

COLLOCATION GAMES

And Their Application to Distributed Resource Management

Jorge Londoño[†] Azer Bestavros[‡] Shang-Hua Teng[♭]
jmlon@cs.bu.edu best@cs.bu.edu steng@cs.bu.edu

Computer Science Department
Boston University
BUCS-TR-2009-002

Abstract

We introduce Collocation Games as the basis of a general framework for modeling, analyzing, and facilitating the interactions between the various stakeholders in distributed systems in general, and in cloud computing environments in particular. Cloud computing enables fixed-capacity (processing, communication, and storage) resources to be offered by infrastructure providers as commodities for sale at a fixed cost in an open marketplace to independent, rational parties (players) interested in setting up their own applications over the Internet. Virtualization technologies enable the partitioning of such fixed-capacity resources so as to allow each player to dynamically acquire appropriate fractions of the resources for unencumbered use. In such a paradigm, the resource management problem reduces to that of partitioning the entire set of applications (players) into subsets, each of which is assigned to fixed-capacity cloud resources. If the infrastructure and the various applications are under a single administrative domain, this partitioning reduces to an optimization problem whose objective is to minimize the overall deployment cost. In a marketplace, in which the infrastructure provider is interested in maximizing its own profit, and in which each player is interested in minimizing its own cost, it should be evident that a global optimization is precisely the wrong framework. Rather, in this paper we use a game-theoretic framework in which the assignment of players to fixed-capacity resources is the outcome of a strategic "Collocation Game". Although we show that determining the existence of an equilibrium for collocation games in general is NP-hard, we present a number of simplified, practically-motivated variants of the collocation game for which we establish convergence to a Nash Equilibrium, and for which we derive convergence and price of anarchy bounds. In addition to these analytical results, we present an experimental evaluation of implementations of some of these variants for cloud infrastructures consisting of a collection of multidimensional resources of homogeneous or heterogeneous capacities. Experimental results using trace-driven simulations and synthetically generated datasets corroborate our analytical results and also illustrate how collocation games offer a feasible distributed resource management alternative for autonomic/self-organizing systems, in which the adoption of a global optimization approach (centralized or distributed) would be neither practical nor justifiable.

[†] Supported in part by the Universidad Pontificia Bolivariana and COLCIENCIAS–Instituto Colombiano para el Desarrollo de la Ciencia y la Tecnología "Francisco José de Caldas".

[‡] Supported in part by NSF awards CCF-0820138, CSR-0720604, EFRI-0735974, CNS-0524477, and CNS-0520166.

[♭] Supported in part by NSF award CCR-0635102.

1 Introduction

Motivation and Scope: *Cloud computing* has emerged as a compelling paradigm for the deployment of distributed applications and services on the Internet [15, 11, 23].

The advent of cloud computing can be attributed in large to the maturity and wide adoption of virtualization technologies. By relying on virtualized resources, users are able to easily deploy, scale up or down, and migrate their applications seamlessly across computing resources offered by one or more infrastructure providers. More importantly, virtualization enables performance isolation, whereby each application is able to acquire appropriate fractions of shared fixed-capacity resources for unencumbered use, ensuring that the application would meet minimal Quality of Service (QoS) or Service-Level Agreement (SLA) requirements. Enabling the specification and dynamic acquisition of the fraction of a resource that is needed for proper application support is important because it allows multiple applications to be *safely* collocated on a shared cloud resource.

The assignment of applications (each with its own resource requirements) to resource instances is a classical resource management problem – that of partitioning the entire set of applications into subsets, each of which is assigned to fixed-capacity resources. If the computing infrastructure as well as the various applications making use of that infrastructure are all under the same administrative domain, this partitioning reduces to an optimization problem whose objective is to minimize the overall deployment cost. This is not the case in a cloud computing environment.

