Cloud Resource Allocation Games

Virajith Jalaparti, Giang Nguyen, Indranil Gupta, Matthew Caesar
University of Illinois, Urbana-Champaign
{jalapar1, nguyen59, indy, caesar}@illinois.edu

***Abstract*—Cloud computing is a newly emerging paradigm in which a client pays as it uses computing resources owned by a cloud provider. Since multiple clients share the cloud's resources, they could potentially interfere with each others' tasks. Current pricing and resource allocation mechanisms are quite preliminary (e.g., fixed pricing in Amazon EC2/S3) and do not take into account the conflict of interests between multiple clients using the cloud simultaneously. This can lead to clients being overpriced, depending upon their allocated resources. Further, these mechanisms do not allow the provider to optimize its resource utilization.**

In this paper, we take the first step towards modeling the complex client-client and client-provider interactions in a cloud by using game theory. We define a new class of games called Cloud Resource Allocation Games (CRAGs). CRAGs solve the resource allocation problem in clouds using game-theoretic mechanisms, ensuring that clients are charged (near) optimal *prices* for their resource usage and that resources of the cloud are used near their optimal capacity. We present the conditions for reaching various stable equilibria in CRAGs and provide algorithms that ensure close to optimal performance. We further provide results of several experiments performed using traces from PlanetLab and the Parallel Workload Archives which show that the new mechanisms result in as much as 15% to 88% increase in performance compared to existing resource allocation mechanisms like Round-Robin.

I. Introduction

Cloud computing has recently been experiencing high rates of growth as evident from the deployment of several cloud infrastructures [1], [3]. It is based on the paradigm that clients pay only for the resources they use, on demand. Cloud providers rent their resources to multiple clients concurrently and typically charge their customers based on the amount of resources used by them. For example, Amazon EC2 [1] charges its clients per CPU-hour consumed.

However current pricing strategies are quite preliminary. They do not take into account the complex interactions between clients sharing the cloud's resources. Such interactions can occur despite the isolation guarantees provided by the use of virtualization techniques. As a result, a client's job can take longer to execute when the cloud is heavily loaded as opposed to when it is lightly loaded. If a per CPU-hour consumed pricing scheme is used (like in Amazon EC2), it could cause the client to pay more under heavy load since the presence of too many VMs can lead to interference between the clients' jobs and thus prolong job completion times. Hence, allocation mechanisms (e.g. FIFO, Round-Robin etc) used in current cloud computing environments can lead to the clients being charged unfair prices by the cloud.

Resource allocation in clouds is thus an important issue affecting not only the performance of the cloud but also the turnaround time experienced by its clients and the prices paid by them. Existing pricing and scheduling schemes provide no price-to-performance guarantees to the clients as they cannot capture the inherent conflict of interests between different clients. Clients are interested in completing their jobs in the least possible time with the least possible total cost which is the amount of money they pay the cloud for the resources used. The cloud provider, on the other hand, is interested in maximizing the resource utilization of the cloud and thus its revenue, which could contradict with the interests of its clients. These result in unintended client-client and client-provider interactions which are not captured by existing pricing and resource allocation mechanisms.

Economic-based approaches are useful to capture the complex interactions between the users of a system. They provide a *socially optimal* method to deal with consumer demand. They can be used to capture the intricacies of problems in complex, shared systems [23]. In this paper, we take the first step towards investigating market-based resource allocation mechanisms that can capture the complex interactions between multiple clients using the cloud simultaneously. In particular, we adopt a game-theoretic approach to model the resource allocation problem in clouds. This allows us to account for the inherently contradicting interests of the clients and the provider of the cloud. Economic approaches have earlier been successfully adopted in shared computing systems [7], [12], [14], [22].

In this paper, we consider a model that takes into account the interactions between the clients of a cloud and the performance seen by them when using the cloud. These important properties of real cloud environments are not accounted for in [15], the only work to have applied economic approaches in cloud computing.. We build upon existing game theoretic models for practical computing systems [8], [21] and develop a model that captures the various aspects of cloud computing, i.e., pricing, resource requests from clients, interactions between clients' jobs and the interactions between the clients and the cloud provider.

We introduce and define a new class of games called Cloud Resource Allocation Games (CRAGs). A CRAG models the resource allocation problem in clouds as a classical *non-cooperative game* [17] in which the clients of a cloud (modeled as the players of the CRAG) *selfishly* try to maximize their

This work was supported in part by NSF grants CCF-0964471 and IIS-0841765.

utility. We model the utility of a client as the negative of the cost incurred by the client which captures the total amount of money the client has to pay the cloud for the resources used. The cost also captures the turnaround time that a client experiences if the amount of money paid by it is directly proportional to its turnaround time (given a desired number of CPUs). Thus, the cost incurred by a client is also a measure of the performance obtained by it. Finally, we model the cost of the system as the sum of the costs incurred by all the clients.

The selfish nature of a client in a CRAG affects the performance seen by the other clients and can potentially decrease the overall performance of the cloud. On the other hand, the cloud provider wishes to maximize its resource utilization which can in turn allow it to increase its revenue by accommodating an increased number of clients. However, the latter may not result in achieving the least possible cost for each client, incentivizing the clients to change their resource allocation. Thus, we look at a variant of CRAGs called Stackelberg CRAGs (SCRAGs) in which we consider how the cloud provider can ensure that the system reaches a global optimum.

We determine various conditions to achieve a Nash equilibrium in a CRAG. Such an equilibrium represents a stable resource allocation to which a CRAG will converge. In a Nash equilibrium, a client cannot increase its utility (i.e., decrease its cost) by unilaterally changing its resource allocation. We show that if linear cost functions are used then the cost to the system at Nash equilibrium is at most a constant factor over the optimal. This is the *Price of Anarchy* [16] in clouds. However, in the general case, the cost of the system at Nash equilibrium can be arbitrarily worse compared to the optimal. Thus, we further investigate the Stackelberg equilibrium achievable in a CRAG which ensures that the cost of the system at equilibrium is atmost a constant times the optimal.

In summary, the main contributions of our paper are:

(a) CRAGs, which capture the conflict of interests between clients of a cloud,
(b) SCRAGs, which capture the client-provider interactions,
(c) Conditions for equilibrium and Price of Anarchy results in both CRAGs and SCRAGs and
(d) Algorithms to ensure close to optimal performance in both CRAGs and SCRAGs.

II. Motivation

In this section, we provide results of an experiment that corroborate our main hypothesis: existing resource allocation policies can cause a client using a heavily loaded cloud to incur unnecessarily higher costs as compared to when the cloud is lightly loaded. The results in this section show the interactions between multiple clients and the suboptimal performance seen by them when simultaneously using the cloud.

