CloudPack[⋆]
Exploiting Workload Flexibility Through Rational Pricing

Vatche Ishakian, Raymond Sweha, Azer Bestavros, Jonathan Appavoo

Computer Science Department, Boston University
Boston, MA 02215, USA
{visahak,remos,best,jappavoo}@cs.bu.edu

Abstract. Infrastructure as a Service pricing models for resources are meant to reflect the operational costs and profit margins for providers to deliver virtualized resources to customers subject to an underlying Service Level Agreements (SLAs). While the operational costs incurred by providers are dynamic – they vary over time depending on factors such as energy cost, cooling strategies, and aggregate demand – the pricing models extended to customers are typically fixed – they are static over time and independent of aggregate demand. This disconnect between the dynamic cost incurred by a provider and the fixed price paid by a customer results in an economically inefficient marketplace. In particular, it does not provide incentives for customers to express workload scheduling flexibilities that may benefit them as well as providers. In this paper, we utilize a dynamic pricing model to address this inefficiency and give customers the opportunity and incentive to take advantage of any flexibilities they may have regarding the provisioning of their workloads. We present CloudPack: a framework for workload colocation, which provides customers with the ability to formally express workload flexibilities using Directed Acyclic Graphs, optimizes the use of cloud resources to minimize total costs while allocating clients' workloads, and utilizes Shapley valuation to rationally – and thus fairly in a game-theoretic sense – attribute costs to the customers. Using extensive simulation, we show the practical utility of our CloudPack colocation framework and the efficacy of the resulting marketplace in terms of cost savings.

Keywords: Cloud computing, resource provisioning, scheduling

1 Introduction

Motivation: Cloud computing has emerged as compelling paradigms for the deployment of distributed applications and services on the Internet. Critical to this, are Infrastructure as a Service (IaaS) providers which own and maintain large physical datacenter installations and use virtualization technologies to provide customers with resources in the form of Virtual Machines. By relying on

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

virtualized resources, customers are able to easily deploy, scale up or down their applications [3].

IaaS providers incur a significant capital investment as part of creating and providing such services. A data center's return on investment (profit) relies heavily on decreasing its overall cost through efficient cooling and energy conservation [16, 33], while increasing its overall utilization (Revenue) 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 [1, 32], while the cost of cooling might be higher during peak utilization times. The location of allocated virtual resources in the data center can also be a crucial factor in cost reduction. An efficient allocation can lead to powering down of resources [16], or in decreased cost of cooling [2]. These approaches are but examples of what providers must consider in order to decrease their overall costs.

Problem Description: Despite the complexities associated with minimizing the overall cost of cloud providers, 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 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.[1] 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 flexibility.

Under a fixed pricing model, customers do not have any incentive to expose (or the means to capitalize on) the flexibility of their workloads. By workload flexibility, we refer to scheduling flexibilities that customers may be able to tolerate, such as requesting a virtual machine for backup operations which can run anytime during a day. This customer-side *demand flexibility* could be seen as an asset that may benefit *both* customers and providers. From the provider's perspective, demand flexibility could be seen as an additional lever in the aforementioned optimization of operational costs, whereas from the customer's perspective, demand flexibility could be seen as a feature of their workloads that should translate to cost savings. Fixed pricing models do not enable demand flexibility to play a role in the marketplace, effectively resulting in an inefficient marketplace [24].

Leveraging customer-side demand flexibility requires the development of dynamic (as opposed to fixed) pricing mechanisms and associated flexible SLA models that provide customers with proper incentives and assurances. In partic-

[1] Amazon spot instance is a prime example of flexible pricing, but unlike our CloudPack framework, it does not provide customers any guarantees in terms of when and for how long a customer's demand is going to be honored.

ular, the pricing mechanism must provably reward (and certainly never mistreat) customers for expressing the scheduling flexibilities in their workloads.