Cloud computing *decouples* the ownership/operation of computing resources from the usage of such resources, in much the same way the ownership and operation of a power plant is decoupled from the usage of the electricity it produces. Cloud computing *users* leverage resources (servers, networks, and storage) hosted by a *provider* and are billed for what they use based on a preset resource-quantum per unit-time price, like a metered utility. For example, using the Amazon cloud, an EC2 instance (a 64-bit windows processing platform with 7.5GB of RAM and 850GB of storage) costs \$0.5 per hour [3]. Thus, to run a server on such an instance for a month would cost \$360. Notice that this cost is borne by the user whether or not the user consumes the full capacity of the instance. In other words, the pricing of the resource is independent of the utilization of that resource. Clearly, it is possible for multiple users to share the use of a cloud resource as long as their aggregate utilization does not exceed the capacity of the resource. While such collocation would be attractive to users (as it would lower their costs) it is not attractive to the provider (as it would reduce their profits). This suggests that providers have no incentive to “optimize” the assignment of users to resources – indeed, they have the exact opposite incentive. This implies that it is up to each user in the cloud to minimize its own cost through collocation with other users – noting that a resulting assignment of users to resources that is best for one user may not be the best for other users.

To summarize, in a marketplace, in which the infrastructure provider is interested in maximizing its own profit, and in which each (independent and rational) user is interested in minimizing its own cost, viewing cloud resource management as a global optimization is precisely the wrong way to approach the problem. Rather, given a user’s ability (afforded by virtualization) to unilaterally migrate from one resource to the other in order to reduce its own cost as a result of better collocation with other users, cloud resource allocation and acquisition is better viewed through a game theoretic perspective. To that end, in this paper we introduce *Collocation Games* as the basis of a general framework for modeling, analyzing, and facilitating the interactions between the various stakeholders in a cloud computing environment.

Related work: Economic models have been used in prior grid resource management frameworks (*e.g.*, [8, 25, 16, 6]) in which commodity markets, auctions, double-auctions, and combinatorial auctions are used for resource allocation and scheduling. VCG mechanisms have the advantage of optimizing the social value and of being strategy-proof, at the expense of being computationally hard. A common element across all of these frameworks is the assumption that there is an authority controlling the market and that all users trust it, and abide by its decisions. An alternative perspective to avoid the need for this central authority is to frame the problem as a game in which player actions are guided by their own selfish and rational goals, but in such a way that the incentives lead to an equilibrium not too far from a socially-desirable outcome.

Algorithmic game-theory has provided us with many examples ([19, 12, 24, 7]) where indeed it is possible to obtain self-organized systems that achieve an equilibrium within a bound (*Price-of-Anarchy*) of the social optimum. For example, in scheduling games (*e.g.*, [19, 12]) users share a set of parallel links (processors) to transfer a file (execute a task) and the goal of each user is to minimize its individual completion time (or *makespan*), whereas in routing games (*e.g.*, [24, 7]) users share a network with a given topology and link capacities, and the goal of each user is to find the minimum-cost path for its flow. In both of these examples, the cost function is assumed to increase with the number of users sharing the resource, and the resources themselves have *no* capacity constraints (and thus are unable to provide any performance guarantees). Motivated by considerations of emerging cloud, grid, and SoA practices, our setting is different as we assume that users have minimum allocation requirements that need to be reserved exclusively (for QoS or SLA purposes), and thus the capacity of the resource is a constraint that limits feasible assignments.