We demonstrate the effect of resource contention in a cloud environment by performing various experiments on the Illinois Cloud Computing Testbed (CCT) [2] using Hadoop. We run computationally intensive Hadoop Sort jobs [6] using data generated by Hadoop's RandomWriter [5]. We perform three types of experiments using different input data sizes (50GB, 100GB, and 200GB): (a) *No Contention*: In this experiment, a single Sort job runs without any contention from other jobs, (b) *Concurrent Jobs*: In this experiment, three Sort jobs run concurrently and we measure the effect of contention from two of them on the third and (c) *File transfer*: In this experiment, a single Sort job runs simultaneously with a HTTP file transfer for a 3.7GB remote file. This helps us understand the effect of contention in the network.

Figure 1 shows the time for completion of the Sort job in each of the three scenarios discussed above, averaged over five runs. It allows us to make two important observations. First, the effect of network contention (File transfer) is low for small input size; however, it causes a significant increase in the completion time (by 25%) of the Sort job when 200GB of input data is used. The impact of job contention on the other hand (3 identical concurrent jobs) is much more severe; it causes a 136% and a 173% increase in the turnaround time of a single Sort job for 50GB and 200GB of input data respectively. These results show that the allocation of resources to various jobs can have a profound effect on the completion time of a single job. Thus resource allocation is an important issue in the cloud.

Second, with a per-CPU hour based Amazon EC2's pricing model, a client running a Sort job in parallel with two others would be charged around 136% to 173% more (depending upon the input size) as opposed to when it has exclusive access to the cloud. This shows that pricing of cloud resources is an important issue which needs to be addressed. In this paper, we address the problem of resource allocation via pricing models using game theory.

III. Cloud Resource Allocation Games

In this section, we define and present Cloud Resource Allocation Games (CRAGs). CRAGs model the resource allocation in a cloud and capture the provider-client and client-client interactions. We show that a Nash equilibrium always exists for CRAGs and derive Price of Anarchy results for it by reducing an instance of CRAG to an instance of Selfish Routing in a network [21]. Next, we introduce the concept of Stackelberg CRAGs and propose mechanisms which ensure that the cost incurred by the system at Stackelberg equilibrium is close to the global optimum. In this paper, we only consider a static scenario in which a set of clients submit their jobs to a cloud as a batch. We leave modeling of the dynamics that can exist in practice to future work.

A. Modeling resource allocation in clouds

In our model, the clients of a cloud are modeled as the players in a CRAG. A client's strategy is represented by the client's resource allocation. We make the following *assumptions* about the various entities involved in a cloud:

(a) The cloud hosts only one type of resource (e.g., CPU) and all clients are concerned with the usage of this type of resource only. The more general case with multiple types of resources is an extension of the model described

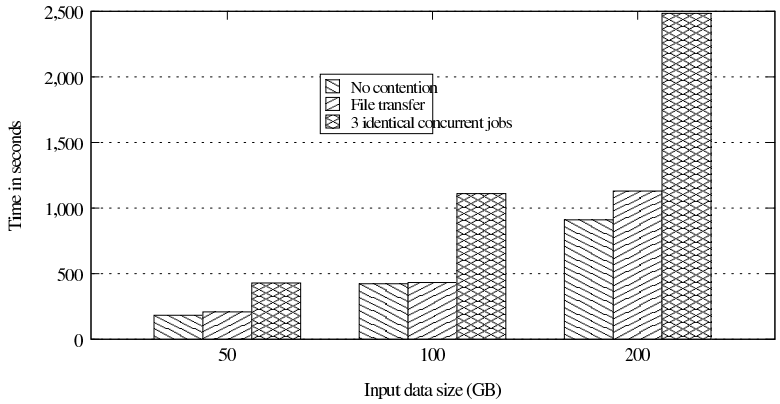


Fig. 1. *Motivation: Average (over five runs) job turnaround time of a Hadoop Sort job under various situations*

below if there are no dependencies between different types of resources. Inter resource-type dependencies (e.g., a Hadoop job might be required to be scheduled on a machine close to its data) are outside the scope of this paper.

(b) Each client acts selfishly and the cloud provider tries to optimize the total utility of the clients while ensuring efficient resource usage. Although cloud providers would be interested in maximizing their revenue, they would not want to charge their clients unnecessarily high prices in order to retain them. This follows from the fact that cloud computing is a competitive market with a choice of multiple providers for clients.

(c) Clients are charged based on per CPU-hour consumed. This implies that the amount of money a client has to pay the cloud is directly proportional to the amount of time the requested amount of resources are used by the client. This of course assumes a fixed number of CPUs requested by a client at a time.

(d) The cloud has sufficient amount of resources to accommodate the resources requested by all the clients.

Suppose a cloud provider C consists of a set $\mathcal{M} = \{1, 2, \ldots, m\}$ of m physical resources (called *machines* from here on). Each machine $i \in \mathcal{M}$ is associated with a cost function $l_i(\cdot)$ ($l_i : \mathbb{R}_+ \rightarrow \mathbb{R}_+$) and a capacity c_i. c_i represents the amount of resource available at machine i. $l_i(x)$ is the cost of using x resources at machine i. We assume that l_i is a monotonic non-decreasing function such that $x \cdot l_i(x)$ is weakly convex for every i. The set $\mathcal{L} = \{l_1, l_2, \ldots, l_m\}$ represents the cost functions of all the machines in the cloud.

We use the function l_i as an abstraction for the cost incurred by the clients who are using machine i. In practice, the cost function associated with a machine represents the amount of money the clients need to pay to use the machine for a particular amount of time. It captures the interactions between the jobs of the various clients using machine i. Since the amount of money paid by a client for a job is directly proportional to the time taken to finish the job, l_i also models the turnaround time seen by the client.

We further assume that a set $\mathcal{U} = \{U_1, U_2, \ldots, U_n\}$ of n clients simultaneously use the cloud C, with client U_j requesting a total of a_j resources from C. a_j represents the total amount of resource-hours that are requested by client U_j. For example, a_j can be the number of CPU-hours requested by U_j. The vector $\mathcal{A} = (a_1, a_2, \ldots, a_n)$ represents the demands of all the clients. A global resource allocation vector Φ defines the resources allocated by the cloud to the clients at various machines. Φ is given by a n-dimensional vector $(\phi_1, \phi_2, \ldots, \phi_n)$ where each ϕ_j maps client U_j's resource requirements to the m machines available. ϕ_j is the *strategy* adopted by client U_j in the CRAG and is given by the vector $(r_j(1), r_j(2), \ldots, r_j(m))$ for $j \in 1 \ldots n$. $r_j(k) \in \mathbb{R}_+$ indicates the resources allocated by Φ to client U_j at machine k. The allocation vector Φ is *feasible* if it satisfies the following properties: (1) $\sum_{k=1}^{m} r_j(k) = a_j$, (2) $r_j(k) \geq 0 \ \forall j, k$ and (3) the total resource utilization R_k at machine k is within its capacity c_k i.e. $R_k = \sum_{i=1}^{n} r_i(k) \leq c_k$.