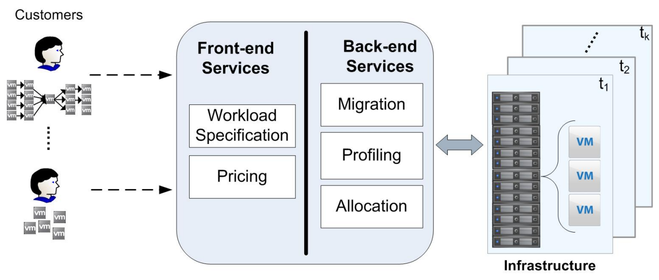

Fig. 1. CloudPack Colocation Framework

Scope and Contribution: In this paper, we present CloudPack: (see Section 3) a colocation framework that achieves the above-stated goals by giving customers both the means and the incentive to express any flexibilities they may have regarding the provisioning of their workloads. Architecturally, our framework can be described as illustrated in Figure 1: it consists of two major services, *Back-end services* and *Front-end services*. The CloudPack framework can be incorporated into an offering by cloud providers; it can be implemented as a value-added proposition or as a secondary market by IaaS resellers; or it can be directly leveraged in a peer-to-peer fashion by IaaS customers.

Front-end services are exposed to the IaaS customers and consists of two components: Workload Specification Component (Section 3.1), and Pricing Component (Section 3.4). The workload specification component provides customers not only the ability to state their requests in terms of virtualized resources subject to SLAs, but also to express their allocation flexibilities represented as Directed Acyclic Graphs (DAGs). The pricing component 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.

Back-end services are oblivious to the IaaS customers and are utilized by the provider to control its resources. The Back-end services consist of the following components: An Allocation Component (Section 3.2) that colocates workloads (virtual resource requests) from multiple customers on the same set of physical resources. The main objective of the Allocation component is with the aim of minimize the total cost of used IaaS resources, while adhering to customers' SLAs provided using the Workload Specification Component. Profiling or monitoring Component whose main purpose is to provide customers with the raw data that enables them to adjust their reservations as well as gaining insight and visibility into resource utilization, overall performance. Finally the migration component is used to eliminate hotspots, enable load balancing, and allow for physical resource maintenance.

Profiling [8, 13, 37, 39] and Migration [20, 21, 25, 27] Components have been extensively studied in the literature and are implemented as standard features in widely popular virtualization technologies such as Xen and VMware, thus we consider them to be beyond the scope of this work.

To demonstrate the promise of using CloudPack framework to manage the colocation of different workloads, using simulation (Section 4), we perform an extensive experimental evaluation of our framework using synthetically generated workloads, selected from a set of representative real workload models. The results 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.

2 CloudPack: Background & Setting

In this section, we present an IaaS resource cost model utilized by CloudPack along with assumptions about the underlying IaaS setting needed to instantiate our colocation framework.

2.1 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 [3, 15]. In addition, it is becoming a norm for datacenters to be charged a variable hourly rate for electricity [32], or for peak usage [15]. 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 [12, 14, 33] suggest that a physical machine's power consumption increases linearly with the system load, with a base idle power draw – power consumed by an idle physical machine – of 60%. Under this simple model one can already observe a generic notion of fixed and variable costs. In addition, Ranganathan et al. [35] 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 physical resource j during time t is:

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

where P_j reflects an amortized 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,[2] 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.[3] Note that $f(t, U_j(t))$ has also a fixed part reflecting the cost of operating the resource if the resource is turned on and is in an idle state.

2.2 IaaS Setting

As an underlying infrastructure for CloudPack, we assume an IaaS 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. The cost of resources follows the IaaS resource cost model presented in the previous section.

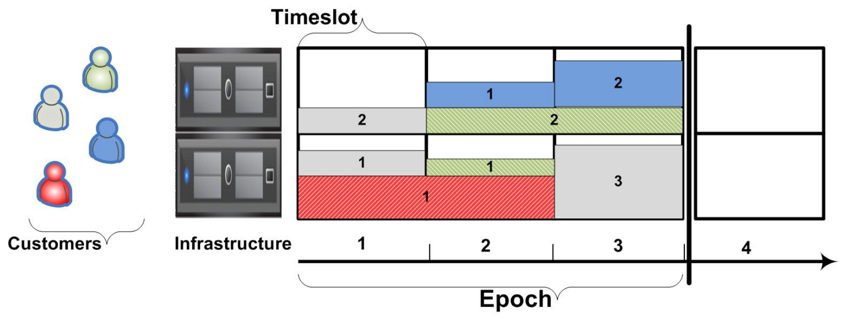

Fig. 2. CloudPack Epoch Example

A fundamental principle in the instantiation of our colocation framework is the concept of epochs. We consider an epoch to be a sequence of periodic timeslots during which the workloads of customers can be colocated. The determination of colocation configurations is calculated at the beginning of an epoch, and is fixed for the entire duration of that epoch. Figure 2 illustrates an example epoch consisting of three timeslots, through which customers' requests (virtual machines) are allocated on the physical machines.

