that optimizes the use of cloud resources, and (3) a game-theoretic inspired pricing mechanism that achieves a rationally fair distribution of incurred costs over customers. We presented performance evaluation results that confirm the utility and potential of our framework.

Our on-going research work is pursued along three dimensions. Along the first, we are investigating extensions to our SLA model to allow for yet more expressive forms of SLAs – *e.g.*, non-parametric constraints, such as geographic location. Our second line of work is focusing on extending our model to allow for resource allocation with uncertainty, i.e., account and provide cost for resource failures. Our third line of work is focusing on providing customers with a *choice* construct that allows them to specify alternative workload configurations and physical resource flexibilities.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," UCB, Tech. Rep. UCB/EECS-2009-28, Feb 2009.

[2] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," in *Proceedings of the 7th USENIX conference on Networked systems design and implementation*. USENIX Association, 2010, p. 17.

[3] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "power" struggles: coordinated multi-level power management for the data center," *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems*, pp. 48–59, 2008.

[4] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for internet-scale systems," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. ACM, 2009, pp. 123–134.

[5] A. R.-T. Prices. (2011, March) Real-time prices. https://www2.ameren.com/RetailEnergy/realtimeprices.aspx. [Online]. Available: https://www2.ameren.com/RetailEnergy/realtimeprices.aspx

[6] F. Ahmad and T. N. Vijaykumar, "Joint optimization of idle and cooling power in data centers while maintaining response time," in *ASPLOS '10*. New York, NY, USA: ACM, 2010, pp. 243–256.

[7] Amazon.com, Inc., "Amazon Elastic Computing Cloud (Amazon EC2)," http://aws.amazon.com/ec2/, Oct 2010.

[8] Rackspace.com, "The Rackspace Cloud," http://www.rackspacecloud.com/, Oct 2010.

[9] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," *CCR Online*, Jan 2009.

[10] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th annual international symposium on Computer architecture*, ser. ISCA '07. New York, NY, USA: ACM, 2007, pp. 13–23.

[11] A. Gandhi, M. Harchol-Balter, R. Das, and C. Lefurgy, "Optimal power allocation in server farms," in *SIGMETRICS '09*. New York, NY, USA: ACM, 2009, pp. 157–168.

[12] P. Ranganathan, P. Leech, D. Irwin, and J. Chase, "Ensemble-level power management for dense blade servers," in *Computer Architecture, 2006. ISCA '06. 33rd International Symposium on*, 0-0 2006, pp. 66 –77.

[13] N. Nisan, *Algorithmic game theory*. Cambridge Univ Press, 2007.

[14] R. Stanojevic, N. Laoutaris, and P. Rodriguez, "On economic heavy hitters: Shapley value analysis of 95th-percentile pricing," in *Proceedings of the 10th annual conference on Internet measurement*. ACM, 2010, pp. 75–80.

[15] J. Castro, D. Gómez, and J. Tejada, "Polynomial calculation of the shapley value based on sampling," *Computers & Operations Research*, vol. 36, no. 5, pp. 1726–1730, 2009.

[16] J. Yu, R. Buyya, and C. Tham, "Cost-based scheduling of scientific workflow application on utility grids," in *Proceedings of the First International Conference on e-Science and Grid Computing*. IEEE Computer Society, 2005, pp. 140–147.

[17] J. Koomey, "Estimating total power consumption by servers in the us and the world," 2007.

[18] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599 – 616, 2009.

[19] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik, "G-commerce: Market Formulations Controlling Resource Allocation on the Computational Grid," *IPDPS*, 2001.

[20] A. AuYoung, P. Buonadonna, B. N. Chun, C. Ng, D. C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat, "Two auction-based resource allocation environments: Design and experience," in *Market Oriented Grid and Utility Computing*, R. Buyya and K. Bubendorfer, Eds. Wiley, 2009, ch. 23.

[21] D. Parkes, R. Cavallo, N. Elprin, A. Juda, S. Lahaie, B. Lubin, L. Michael, J. Shneidman, and H. Sultan, "ICE: An iterative combinatorial exchange," in *Proceedings of the 6th ACM conference on Electronic commerce*. ACM, 2005, pp. 249–258.

[22] Amazon.com, Inc., "Amazon EC2 Spot Instances," http://aws.amazon.com/ec2/spot-instances/, May 2011.

[23] J. Londoño, A. Bestavros, and S.-H. Teng, "Collocation Games And Their Application to Distributed Resource Management," in *HotCloud'09: Workshop on Hot Topics in Cloud Computing*. USENIX, 2009.

[24] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle, "Managing energy and server resources in hosting centers," in *SOSP '01*. New York, NY, USA: ACM, 2001, pp. 103–116.

[25] L. Parolini, B. Sinopoli, and B. Krogh, "Reducing data center energy consumption via coordinated cooling and load management," in *Proceedings of the 2008 conference on Power aware computing and systems*. USENIX Association, 2008, p. 14.

[26] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science, D. Feitelson, L. Rudolph, and U. Schwiegelshohn, Eds., vol. 2537. Springer Berlin / Heidelberg, 2002, pp. 153–183.

[27] M. A. Netto, K. Bubendorfer, and R. Buyya, "SLA-Based Advance Reservations with Flexible and Adaptive Time QoS Parameters," in *Proceedings of the 5th international conference on Service-Oriented Computing*, ser. ICSOC '07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 119–131.

[28] A. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *J. Netw. Syst. Manage.*, vol. 11, pp. 57–81, March 2003.

[29] A. Barmouta and R. Buyya, "GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration," in *Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '03. Washington, DC, USA: IEEE Computer Society, 2003.

[30] L.-O. Burchard, M. Hovestadt, O. Kao, A. Keller, and B. Linnert, "The Virtual Resource Manager: An Architecture For SLA-aware Resource Management," in *Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, ser. CCGRID '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 126–133.

[31] IBM, "Job Control Language," http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.zcourses/zcourses_jclintro.htm, May 2011.

[32] D. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand, "CIEL: a universal execution engine for distributed dataflow computing," in *Proceedings of NSDI*, 2011.

[33] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: distributed data-parallel programs from sequential building blocks," in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. ACM, 2007, pp. 59–72.

[34] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[35] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus: Mapping scientific workflows onto the grid," in *Grid Computing*, ser. Lecture Notes in Computer Science, M. Dikaiakos, Ed. Springer Berlin / Heidelberg, 2004, vol. 3165, pp. 131–140.

[36] B. Lubin, A. I. Juda, R. Cavallo, S. Lahaie, J. Shneidman, and D. C. Parkes, "Ice: an expressive iterative combinatorial exchange," *J. Artif. Int. Res.*, vol. 33, pp. 33–77, September 2008.

[37] L. Ramakrishnan, J. S. Chase, D. Gannon, D. Nurmi, and R. Wolski, "Deadline-sensitive workflow orchestration without explicit resource control," *J. Parallel Distrib. Comput.*, vol. 71, pp. 343–353, March 2011.

[38] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. Dikaiakos, "Scheduling workflows with budget constraints," *Integrated Research in Grid Computing*, pp. 189–202, 2007.

[39] A. Mandal, K. Kennedy, C. Koelbel, G. Marin, J. Mellor-Crummey, B. Liu, and L. Johnsson, "Scheduling strategies for mapping application workflows onto the grid," in *Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 125–134.

[40] T. Henzinger, V. Singh, T. Wies, and D. Zufferey, "Scheduling large jobs by abstraction refinement," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 329–342.