The 3-tuple $(\mathcal{M}, \mathcal{L}, \mathcal{A})$ defines a *Cloud Resource Allocation Game* (CRAG) where player (client) U_j's strategy is given by its resource allocation ϕ_j. Client U_j wishes to minimize its *cost* which is given by

$$L_\Phi(j) = \sum_{i=1}^{m} \psi(i, j) \cdot l_i(R_i)$$

where $\psi(i, j) = 1$, if machine i is used by U_j in the allocation Φ and $\psi(i, j) = 0$, otherwise. Thus, the cost incurred by a client is the sum of the costs incurred by it at all the machines that are used by it. U_j tries to maximize its *utility* which is given by $-L_\Phi(j)$. The performance of the system is measured by the overall cost C_Φ incurred by the system due to the allocation Φ i.e.

$$C_\Phi = \sum_{j=1}^{n} \sum_{i=1}^{m} r_j(i) \cdot l_i(R_i)$$

The cloud provider is concerned about minimizing the cost C_Φ of the system which maximizes the total resource utilization of the cloud. C_Φ is a measure of the total amount of money all the clients pay to the cloud.

Since cloud providers typically have many available machines, we do not focus on feasibility of schedules in this

paper. Rather, given a sufficiently large C, we focus on the game theoretic issues.

B. Achieving Nash Equilibria in CRAGs

The formulation considered in Section III-A provides the basic model for CRAGs, an *instance* of which is described by the 3-tuple $(\mathcal{M}, \mathcal{L}, \mathcal{A})$. Client U_i selfishly tries to modify its strategy ϕ_i, in order to decrease its cost $L_\Phi(i)$. However, this could increase the cost $L_\Phi(j)$ of another client U_j. Here-in lies the inherent conflict between the requirements of different clients: each client tries to minimize the incurred cost and changes its resource allocation (strategy) until it is unable to decrease its cost further. Such a dynamism would persist unless the system reaches a state in which no client can strictly decrease its cost further. Such a state represents a *Nash equilibrium* [17] and it can be formally defined for our model as follows:

Definition 1: A feasible global resource allocation $\Phi = (\phi_1, \phi_2, \ldots, \phi_n)$ is said to be at *Nash equilibrium* (and called a *Nash assignment*) if $\forall U_i \in \mathcal{U}, L_\Phi(i) \leq L_{\Phi'}(i)$, for any feasible allocation $\Phi' = (\phi'_1, \phi'_2, \ldots, \phi'_n)$ with $\phi'_j = \phi_j$ for $j \neq i$ and $\phi'_i = (r'_i(1), r'_i(2), \ldots, r'_i(m))$ such that $\sum_{k=1}^{m} r_i(k) = \sum_{k=1}^{m} r'_i(k) = a_i$. In other words, a resource allocation is at Nash equilibrium if no client can decrease its cost by unilaterally changing its resource allocation, i.e., no client has any incentive to change its current strategy.

Sub-optimality of Nash: While a Nash equilibrium represents a stable state of the system, the cost C_Φ incurred by the system due to a Nash assignment Φ need not be globally optimal [9]. Thus, in order to investigate the optimal performance that can be achieved in a CRAG, we formulate the following cost optimization problem:

$$\text{Minimize } C_\Phi = \sum_{i=1}^{n} \sum_{k=1}^{m} r_i(k) \cdot l_k(R_k) \tag{1}$$

with the following constraints:

$$\sum_{k=1}^{m} r_i(k) = a_i$$

$$R_k = \sum_{i=1}^{n} r_i(k) \leq c_i$$

$$r_i(k) \geq 0 \ \forall \ 1 \leq i \leq n \ , \ 1 \leq k \leq m$$

Equation 1 can be rewritten as

$$C_\Phi = \sum_{k=1}^{m} R_k l_k(R_k)$$

If $x \cdot l_k(x)$ is convex for all k, the above non-linear optimization problem is a convex program for which the local optimal coincides with the global optimal [19]. Denoting $x \cdot l_k(x)$ by $C_k(x)$ we have $C_\Phi = \sum_{k=1}^{m} C_k(R_k)$. Using this representation of C_Φ and the above constraints, this optimization formulation becomes similar to the *NLP* formulation for Selfish Routing [21] with the following change in terminology: (a) "client" in a CRAG is to be replaced with "path" in an instance of Selfish routing and (b) "physical resource" in a CRAG is to be replaced with "edge" in an instance of Selfish routing. Further, the concept of *flow* in Selfish routing [21] is equivalent to that of global resource allocation in CRAGs.

Thus, the equivalence of CRAGs and Selfish routing [21] can be derived from the formulation of the optimization problems in each of them. Using this equivalence, the following results can be shown to be true (the actual proofs parallel those in [21]):

Theorem 1: The CRAG $(\mathcal{M}, \mathcal{L}, \mathcal{A})$ with continuous non-decreasing cost functions admits a feasible Nash assignment. Further, a Nash assignment of $(\mathcal{M}, \mathcal{L}, \mathcal{A})$ is the solution of the following optimization problem:

$$\text{Minimize } C_\Phi = \sum_{k=1}^{m} l'_k(R_k) \tag{2}$$

with the following constraints:

$$\sum_{k=1}^{m} r_i(k) = a_i$$

$$R_k = \sum_{i=1}^{n} r_i(k) \leq c_i$$

$$r_i(k) \geq 0 \ \forall \ 1 \leq i \leq n \ , \ 1 \leq k \leq m$$

where $l'_i(x) = \int_0^x l_i(t)dt$.

Theorem 2: If the CRAG $(\mathcal{M}, \mathcal{L}, \mathcal{A})$ has linear cost functions then any Nash assignment would incur a cost that is at most $4/3$ times that of the optimal assignment for it.

Theorem 1 shows that the Nash equilibrium for any CRAG satisfying our assumptions exists, is achievable and is same as the solution of the optimization problem given by Equation 2. Theorem 2 gives the Price of Anarchy result in clouds in which all cost functions are linear.