Customers who are not able to join at the beginning of an epoch will only be considered for colocation during the next epoch. Similar to grid markets, we envision different marketplaces operating at different timescales, with epochs ranging from days to weeks to months. One way to minimize customer wait time is to instantiate marketplaces with overlapping epochs of the same duration. Another method would be to have multiple marketplaces of epochs with exponentially increasing time scales, where a customer can colocate in a logarithmic number of shorter time-scale epochs before reaching the epoch he desires to join [18].

[2] Throughout this paper, we take v_0 to be 60% [12, 14, 33].

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

3 CloudPack: The Framework

In this section, we present the three major components of the CloudPack colocation framework: Workload Specification, Allocation, and Pricing.

3.1 CloudPack: Workload Specification Component

We propose an expressive resource specification language for customer workloads, which allows them to declare their quantitative resource requirements as well as any associated temporal flexibilities.[4] Our resource specification language is XML based, we omit the syntax due to space constraints. A workload is represented as a DAG. A node in the graph represents a single task (virtual machine), to be mapped to a resource, and consumes some of the resource dimensions. A task has two attributes: The total number d of timeslots (periods) during which the task must remain on the same resource, and a quantitative resource request matrix $V \in \mathbb{R}^{m \times d}$ where d represents the required duration and m represents the different dimensions requested during each period. The directed edges in the graph represent the temporal dependencies between tasks. An edge between node k and k' dictates that task k needs to finish execution before task k' 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 k'. 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 3 (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 five 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 execution 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 of size $m \times 24$ that varies accordingly. Figure 3 (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.

The above example illustrates a scenario in which the customer has no scheduling flexibilities. Figure 3 (c) illustrates a typical MapReduce workload, where a scheduling task needs to execute, followed by a set of independent *map* tasks, and finishing with a *reduce* task. Figure 3 (d) is a constrained version of

[4] 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.

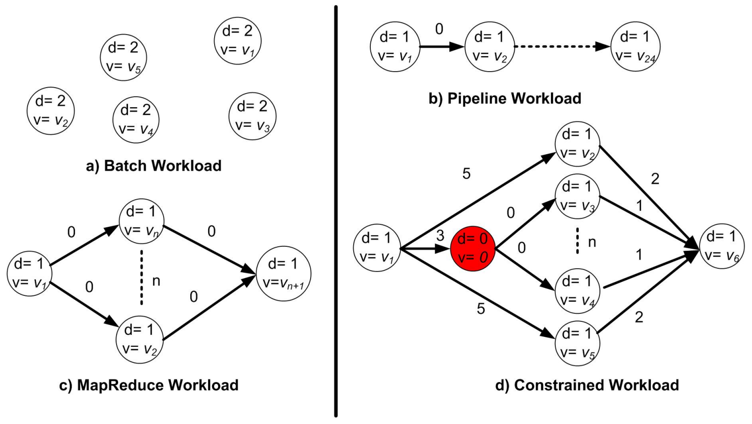

Fig. 3. An example illustrating different workload models.

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. This feature is essential for High Performance Computing workloads.

Note that current customers of cloud offerings such as Amazon need to specify and map their actual applications to resource requests as part of their adequate resource reservation (e.g. small, medium, large). Profiling and benchmarking techniques such as the ones described in [8, 39] can be used to predict an applications resource consumption.

3.2 CloudPack: Allocation Component

In the previous section, we presented our workload specification language, which allows IaaS customers to describe their workloads. In this section, we formulate the allocation problem and present a linear programming optimization solution. The objective of the system is to fulfill the requests of all customers, taking into consideration their flexibility (constraints) 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

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

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

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

Subject to:

$$\sum_{\forall l} X(j,t,k,l) \leq Y(t,j) \qquad \forall t,j,k \tag{2}$$

$$\sum_{\forall k, 1 \leq l \leq d_k} X(j,t,k,l) \times u(k,l) \leq R_j \qquad \forall j,t \tag{3}$$