Cost-sharing games [20, 5] deal specifically with settings in which resource costs are fixed and distributed among users sharing them. In particular, *cooperative cost sharing games* [20] define two models: The Non-Transferable Utility (NTU) model or the Transferable Utility Model (TU). The concept of equilibrium is captured by the notion of the *core* of the game, *i.e.*, a coalition where no player would benefit by breaking away. For TU-games, where the cost of the resources may be arbitrarily assigned, approximate solutions can be found using linear programming. Finding the core for NTU games is exponential in the worst-case. In contrast, our collocation games are pure-strategy games, where the cost is distributed *in proportion* to the allocated shares. So, although the cost is variable, it is determined by the allocation. This is an important differentiator, which is motivated by a real-world notion of fairness in apportioning cost based on resource allocation. On the other hand, Anshelevich et al[5] present a pure-strategies cost sharing scheme applied to network formation games and generalized to allocation of subsets of resources from an global pool or resources. In this case, the cost is equally shared among all players using a single resource, which leads to a well behaved potential function. This model does not generalize to games with more than two players with weighted requirements.

Contributions: We present a game-theoretic model for the interaction of rational, selfish players sharing resources in a distributed environment, where users can easily relocate their tasks subject to QoS constraints. The general model describes the behavior of a wide range of *self-organizing* distributed systems, where all interactions are guided by players' selfish goals. Under this general setting, we show that the existence of a Nash equilibrium is an NP-complete problem. Next, we explore the mechanism design problem of creating a cost function that induces a particular (desirable) user behavior. In particular we explore the goal of maximizing resource utilization so that all users can perform their tasks subject to their QoS guarantees, but minimizing the total (social) cost of the allocated resources. We present a simplified version of the collocation game, called *the Process Collocation Game*, for which we provide analytical bounds on convergence and price of anarchy for (unidimensional) games involving a single type of resources. We also show that our convergence results extend to (multidimensional) games involving multiple types of resources as well. We also present results for PPCG, a variant of PCG that allows users to request an aggregate set of resources (*e.g.*, a server farm, or servers arranged in a particular topology) as opposed to a single resource.

In addition to these analytical findings, we also present empirical results obtained from two sets of experiments. In the first set, we used synthetically-generated workloads to explore the characteristics of the game under a wide range of settings. In the second, we used PlanetLab traces to evaluate the game dynamics under realistic multi-resource scenarios. In addition to corroborating our analytical findings, our experimental results suggest that collocation games could be used as the basis for building distributed, on-line, self-organizing systems, in which the adoption of a global optimization approach (centralized or distributed) would be neither practical nor justifiable.

2 Collocation Games: Definition and Applications

In this section we present the basic definitions underlying collocation games, relating these definitions as much as possible to applications in distributed systems in general, and to our target cloud computing applications in particular.

Model and Notation: We use a labelled graph $G = \langle V, E \rangle$ to model general infrastructure (cloud) resources that are available to a set of users. We refer to this graph as the hosting (or infrastructure) graph. Vertices (or nodes) in G represent standalone resources, whereas edges represent relationships between these resources. Examples of standalone resources include processors and storage (such as Amazon’s EC-2 or S3 instances), both of which would be represented as vertices in G . Examples of relationships between standalone resources include communication links and spatio-temporal adjacencies, both of which would be represented as edges in G . Here we note that edges in G may be directed or undirected. For instance, a pipeline of processing units would be represented using a chain of vertices, in which each pair of adjacent vertices are connected using a directed edge, whereas a cluster of processing units on a LAN would be represented using a fully-connected subgraph, in which each pair of vertices are connected using an undirected edge.

We use a labelled graph T to model the set of cloud resources and underlying relationships that are necessary to support a specific user application or *task*. We refer to this graph as the user requested task graph. Vertices and edges in T have the same meaning as those in the hosting graph G . Labels in G or T may decorate either vertices or edges. In a hosting graph, labels specify *supply* attributes such as unit capacities and unit prices of processing or communication links.¹ In a requested task graph, labels specify *demand* attributes such as the minimum CPU utilization and storage needed by a standalone process, or the minimum bandwidth tolerable by communicating processes, *etc.* in the task. Notice that resources may have *multidimensional capacities*. For example, an Amazon EC-2 instance specifies capacities along three dimensions corresponding to CPU, RAM, and disk. Thus, it is convenient to refer to the supply of (or demand on) a resource with a capacity (utilization) vector.