C. Stackelberg Equilibria in CRAGs

To overcome the sub-optimality of the Nash equilibrium for arbitrary cost functions, we consider the Stackelberg variant of CRAGs. Here the cloud provider imposes restrictions such that the cost of the resulting Nash assignment is close to the optimal. In this section, we first formulate a formal definition for a Stackelberg equilibrium in a CRAG and then provide two strategies, *Aloof* and *Least Cost First*, that attempt to ensure that the cost of the equilibrium reached in the Stackelberg game is close to the optimal.

Stackelberg games are a special category of games consisting of two types of players: the *leader*, who tries to optimize the system performance and the *followers* who are selfish players, trying to optimize their own utility. While the leader has no direct control on the followers' strategies, they are constrained by the strategy adopted by the leader. Thus the leader can use his strategy to drive the system to a near-optimal point of operation. CRAGs can be modified as Stackelberg games in which the *cloud provider acts as the leader* trying to optimize performance of the system and *the clients act as followers*, selfishly trying to decrease their individual costs.

A Stackelberg CRAG (SCRAG) is characterized by a 4-tuple $(\mathcal{M}, \mathcal{L}, \mathcal{A}, \phi_L)$, where $(\mathcal{M}, \mathcal{L}, \mathcal{A})$ represents a CRAG (as defined in Section III-B) and $\phi_L = (r_L(1), r_L(2), \ldots, r_L(m)$ denotes the resource allocation of the cloud provider i.e. the strategy of the leader of the SCRAG. $r_L(k)$ denotes the amount of resources used by the leader at machine k. The total resource usage of the leader is given by $a_L = \sum_{k=1}^{m} r_L(k)$. The Stackelberg global resource allocation Φ^S is given by the vector $(\phi_1, \phi_2, \ldots, \phi_n, \phi_L)$ where ϕ_i for $i = 1 \ldots n$ is as defined earlier. The total resource utilization at machine k is given by $R_k^S = r_L(k) + \sum_{i=1}^{n} r_i(k) \leq c_k$. Thus, each client sees that the resources at machine k are decreased by $r_L(k)$, which affects the equilibrium reached by them. The provider's resource usage a_L can be due to any set of jobs that are under its control. In practice, cloud providers themselves need to run various jobs on the cloud infrastructure that are responsible for various tasks e.g, client job scheduling, monitoring, infrastructure management protocols etc. Such jobs could account for the resources a_L of the leader allocated by ϕ_L.

Similar to the Nash equilibrium in simple non-cooperative games, Stackelberg games also reach an equilibrium in which, for a fixed strategy of the leader, no player can strictly decrease its cost by altering its strategy alone. This represents a stable state in which there is no incentive for any client to change its resource allocation. We formally define the Stackelberg equilibrium achieved in a SCRAG as follows:

Definition 2: For the SCRAG $(\mathcal{M}, \mathcal{L}, \mathcal{A}, \phi_L)$, the Stackelberg global resource allocation $\Phi^S = (\phi_1, \phi_2, \ldots, \phi_n, \phi_L)$ is said to be at *Stackelberg Equilibrium* (and called a *Stackelberg assignment*) if $\Phi = (\phi_1, \phi_2, \ldots, \phi_n)$ is a Nash assignment for the CRAG $(\mathcal{M}, \mathcal{L}^N, \mathcal{A})$ where $\mathcal{L}^N = \{l_k^N \mid l_k^N(x) = l_k(x + r_L(k)), k = 1, 2, \ldots, m\}$.

Definition 2 shows the relation between a SCRAG and a CRAG with a modified set of cost functions that depend on the resources used by the leader. A Stackelberg assignment is *optimal* if the cost function C_{Φ^s} induced by it is the global minimum.

The optimality of the Stackelberg equilibrium depends on the leader's strategy and thus is important in determining the performance of the system. We consider two leader strategies for SCRAGs: *Aloof* and *Largest Cost First* (LCF). These strategies are inspired by [20].

Aloof strategy: In the Aloof strategy, the cloud provider is indifferent to the strategies of the clients and tries to find ϕ_L such that it is the optimal assignment for the CRAG $(\mathcal{M}, \mathcal{L}, \mathcal{A}_{\mathcal{L}})$, where $\mathcal{M}$ and $\mathcal{L}$ are as defined earlier and $\mathcal{A}_{\mathcal{L}} = \{a_L\}$. A leader following the aloof strategy does not take into account the strategies adopted by the followers and decides upon its strategy independent of them.

Least Cost First strategy: On the other hand, in the LCF strategy the leader uses the resources that other clients tend to avoid, i.e., the leader uses resources that incur higher costs. The LCF strategy is obtained using the following algorithm:

1) The leader first computes the optimal assignment Φ^* for the CRAG $(\mathcal{M}, \mathcal{L}, \mathcal{A}^*)$, where ($\mathcal{M}$ and $\mathcal{L}$ are as defined earlier and $\mathcal{A}^* = (a_1, a_2, \ldots, a_n, a_L)$. Note that such an assignment need not be at equilibrium for the clients of the cloud i.e., some client U_i could potentially decrease its cost by changing ϕ_i^*. Φ^* can be calculated by solving the convex program given by Equation 1 (Section III-B).
2) Order the resources such that $l_1(\Phi^*(1)) \leq l_2(\Phi^*(2)) \leq \ldots \leq l_m(\Phi^*(m))$ and find the minimum k^* such that $k^* \leq m$ and $\sum_{k=k^*}^{m} \Phi^*(k) \leq a_L$. $\Phi^*(k)$ is the total amount of resources allocated by Φ^* at machine k.
3) The strategy of the leader $\phi_{S_L}^* = (r_L(1), r_L(2), \ldots, r_L(m))$ is given by $r_L(k) = \Phi^*(k)$ for $k > k^*, r_L(k^*) = a_L - \sum_{k=k^*}^{m} \Phi^*(k)$ and $r_L(k) = 0$ for $k < k^*$.

Thus, in the LCF strategy the leader occupies the $(m - k^*)$ most expensive machines and then allows the followers to converge to a Nash Equilibrium. The following theorem shows that the cost incurred by adopting the LCF strategy is at most $\frac{1}{\alpha}$ times worse compared to the optimal assignment for the CRAG under consideration, where α indicates the fraction of jobs that belong to the leader. The proof of this theorem parallels that in [20].

Theorem 3: If ϕ_L^S is the LCF strategy for the leader in the SCRAG $(\mathcal{M}, \mathcal{L}, \mathcal{A}, \phi_L)$ inducing a Nash assignment Φ_F^S for the followers and Φ^* is the optimal assignment for the CRAG $(\mathcal{M}, \mathcal{L}, \mathcal{A})$, then $C_{(\Phi_F^S, \Phi_L^S)} \leq \frac{1}{\alpha} C_{\Phi^*}$ where $\alpha = \frac{a_L}{\sum_{i=1}^{n} a_i}$.