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

$$X(j,t,k,l) = X(j,t+1,k,l+1) \qquad \forall j,t,k \in V, 1 \leq l < d_k \tag{5}$$

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

$$X(j,t,k,l) = 0 \qquad \forall j,k \in V_i, t > T_i^e, 1 \leq l \leq d_k \tag{7}$$

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

$$\sum_{j} X(j,t',k,d_k) \leq \sum_{j',t'<t\leq t'+W_e+1} X(j',t,k',1) \qquad \forall t', (k,k') \in E \tag{9}$$

where P_j and R_j are the cost and capacity of a specific physical resource j, $u(k,l)$ is the utilization request of a nodes k's duration l, $U_j(t)$ is the total utilization of resource j at time t is formally defined as $(\sum_{\forall k,1\leq l\leq d_k} X(j,t,k,l) \times u(k,l))/R_j$, v_0 is the energy consumed by resource j while idle, and $\alpha(t)$ is the cost of energy at time t. 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 while keeping in line with each customer's specified flexibility. 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 if that resource is in use at time selected at time t, and $\alpha(t) \times (1-v_0)U_j(t) \times R_j$ stands for the additional (variable) cost as a consequence for utilizing the resource.[5]

Equation (2) ensures that a resource j is utilized at time t, by setting $Y(j,t)$ to one if that resource is used to serve the requests of any customer during

[5] We do not multiply the third component of Equation (1) by $Y(t,j)$, since if the resource j is not assigned during time t, then its $U_j(t) = 0$.

that time. Equation (3) ensures that the utilization of a single resource does not exceed a fixed capacity R_j. This constraint is needed not to overprovision the resources. Equation (4) guarantees that all periods of each task are fulfilled exactly once. Equation (5) ensures that a task's periods are allocated consecutively on the same resource. This constraint is essential for fulfilling requirements of workloads such as a WebServer. Equation (6) and (7) 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 (8) and (9), guarantee that the allocation of resources respects the client's edge constraints (flexibility). In particular, Equation (8) constrains the allocation of the first timeslot of a request k' to follow the resources allocated to the last timeslot of request k, while Equation (9) guarantees that such an allocation happens within the specified client's delay W_e on edge (k, k').

3.3 CloudPack: Greedy Heuristic

The optimization problem defined in the previous section is a variant of mixed-integer programming, which is known to be NP-hard in general[6]. Therefore, in this section, we propose a greedy algorithm that results in solutions to our allocation 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 final solution is the configuration based on which physical resources are going to be allocated to the customers.

The initial solution is generated 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 (iteration), the algorithm chooses a workload which has the highest current-to-optimal cost ratio r among all customer workloads. Calculating the 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 utopiancost 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.

Once the workload with the highest r is identified, we proceeds to relocate it such that r is minimized. If the relocation results in reducing the total cost of the solution, then the relocation (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.

3.4 CloudPack: Pricing Component

The allocation component is designed to minimize the total aggregate cost of using resources. However, we need a pricing component to apportion (distribute)

[6] The proof of NP-hardness is omitted due to space limitations.

this total cost across all customers. This component requires an appropriate pricing mechanism, which ensures 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 flexibility.

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. For all costs of $c_h > 2 \times c_l$, a proportional share pricing mechanism would divide the total cost across both customers, thus forcing unfairly customer B to pay more than what he/she would have paid (c_l) had he/she run by herself at the lowest 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 [29], 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:

$$\phi_u(C) = \frac{1}{N!} \sum_{\pi \in S_N} (C(S(\pi, u)) - C(S(\pi, u) \setminus u)) \tag{10}$$

where $S(\pi, u)$ 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 (10) 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 using sampling.[7] We utilize Castro's [7] 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 the vector of estimated Shapley values based on all possible $N!$ permutations be $Sh = (\phi_1(C), \phi_2(C), \cdots \phi_n(C))$; Let the vector of estimated Shapley values based on m sample permutations be $\hat{Sh} = (\hat{\phi}_1(C), \hat{\phi}_2(C), \cdots, \hat{\phi}_n(C))$. 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,

[7] Estimating Shapley value has proven to be effective in calculating the contribution of customers to the effective network peak demand [36].

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 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.