As illustrated in Figure 1, a set of requested task graphs (one per user) constitutes the overall *workload* to be hosted on the infrastructure graph G . A *mapping* M of the (vertices and edges in the) set of request graphs to the (vertices and edges in the) hosting graph constitutes a configuration underscoring a specific assignment of users to resources. A *valid configuration* is one wherein supply meets demand – for example, the aggregate demand (*e.g.*, CPU utilization) of all users sharing a vertex in G do not exceed the supply (*e.g.*, CPU capacity) of that vertex. Given a valid configuration, the infrastructure provider expects to be paid for any resource in G used by at least one task (user), but not for (idle) resources to which no tasks were mapped. The price charged per resource is fixed, independent of the number of users sharing that resource. The cost incurred by a user is given by a *cost function* which apportions the price of each resource in G among all users with tasks mapped to that resource. The cost function can be seen as the marketplace mechanism that governs and induces symbiotic relationships among rational, selfish agents (the users). Without loss of generality, in this paper, we adopt a specific form of cost functions that may be conceived as fair – namely, those that split the fixed cost of a resource among tasks in some proportional (*e.g.*, linear) fashion based on the utilization of that resource by the various tasks assigned to it.

The General Collocation Game (GCG): Given a hosting graph $G = \langle V, E \rangle$ where each vertex and edge in G is labelled with a *resource capacity vector* (R) and a *price* (P), and given a collection of tasks, each in the form of a graph $T_i = \langle V_i, E_i \rangle$, where each vertex and edge in T_i is labelled with a *weight* underlying a resource utilization vector (W), the *General Collocation Game* is the pure-strategies game, in which each task is able to make a (better response) move whereby, if possible, the task modifies a valid mapping M into another M' so as to minimize its own cost, given by a function

¹ Without loss of generality, we restrict our attention to capacity and pricing, noting that other attributes may well be considered, *e.g.*, the maximum delay on a link or the OS version and supported APIs on a processing unit. Such attributes would be used to determine the feasibility of mapping vertices and edges from T to G .

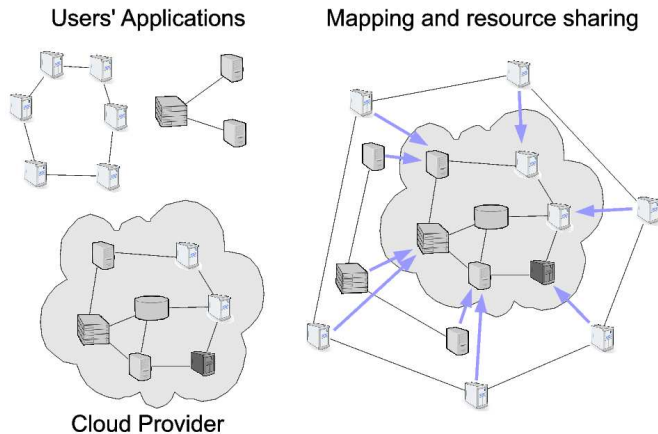


Figure 1: The Collocation Game

$c_M(T_i)$ for the cost of task T_i when hosted in G according to a mapping M :²

$$c_M(T_i) = \sum_{j \in \{V_i, E_i\}} P_j \cdot \left(w_{ij} + (1 - U_j) \frac{w_{ij}}{U_j} \right) = \sum_{j \in \{V_i, E_i\}} P_j \frac{w_{ij}}{U_j} \quad (1)$$

where w_{ij} is the weight (or utilization) imposed on resource j by task T_i , P_j is the price of the resource, and U_j is the total utilization of the resource (by all tasks assigned to the resource by M).