D. CRAGs in Practice

In the earlier subsection we have shown how the resource allocation in clouds can be modeled using CRAGs. We have derived several results which determine the relationship of the Nash and Stackelberg equilibria achievable in a CRAG with the optimal solution. In this section, we show how these mechanisms can be adopted in practice.

The resource allocation based on the equilibria takes place in rounds as the model given in Section III-A is applicable only to a static scenario. At the start of each round, all the clients submit the jobs that have to be run in that round, to the cloud. The actual set of clients can vary across rounds. If the Stackelberg mechanism is used, the provider first calculates his resource allocation vector ϕ_L using either the Aloof or LCF strategy of Section III-C. The provider then advertises the amount of resources left at each of the machines in the cloud to the clients. Each client then chooses the machines that would satisfy its resource requirement and minimize its total cost. Next, the clients will actively change their resource allocation as long as they can decrease their costs. They can follow any of the standard update mechanisms in the literature [16] to determine how they change their strategy. All the clients must follow the same update mechanism.

When no client can decrease its cost by changing its strategy alone, the system has achieved a stable equilibrium. Since such a equilibrium state exists (Theorem 1 in Section III-B), it will be reached with the convergence time depending on the type of update mechanism used by the clients [16]. This equilibrium state corresponds to one of the equilibria (Nash or Stackelberg depending upon the mechanism adopted) defined earlier. Such a state satisfies all the conditions outlined earlier

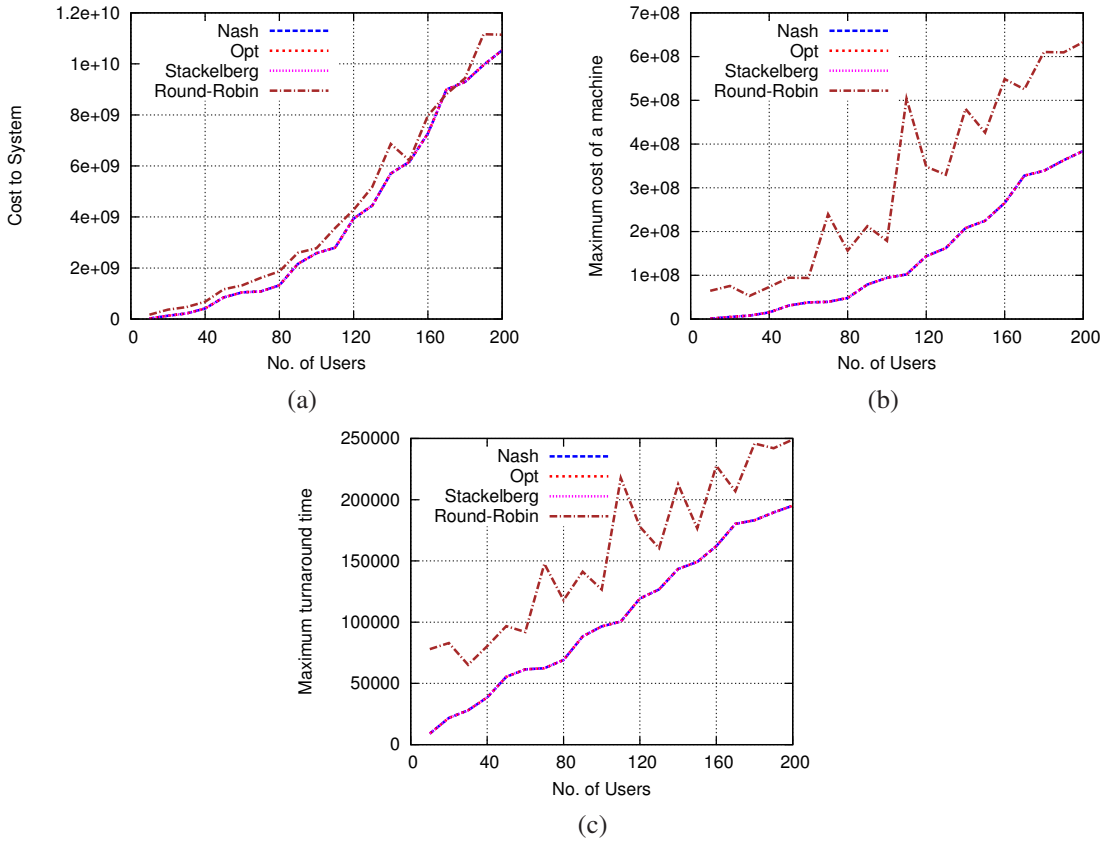


Fig. 2. *Results of simulations with linear cost functions and synthetic workloads: (a) Total Cost to the System, (b) Maximum cost incurred by a client for using a machine and (c) Maximum turnaround time experienced by a client. The lines corresponding to Nash, Stackelberg and Optimal are quite close to each other.*

and the results regarding the bounds on the cost to the system at equilibrium hold.

IV. Experimental Evaluation

In this section, we present experimental results from trace-driven simulations performed to verify how the various equilibria introduced in Section III perform in practice and how they compare to non-game theoretic resource allocation schemes. We use both synthetic and real-world traces for client resource requirements in these simulations. While the former provide us the flexibility to verify how the equilibria of Section III perform in a variety of scenarios, the latter allows us to evaluate them in more practical scenarios.

Metrics: We use the following metrics to determine the efficiency of resource allocation mechanisms:

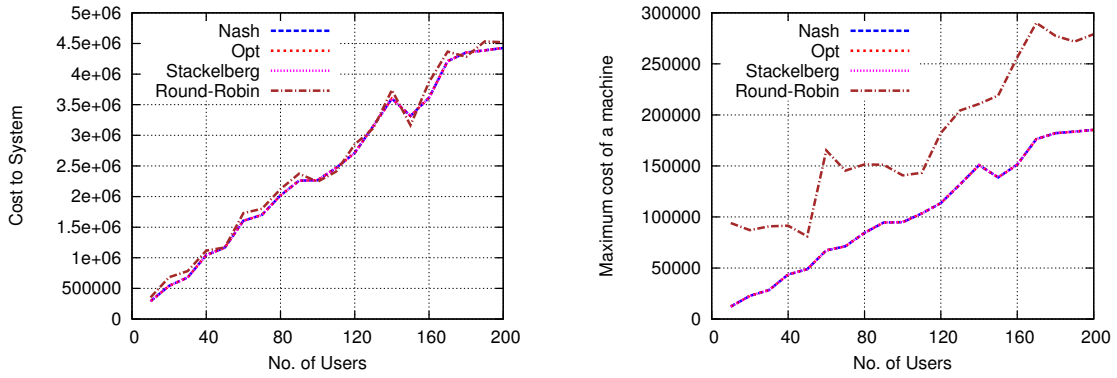
(a) total cost incurred by the system,
(b) maximum cost of a machine: this measures the maximum cost that can be incurred by any client when it uses any machine in the cloud and
(c) maximum turnaround time experienced by any client of the cloud: this is the time between when the client submits a job to when it is completed.