4 CloudPack: Experimental Evaluation

In this section, we present results from extensive experimental evaluations of CloudPack 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 distribution of costs among customers, by (1) establishing the efficacy of our greedy heuristic by comparing it to optimally allocated workloads, (2) evaluating the cost incurred by the customer

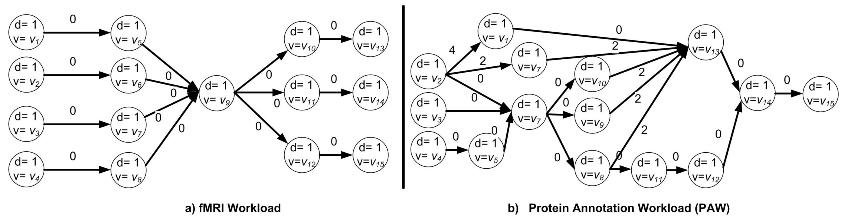

Fig. 4. High Performance Computing Workloads

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 3), 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.[8] In addition, we enrich our set of workloads with two additional High Performance Computing workloads (c.f. Figure 4) for Protein annotation workflow (*PAW*), and Cognitive Neuroscience (*fMRI*) [40]. We believe that this set of workload models is representative for many cloud based applications. We assume homogeneous resources with the fixed cost part equal to 10 cents per hour, a resource capacity equal to one, and an epoch consisting of twenty four hours where customers configurations are calculated at the beginning of the epoch. 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 [23], specifically for mid-range server, we assume that a physical resource's power consumption is 500 watts per hour.

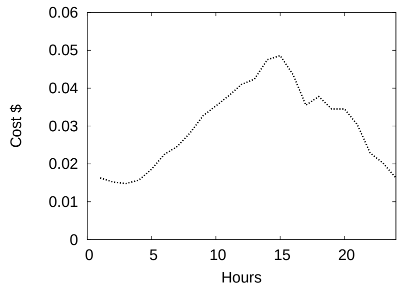

Fig. 5. Energy Cost (KW/H)

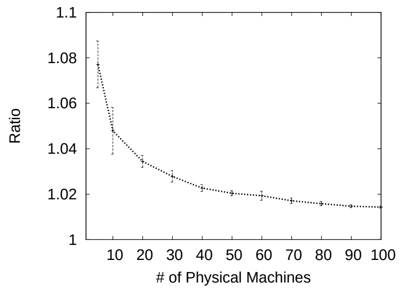

Fig. 6. Packing Ratio (Heuristic/Optimal)

Energy Cost: To model the energy cost for our framework, we use real energy costs from the Ameren website [1]. Ameren publishes energy costs daily on an hourly basis. We get energy cost for a one month period (from 08/01/11 to

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

08/31/11) and average them per hour. Figure 5 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 allocation of tasks. Since knowing an optimal allocation is difficult (bin packing is NP-hard), we resort to generating workloads for which we know (by construction) that an optimal allocation exists.

We do so by simulating a set of physical machines for the duration of an epoch, and repeatedly creating fragments that sum up to a physical machine's full capacity. We generate fragments based on a uniform distribution between zero and one, thus the average number of fragments per resources is two.[9] Similar results for physical machine's fragmentation were observed given other distributions but were omitted due to lack of space. We proceed in a round-robin fashion over the set of workload models in our disposal (except the batch), and greedily embed each workload over the physical machines. 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" allocation 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 allocation. Therefore, we assume that the cost of electricity is fixed (i.e., independent of time).

Figure 6 shows the ratio of allocation achieved using our algorithm relative to an optimal allocation. The x-axis shows the number of physical machines used, and the y axis shows the ratio of workload allocation achieved using our heuristic over that of an optimal allocation. 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 physical machines, the ratio decreases.

Fair pricing scheme vs. utopian customer cost: Unlike the previous experiment, which aimed to show the efficacy of our heuristic by comparing its performance to an optimally-allocated 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 model: HPC (fMRI, PAW), Web-Server, 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

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

to be the start time plus the length of execution of the workload. This is an easy step since all of the workloads 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.