The *social cost* of GCG for a given mapping M is the sum of the costs of all tasks:

$$s = \sum_{\forall T_i} c_M(T_i) \quad (2)$$

The Process Collocation Game (PCG): PCG is a restricted (simpler) version of GCG for which we have concrete analytical results (presented in §3). In a PCG, a task graph consists of a *single vertex* representing an independent process (say a computation) that needs to be assigned to a single resource (say a processor). In a PCG, the cost function for process i when mapped to resource j is

$$c_j(i) = P_j \cdot w_i / U_j \quad (3)$$

where U_j is the overall utilization of resource j , which must satisfy its capacity constraint:

$$U_j = \sum_{i \in j} w_i \leq R_j \quad (4)$$

Assuming all processes (users) are rational and selfish, the only move that a process would make in PCG is one that results in a reduction of its own cost *and* would also benefit other processes with which the process would be collocated as a result of the move. When a process i moves from resource a to resource b , two situations are possible. In the first situation, the move by process i does not result in a displacement of any of the processes on resource b . We call such moves *placement moves*. This situation occurs if resource b has enough capacity to host process i (in addition to all the processes already on it before the move). In this case, it is easy to show that the utilization of resource b increases and that the cost for all processes already on resource b is reduced. In the second situation, the move by process i results in displacing one or more of the processes that were on resource b before the move. We call moves that result in such displacement *replacement moves*. For a replacement move to be possible the cost of all processes that are not replaced by the move

² Throughout the paper, we conveniently refer to processes and resources using their indices, *e.g.*, i instead of T_i .

must be reduced (otherwise, as rational players, such processes would not accept to be collocated with process i on resource b). This will be the case if the move results in a strict increase in the total utilization of b .

Let U'_b refer to the utilization of resource b after a move by process i into it. Clearly, for both placement and replacement moves, $U'_b > U_b$ - i.e., U'_b increases as a result of the move. The only reason for a process i to “move” from resource a to resource b is that $c_b(i) < c_a(i)$. This, along with our observation that $U'_b > U_b$, yields the following conditions for what constitutes a *valid move* by process i into resource b :

$$U'_b > U_b \quad \text{and} \quad \frac{P_b}{U'_b} < \frac{P_a}{U_a} \quad (5)$$

The Multidimensional Process Collocation Game: As we alluded before, a resource (vertex) in the hosting graph may have a multidimensional capacity (recall the CPU, memory, and storage attributes of an EC-2 instance) and is thus represented by a capacity vector $R_j = [r_{j1}, \dots, r_{jd}]$ and a fixed (scalar) price P_j . A natural representation of the demand from such a resource is a vector $X_i = [x_{i1}, \dots, x_{id}]$, whose components represent the task’s demand (or requested utilization) from each dimension of the resource. A multidimensional PCG is a game in which resources (and process utilizations) are multidimensional.

For the purpose of defining a cost function and valid moves for multidimensional PCG, we need to make a few adjustments to our previous definitions. In particular, a set of tasks can be assigned to resource j if the sum vector of demands is component-wise less than the capacity vector:

$$\sum_{i \in j} X_i \leq R_j \quad (6)$$

To apportion the multidimensional resource price among the processes collocated at the resource, we define the multidimensional utilization of a single process by its volume: $v_i = \prod_{k=1}^d x_{ik}$. The total utilization of all processes collocated at resource j is the sum of the multidimensional utilization of these processes: $U_j = \sum_{i \in j} v_i$. This leads us to the following definition for the cost of task i when allocated to resource j (noting that the sum of the cost for all tasks collocated at resource j matches the price set for resource j):

$$c_j(i) = P_j \cdot \frac{v_i}{U_j} \quad (7)$$

In a multidimensional PCG, a *valid move* is one that meets the feasibility constraint (6) and that gives a cost reduction for the task, so when process i moves from a to b , $c_b(i) < c_a(i)$ and $\frac{P_b}{U_b + v_i} < \frac{P_a}{U_a}$. As before, this definition extends to *replacement moves* as well, by assuring that the task that moves and the tasks that remain in the destination reduce their costs

$$U'_b > U_b \quad \text{and} \quad \frac{P_b}{U'_b} < \frac{P_a}{U_a} \quad (8)$$