The *cost* incurred by a client is the total amount of money paid by the client to the cloud. The total cost incurred by the system is the total amount of money all the clients pay to the cloud provider. While (a) and (b) help us measure the amount of money clients pay to the cloud, (c) deals with the performance obtained by the clients. These metrics measure how efficiently a cloud is used and the performance experienced by the clients of the cloud. In the ideal case, the total cost incurred by the cloud should be equal to the optimal and the maximum cost incurred by a client should be equal to the average cost incurred by any client. Using trace-driven simulations, we measure how various resource allocation strategies affect each of these metrics.

Trace details: We use two different real-world traces which represent two types of workloads that are typically encountered on clouds:

1) the PlanetLab traces [18] which contain long running network intensive jobs and
2) the Parallel Workloads Archives [4] which typically contain batch jobs that are relatively short-lived.

The PlanetLab trace we use is collected from a total of 69 physical PlanetLab machines. These traces provide us the details of the slices requested by different users using the machines which we use for the values of a_j's in our model. From the Parallel Workload Archives, we use the reduced LLNL-ATLAS-2006, LLNL-uBGL-2006, LLNL-Thunder-2007 and



(a) (b)

Fig. 3. *Results of simulations with exponential cost functions and synthetic workloads: (a) Total cost to the system, (b) Maximum cost experienced by a client. The lines corresponding to Nash, Stackelberg and Optimal are quite close to each other.*

LCG-2005 traces. From these, we use the number of processors and time requested by various users.

The synthetic traces in our experiments were generated as follows: the resource requirements of a client U_i are drawn uniformly at random between 0 and 1000 and *normalized* (divided by) to the total amount of resources available in the cloud in order to ensure that the cloud contains sufficient amount of resources for all the clients. The total number of physical machines in the cloud for these workloads is fixed at 50. We vary the number of clients using the cloud and measure the change in the various metrics we are concerned with. This in turn measures the effect of variation of load on the performance of the equilibrium based mechanisms in the cloud.

Evaluation Approach: We compare the performance of the Nash, Optimal and Stackelberg equilibria with a non-game theoretic method, "Round-Robin", that has been believed to work well in systems like Hadoop. In the Round-Robin method, the n machines in the cloud are arbitrarily numbered from $1 \ldots n$. When a client schedules a job which requests a resources at k machines each, it is scheduled on the next k machines, chosen in a Round-Robin manner, that have the required amount of resources left. If the required number of machines is unavailable, the job is paused until the required resources are freed.

We perform all the simulations using a trace-driven simulator that solves the convex optimization problems formulated in Section III using the Linear approximation method, an iterative method for solving convex programs. The Nash equilibrium is found by solving the optimization problem (Equation 2) formulated in Theorem 1. The LCF strategy (with $\alpha = 0.01$) described in Section III-C is used to find the Stackelberg equilibrium.

We use two types of functions to model the costs associated with the machines of the cloud:

1) linear cost functions, represented as $l_i(x) = c_i \cdot x$ and
2) exponential cost functions, represented as $l_i(x) = c_i \cdot e^{(k_i x)}$.

These functions model the time for completion of a job using the corresponding machines and depend on the total resource consumption at the machine. For example, the job completion time for a client using a machine i with 50% resource usage is proportional to $l_i(50)$. The constants c_i in these functions are proportional to the money the client has to pay for using the corresponding resources in the cloud.

Microbenchmarks: Figure 2 compares the performance of the various equilibria with that of the Round-Robin method in terms of the various metrics of interest by varying the number of clients. We use linear cost functions for these simulations along with synthetic traces. While the total costs incurred by the system at the equilibria closely overlap those at the optimal (Figure 2(a)), the Round-Robin procedure performs about 15% worse on average. The difference is more evident when the other two metrics are taken into account. The maximum cost incurred by using a machine (Figure 2(b)) in the Round-Robin method is about 5 times more than that incurred while using the equilibrium mechanisms. Similarly, the maximum turnaround time experienced by a client in the Round-Robin method (Figure 2(c)) is about twice compared to that seen in the case of the equilibrium based mechanisms. Thus, we can conclude that existing schemes like Round-Robin result in low performance compared to resource allocation based on CRAGs.

Figure 3 provides a comparison between the performance of the Round-Robin mechanism and the various equilibria for exponential cost functions. This is similar to the case of linear cost functions and the conclusions are similar to those for Figure 2. This shows that the effect of exponential cost functions is similar to that of linear cost functions.

PlanetLab Experiments: The results of the simulations performed with real-world PlanetLab based traces are shown in Figure 4. In these simulations, we use linear cost functions. The results with exponential cost functions are similar and not shown. These results help us understand how the various equilibria discussed in Section III perform in practice and how they compare to existing scheduling strategies like Round-Robin.