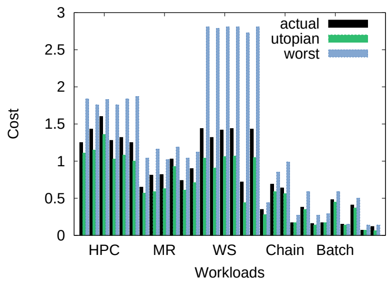

Fig. 7. Per workload cost comparison

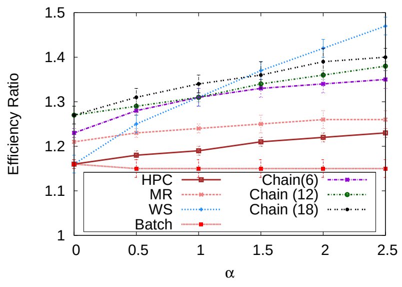

Fig. 8. Effect of energy fluctuation on workload cost

Figure 7 shows the distribution of costs based on sampled Shapley value for 30 workloads, where all workload models 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.

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 5, and modulate it by multiplying it with α, where α varies between 0 and 2.5. For each workload model, we generate 50 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 ratio as the ratio between the actual customer cost over the utopian cost. Figure 8 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 5. 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.

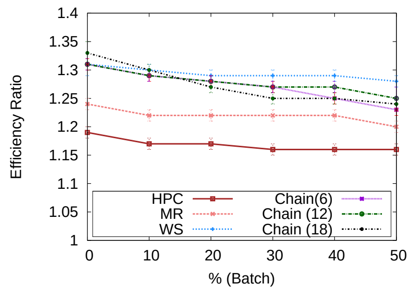

Fig. 9. Workloads with batch mix

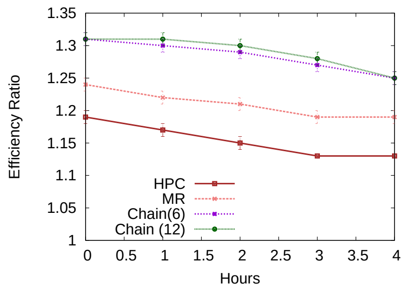

Fig. 10. Effect of Flexibility

Given the fluidity (maximal flexibility) 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, we mix it with different percentages of batch workloads. Figure 9 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 allocation outcome is due to the existence of parallel branches in MR and HPC models, which provides these workloads – unlike chain and webserver workloads – an additional opportunity for allocation.

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 10 shows the effect of customer flexibility on workloads.[10] As expected, the more flexible a workload is, the better the efficiency ratio.

5 Related Work

Economic models for resource management: Several resource management techniques have been proposed for large-scale computing infrastructures using

[10] 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.

various micro-economic models such as auctions, commodity markets, and iterative combinatorial exchange [4,6,30,38]. Amazon EC2 spot instance is a prime example of one of these markets. Customers bid for resources, and will be allocated such resources as long as their bid is higher than the market price at the time of allocation. Unlike EC2 spot instance which does not provide an SLA regarding the allocation period, in CloudPack, customers are guaranteed to execute throughout the entire time of their allocation.

Ishakian et al, [20] develop a colocation service which allows for migration, profiling and allocation of workloads. In that setting, a customer's workload consists of a *single* task and interactions are driven by the rational behavior of customers, who are free to relocate as long as the relocation minimize their cost. In this 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, CloudPack 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 [14,33], optimized workload distribution and consolidation [16,32] or better cooling [31]. The authors in [33] motivate the need for coordination among different energy management approaches because they may interfere with one another in unpredictable (and potentially dangerous) ways. They present 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 is that the IaaS provider 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 flexible customers to take advantage of their flexibility.

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 [17,26,34,40]. Henzinger et al [17] provide a static scheduling framework that is based on small state abstractions of large workloads, Similar to previous work, Our model aims to minimize the overall operational cost of the datacenter. However, we provide a provably fair pricing mechanism which distributes the cost of leasing resource over customers and provides them with the incentive to declare their flexibility.

Service Level Agreements: There has been significant amount of research on various topics related to SLAs. The usage , specification, and economic aspects of resource management in grids have been considered in [5,9,22,28]. 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 a better colocation.

Languages and execution environments: Workflow/dataflow languages have been proposed since the sixties, with IBM job control language [19] a prime example. Since then, different languages and execution engines have been developed [10, 11, 30]. These languages modeled coordination or dependencies among tasks (programs) as DAGs. Task dependencies reflect data dependencies between tasks. In our language, workloads define resource requests and dependencies are model customer temporal tolerance or flexibility.

Parkes et al [30] outline a tree based bidding language (TBBL), where resources are mapped to the leaves of the tree, and inner nodes model logical operations. TBBL can be used to describe customer requests, however, a such description would be inefficient due to the exponential increasing number of nodes resulting from a customer's flexibility.

6 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 CloudPack: a 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 specification language to allow for yet more expressive forms of SLAs – *e.g.*, non-parametric constraints, such as geographic location, anti-colocation, and network proximity, as well as providing customers with a *choice* construct that allows them to specify alternative workload configurations and physical resource flexibilities. Our second line of work is focusing on extending CloudPack to allow for resource allocation with uncertainty, i.e., account and provide cost for resource failures. Our third line of work is focused on developing a prototype of a our colocation framework that will allow us to conduct experiments in a dynamic setting that is subject to the overheads resulting from actual allocation and relocation of workloads. Elements of this prototype have been developed as part of our earlier work on XCS a VM cloud colocation service [20].

References

1. Ameren real-time prices.
https://www2.ameren.com/RetailEnergy/realtimeprices.aspx (March 2011)

2. Ahmad, F., Vijaykumar, T.N.: Joint optimization of idle and cooling power in data centers while maintaining response time. In: ASPLOS '10. pp. 243–256. ACM, New York, NY, USA (2010)
3. Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., et al.: A view of cloud computing. Communications of the ACM 53(4) (2010)
4. AuYoung, A., Buonadonna, P., Chun, B.N., Ng, C., Parkes, D.C., Shneidman, J., Snoeren, A.C., Vahdat, A.: Two auction-based resource allocation environments: Design and experience. In: Buyya, R., Bubendorfer, K. (eds.) Market Oriented Grid and Utility Computing, chap. 23. Wiley (2009)
5. Barmouta, A., Buyya, R.: GridBank: A Grid Accounting Services Architecture (GASA) for Distributed Systems Sharing and Integration. In: IPDPS '03. IEEE Computer Society, Washington, DC, USA (2003)
6. Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. Future Generation Computer Systems 25(6) (2009)
7. Castro, J., Gómez, D., Tejada, J.: Polynomial calculation of the shapley value based on sampling. Computers & Operations Research 36(5) (2009)
8. Chen, J., Wang, C., Zhou, B.B., Sun, L., Lee, Y.C., Zomaya, A.Y.: Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In: HPDC. New York, NY, USA (2011)
9. Czajkowski, K., Foster, I., Kesselman, C., Sander, V., Tuecke, S.: SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) Job Scheduling Strategies for Parallel Processing. Lecture Notes in Computer Science, vol. 2537. Springer Berlin / Heidelberg (2002)
10. Dean, J., Ghemawat, S.: MapReduce: Simplified data processing on large clusters. Communications of the ACM 51(1) (2008)
11. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M., Vahi, K., Livny, M.: Pegasus: Mapping scientific workflows onto the grid. In: Grid Computing. pp. 131–140. Springer (2004)
12. Fan, X., Weber, W.D., Barroso, L.A.: Power provisioning for a warehouse-sized computer. In: ISCA. New York, NY, USA (2007)
13. Ferguson, A.D., Bodík, P., Kandula, S., Boutin, E., Fonseca, R.: Jockey: guaranteed job latency in data parallel clusters. In: EuroSys. pp. 99–112 (2012)
14. Gandhi, A., Harchol-Balter, M., Das, R., Lefurgy, C.: Optimal power allocation in server farms. In: SIGMETRICS '09. ACM, New York, NY, USA (2009)
15. Greenberg, A., Hamilton, J., Maltz, D.A., Patel, P.: The cost of a cloud: Research problems in data center networks. CCR Online (Jan 2009)
16. Heller, B., Seetharaman, S., Mahadevan, P., Yiakoumis, Y., Sharma, P., Banerjee, S., McKeown, N.: ElasticTree: Saving energy in data center networks. In: NSDI (2010)
17. Henzinger, T., Singh, V., Wies, T., Zufferey, D.: Scheduling large jobs by abstraction refinement. In: Proceedings of the sixth conference on Computer systems. pp. 329–342. ACM (2011)
18. Hua, K., Sheu, S.: Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems. In: ACM SIGCOMM Computer Communication Review. vol. 27. ACM (1997)
19. IBM: Job Control Language. http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp?topic=/com.ibm.zos.zcourses/zcourses_jclintro.htm (May 2011)