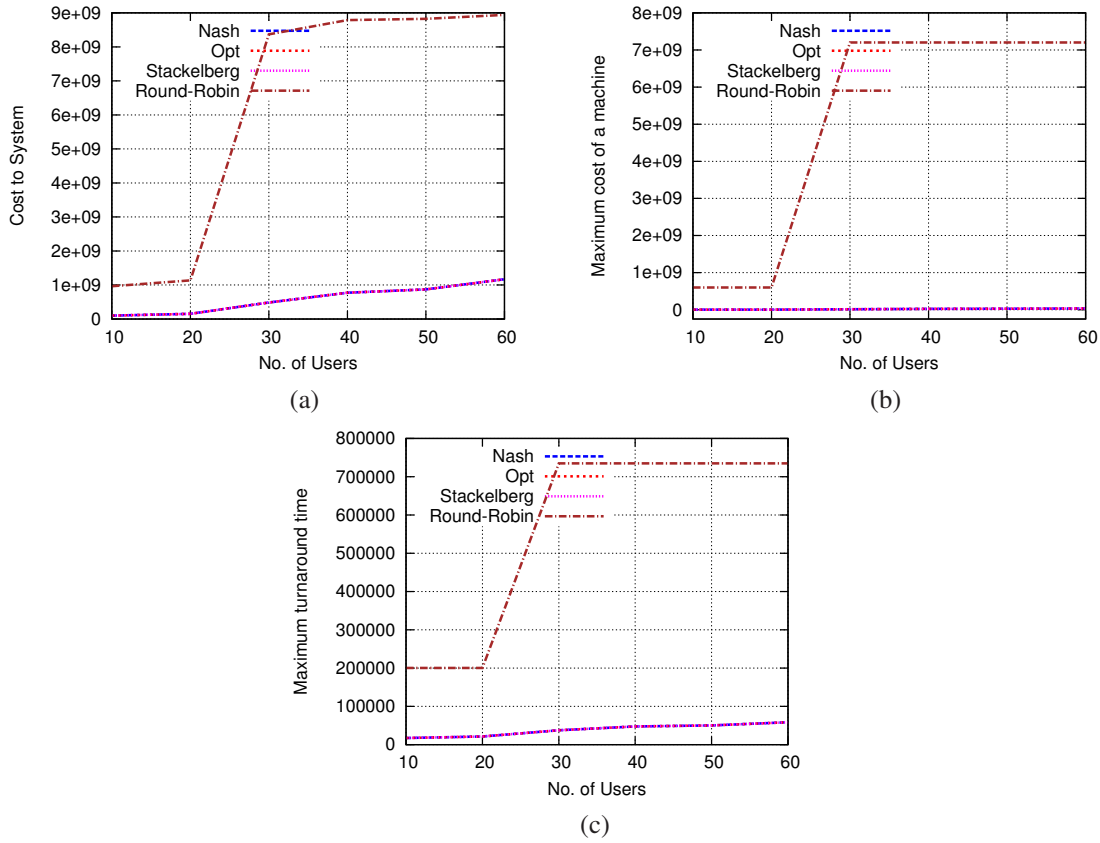


Fig. 4. *Results of simulations with linear cost functions and workloads from PlanetLab: (a) Total cost to the system, (b) Maximum cost incurred by a client for using a machine and (c) Maximum turnaround time experienced by a client. The lines corresponding to Nash, Stackelberg and Optimal are quite close to each other.*

Figure 4(a) compares the total cost incurred by the system while using the Round-Robin mechanism with that while using the various equilibria of Section III. It shows that the Round-Robin mechanism can incur around 6 times more cost than any of the equilibrium mechanisms. Figures 4(b), (c) compare the Round-Robin method with the equilibria in terms of the maximum cost of a machine and maximum turnaround time of a client and show that it can perform much worse. Further, these figures illustrate that the *worst case bounds of the Nash and Stackelberg equilibria shown in Section III do not necessarily arise in practice*. In particular, Figure 4 shows that the cost incurred at the equilibria is quite close to that of the optimal.

Parallel Workload Experiments Next we provide results of the experiments performed using traces from the Parallel Workload archives [4]. These provide a real-world example of batch processing jobs. Figure 5 shows the results of these experiments using four different types of workloads: LCG-2005(LCG), LLNL-ATLAS-2006 (LLNL-AT), LLNL-Thunder-2007 (LLNL-Th) and LLNL-uBGL-2006 (LLNL-uB) from [4]. Figure 5(a) shows the maximum cost that can be incurred by a client relative to the optimal. While the relative cost incurred at the Nash and the Stackelberg equilibria are quite close to 1, the cost incurred by the Round-Robin mechanism is worse by several orders of magnitude compared to the optimal. Figure 5(b) gives the average turnaround time experienced by the clients relative to the optimal. It shows that the Nash and Stackelberg equilibria perform close to the optimal while the Round-Robin is around 5% to 15% greater than the optimal.

Summary: The simulations in this section provide us with the following general conclusions: while Nash and Stackelberg equilibria result in costs close to the optimal, methods like Round-Robin can perform much worse compared to it, sometimes by several orders of magnitude. Performance-based metrics like turnaround time in Round-Robin mechanism can be several times worse (15% to 500% depending upon the scenario) compared to that in the Nash and Stackelberg equilibria. Further, the Nash and Stackelberg equilibria show similar performance and incur costs that do not differ significantly from the optimal for the real-world based workloads.

V. Related Work

In this paper, we consider the use of game theoretic approaches for resource allocation in cloud computing environments. Our goals are related to works in three areas:

Resource allocation in shared-computing environments: Several recent works (e.g., [7], [11]) try to ensure efficient

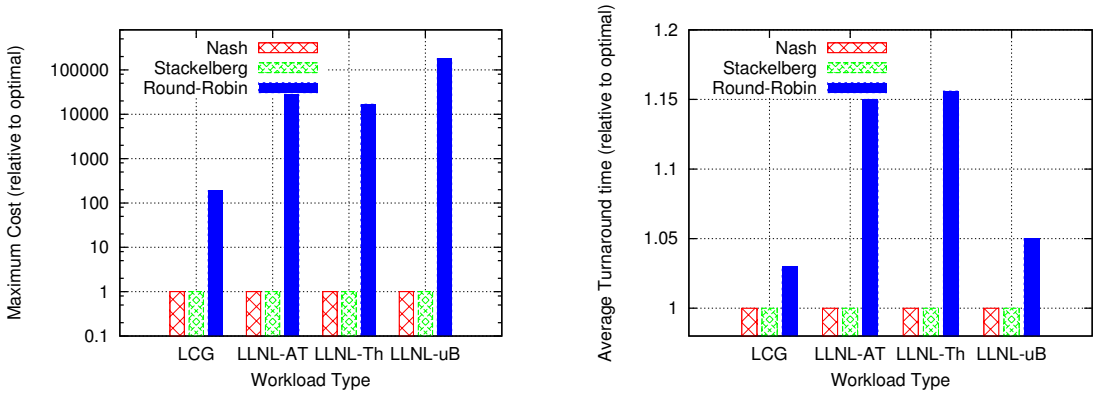


Fig. 5. *Results of simulations with linear cost functions and workloads from Parallel Workload Archives: (a) Maximum cost incurred by a client for using a machine and (b) Average turnaround time experienced by a client.*

usage of shared resources. However, they generally adopt a global optimization approach which does not work [15] for cloud computing in which the resources are made public and multiple clients share the same physical infrastructure. Unlike earlier works, we take a comprehensive approach in this paper: irrespective of the type of resource, our algorithms can be used to ensure optimal allocation of that resource among the clients.

Economic approaches in shared systems: Simple reservation mechanisms try to cater to the clients' needs but do not provide bounds on the system performance compared to optimal solutions. On the other hand, economic approaches can be used to ensure that clients can achieve near optimal system performance. These approaches [10] have been used to model and solve problems related to the supply-demand economics of complex systems where multiple clients and multiple providers exist. Sharp [12], Tycoon [14] and grid markets [22] are examples of such approaches in grid computing where economic approaches have been successfully adopted to deal with efficient resource allocation. More recently, Google introduced an auction-based mechanism [24] for reserving resources on their internal systems.