20. Ishakian, V., Sweha, R., Londoño, J., Bestavros, A.: Colocation as a Service. Strategic and Operational Services for Cloud Colocation. In: IEEE NCA (2010)
21. Keir, C.C., Clark, C., Fraser, K., H, S., Hansen, J.G., Jul, E., Limpach, C., Pratt, I., Warfield, A.: Live Migration of Virtual Machines. In: NSDI (2005)
22. Keller, A., Ludwig, H.: The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. J. Netw. Syst. Manage. 11, 57–81 (March 2003)
23. Koomey, J.: Estimating total power consumption by servers in the us and the world (2007)
24. Lai, K.: Markets are dead, long live markets. SIGecom Exch. 5(4), 1–10 (Jul 2005)
25. Liu, H., Jin, H., Liao, X., Hu, L., Yu, C.: Live Migration of Virtual Machine Based on Full System Trace and Replay. In: Proc. of the 18th ACM HPDC (2009)
26. Mandal, A., Kennedy, K., Koelbel, C., Marin, G., Mellor-Crummey, J., Liu, B., Johnsson, L.: Scheduling strategies for mapping application workflows onto the grid. In: HPDC. Washington, DC, USA (2005)
27. Nelson, M., Lim, B.H., Hutchins, G.: Fast Transparent Migration for Virtual Machines. In: ATEC '05: Proceedings of the annual conference on USENIX Annual Technical Conference. pp. 25–25. USENIX Association, Berkeley, CA, USA (2005)
28. Netto, M.A., Bubendorfer, K., Buyya, R.: SLA-Based Advance Reservations with Flexible and Adaptive Time QoS Parameters. In: Proceedings of the 5th international conference on Service-Oriented Computing (2007)
29. Nisan, N.: Algorithmic game theory. Cambridge Univ Press (2007)
30. Parkes, D., Cavallo, R., Elprin, N., Juda, A., Lahaie, S., Lubin, B., Michael, L., Shneidman, J., Sultan, H.: ICE: An iterative combinatorial exchange. In: Proceedings of the 6th ACM conference on Electronic commerce. pp. 249–258. ACM (2005)
31. Parolini, L., Sinopoli, B., Krogh, B.: Reducing data center energy consumption via coordinated cooling and load management. In: USENIX conference on Power aware computing and systems (2008)
32. Qureshi, A., Weber, R., Balakrishnan, H., Guttag, J., Maggs, B.: Cutting the electric bill for internet-scale systems. In: SIGCOMM. pp. 123–134 (2009)
33. Raghavendra, R., Ranganathan, P., Talwar, V., Wang, Z., Zhu, X.: No "power" struggles: coordinated multi-level power management for the data center. in ASPLOS '08 pp. 48–59 (2008)
34. Ramakrishnan, L., Chase, J.S., Gannon, D., Nurmi, D., Wolski, R.: Deadline-sensitive workflow orchestration without explicit resource control. J. Parallel Distrib. Comput. 71, 343–353 (March 2011)
35. Ranganathan, P., Leech, P., Irwin, D., Chase, J.: Ensemble-level power management for dense blade servers. In: ISCA. pp. 66 –77 (2006)
36. Stanojevic, R., Laoutaris, N., Rodriguez, P.: On economic heavy hitters: Shapley value analysis of 95th-percentile pricing. In: Proceedings of the 10th annual conference on Internet measurement. ACM (2010)
37. Verma, A., Cherkasova, L., Campbell, R.: Resource provisioning framework for mapreduce jobs with performance goals. Middleware 2011 pp. 165–186 (2011)
38. Wolski, R., Plank, J.S., Bryan, T., Brevik, J.: G-commerce: Market Formulations Controlling Resource Allocation on the Computational Grid. IPDPS (2001)
39. Wood, T., Cherkasova, L., Ozonat, K., Shenoy, P.: Profiling and modeling resource usage of virtualized applications. In: ACM/IFIP/USENIX Middleware. New York, NY, USA (2008)
40. Yu, J., Buyya, R., Tham, C.: Cost-based scheduling of scientific workflow application on utility grids. In: Proceedings of the First International Conference on e-Science and Grid Computing. pp. 140–147. IEEE Computer Society (2005)