Game theoretic approaches in practical computing systems: In this paper, we take a game-theoretic approach that has been widely used as a tool to achieve optimal resource usage and to design optimal scheduling mechanisms. Game theory [16] has been used to model several aspects of computing systems including routing, resource allocation and scheduling. Several works related to selfish routing [8], [21] present game-theoretic models for routing traffic in networks in order to optimize the latency experienced by the end users under congestion. We draw upon these works to derive several properties of CRAGs. Further, Stackelberg scheduling has been used in several practical systems [13], [20] to achieve optimal system usage by using leader strategies which ensure that followers achieve a Nash Equilibrium whose cost (nearly) coincides with the optimum. We exploit this property of Stackelberg games and ensure that SCRAGs achieve a close to optimal allocation of resources in the cloud.

Game-theoretic concepts have been applied to cloud computing in [15], which introduces the concept of collocation games. While this is most closely related to our current work, it differs from our model of the cloud in the following ways:

(a) it assumes that there is no interaction between clients, i.e. each client can be allocated resources without affecting the performance of others,
(b) it assumes that the cloud provider's prices for different resources are fixed, and
(c) there is no concept of time; the price each client pays is modeled as a constant fraction, proportional to the amount of resource requested by it but does not depend on the time for which it uses the resource (which implicitly assumes that different clients use all the resources for the same amount of time).

Further, it only finds solutions for simple collocation games (which capture a small subset of practical scenarios) where clients request for a single slice on a single resource. Our work on the other hand takes such issues into account and deals with a more general problem by modeling the cloud closer to reality.

VI. Conclusion and Future Work

In this paper, we argued that fixed pricing and resource allocation mechanisms followed by today's cloud providers can lead to suboptimal performance. We adopted a game-theoretic approach to capture the subtleties of interactions between multiple clients and the cloud provider and modeled them as *Cloud Resource Allocation Games* (CRAGs). We derived conditions for various equilibria in CRAGs, provided worst case bounds (price of anarchy results) and gave algorithms which ensure that the cost to the system is nearly optimal. We further showed using extensive trace-based simulations, that the performance of the cloud using heuristics like Round-Robin can be about 5 times worse as compared to that using resource allocation according to the Nash, Optimal and Stackelberg equilibria. We also showed that, in practice, the costs at the Nash and Stackelberg equilibria are quite close to the optimal.

Our paper opens up several exciting avenues for future work. Although we have taken several aspects of a real cloud computing environment into account, our model can be extended to capture more of the complexities of today's clouds such as:

(a) the dependencies across multiple types of resources in a cloud,
(b) the interactions between multiple cloud providers, and
(c) various privacy and security constraints that a user might require, e.g, Client X may not want to be collocated with Client Y.

We also assumed that resources are infinitely divisible which is not true in reality. Accounting for these factors could affect the optimality of the Nash and Stackelberg Equilibria in CRAGs. Quantifying the effects of such factors is necessary to develop and analyze a comprehensive model of the cloud. Further, investigating how a market of multiple cloud providers can be used by a set of clients efficiently (i.e., minimize the costs incurred by them) is also a part of our future work.

References

[1] "Amazon elastic compute cloud (EC2)," http://aws.amazon.com/ec2/.
[2] "Cloud computing testbed," http://cloud.cs.illinois.edu/.
[3] "Google app engine," http://appengine.google.com/.
[4] "Parallel workload archives," http://www.cs.huji.ac.il/labs/parallel/workload/.
[5] "RandomWriter - Hadoop Wiki," http://wiki.apache.org/hadoop/RandomWriter.
[6] "Sort - Hadoop Wiki," http://wiki.apache.org/hadoop/Sort.
[7] A. Batsakis, J. Lentini, R. Burns, T. Talpey, and A. Kanevsky, "CA-NFS: A congestion-aware network file system," in *FAST*, 2009.
[8] A. Czumaj, P. Krysta, and B. Vong, "Selfish traffic allocation for server farms," in *Proceedings of the 34th Annual Symposium on Theory of Computing (STOC)*, 2002, pp. 287–296.
[9] P. Dubey, "Inefficiency of nash equilibria," *Math. Oper. Res.*, vol. 11, no. 1, pp. 1–8, 1986.
[10] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini, "Economic models for allocating resources in computer systems," in *Market Based Control of Distributed Systems. World Scientific*. Press, 1996, pp. 156–183.
[11] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. New York, NY, USA: ACM, 2009, pp. 261–276.
[12] Y. F. Jeffrey, J. Chase, B. Chun, S. Schwab, and A. Vahdat, "Sharp: An architecture for secure resource peering," in *Proceedings of the 19th ACM Symposium on Operating System Principles*, pp. 133–148.
[13] Y. A. Korilis, A. A. Lazar, and A. Orda, "Achieving network optima using stackelberg routing strategies," 1997.
[14] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman, "Tycoon: An implementation of a distributed, market-based resource allocation system," *Multiagent Grid Syst.*, vol. 1, no. 3, pp. 169–182, 2005.
[15] J. Londono, A. Bestavros, and S.-H. Teng, "Collocation games and their application to distributed resource management," in *HotCloud*, 2009.
[16] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. Cambridge University Press, 2007.
[17] G. Owen, *Game Theory*, 3rd ed. Academic Press, 1995.
[18] K. Park and V. S. Pai, "Comon: a mostly-scalable monitoring system for planetlab," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 1, pp. 65–74, 2006.
[19] A. L. Peressini, F. E. Sullivan, and J. J. Jerry Uhl, *The Mathematics of Nonlinear Programming*. Springer-Verlag, 1988.
[20] T. Roughgarden, "Stackelberg scheduling strategies," in *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, 2001, pp. 104–113.
[21] T. Roughgarden and E. Tardos, "How bad is selfish routing?" *J. ACM*, vol. 49, no. 2, pp. 236–259, 2002.
[22] T. Sandholm, J. A. O. J. Odeberg, and K. Lai, "Market-based resource allocation using price prediction in a high performance computing grid for scientific applications," in *Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing*. IEEE, 2006, pp. 132–143.
[23] J. Shneidman, C. Ng, D. C. Parkes, A. Auyoung, A. C. Snoeren, A. Vahdat, and B. Chun, "Why markets could (but don't currently) solve resource allocation problems in systems," in *Proceedings of the 10th USENIX Workshop on Hot Topics in Operating Systems*, 2005, p. 7.
[24] M. Stokely, J. Winget, E. Keyes, C. Grimes, and B. Yolken, "Using a market economy to provision compute resources across planet-wide clusters," in *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–8.