The Parallel Process Collocation Game (PPCG): We consider the following extension of PCG which allows for tasks to consist of a set of parallel processes. In PPCG, the hosting graph G is a complete graph of n vertices (or nodes), where the i^{th} node has resource capacity R_i and price P_i . In PPCG, a workload consists of a collection of task graphs T_1, \dots, T_m , where task graph T_i consists of a set of k_i nodes, each of which with a specific utilization requirement (weight). Thus T_i is fully specified by parameters $(k_i, w_{i1}, \dots, w_{ik_i})$, where k_i is a positive integer denoting the number of nodes (e.g., processors) needed by T_i and $0 < w_{ij} \leq 1$ denotes the utilization demanded from node j .³ Notice that by definition, the k_i nodes requested by T_i must be mapped to different nodes in the hosting graph G . When $w_{i1} = w_{i2} = \dots = w_{ik_i}$, we call T_i a *uniform parallel process* and we refer to the resulting game as a *uniform PPCG* (non-uniform PPCG, otherwise).

As described above, in PPCG there are no topological constraints on the set of nodes requested by a task (process) since graph T_i featured no edges. In general, however, a parallel process may

³ Clearly, our standard PCG is a special case of PPCG where $k_i = 1$.

specify a particular topology, which must be satisfied in any valid mapping of the task graph onto the hosting graph. Of particular interest are parallel process collocation games in which the parallel processes have regular topologies (*e.g.*, a ring, a mesh, a hypercube, *etc.*) Also, as described above, in PPCG the hosting graph was a complete graph. In general, however, the hosting graph itself may feature a regular topology onto which the parallel process structures would be mapped. Again, of special interest are hosting graphs that exhibit regular topologies.

Applications of Collocation Games: We conclude this section with a set of distributed resource management problems, illustrating how each such problem may be cast as a collocation game.

Content Distribution Networks: Caching content closer to its consumers is the cornerstone of CDN applications [4, 22, 13, 14]. If CDN overlays were able to acquire physical resources on demand (as enabled by services such as Amazon CloudFront [2]), they could potentially reduce their cost by not having to pay for a fixed infrastructure, and by adapting quickly to bursts in demand. Each CDN has therefore varying requirements for allocating storage/processing power and nodes, and network capacity. Viewing each overlay as a player, multiple players may achieve economies of scale by strategically collocating their services. CDN collocation is an example of GCGs, in which the hosting and task graphs may exhibit arbitrary topologies, dimensionality, *etc.*

Media Streaming Networks: Online media streaming uses various kinds of overlays to deliver content over the network [18, 9, 26]. In this case, bandwidth, delay and jitter are important QoS parameters for allocating network links, as well as CPU capacity for packet forwarding at intermediate nodes. The resulting overlay may be a tree, multiple parallel trees or a mesh. In all such cases, the content providers may collocate portions of their overlays to reduce their costs. Collocation of media streaming networks is an example of PPCG, in which the hosting and task graphs may exhibit regular topologies and QoS constraints.

Service Oriented Architectures - Service composition: In SOA/Grid architectures an application is specified as the composition of services. A natural way to describe these systems is as a directed graph where nodes are services and edges are the communication between services. Nodes and edges can be labelled with their computational/communication demands. Existing work, *e.g.* [1, 17], is based on the assumption that applications do not compete for the shared resources, which is precisely what collocation games would enable.

Virtual Machine Collocation: Server virtualization at the datacenter enables better utilization of resources. In some cases, the providers offer very coarse allocation/pricing schemes. Coarse grained allocation policies lock customers into configurations that may cost them significantly more than what they really need. Providers have no incentive to allow users to mix and match, not to mention interact with other providers. On the other hand, if customers themselves are able to identify each other in a distributed peer-to-peer fashion, allocate and partition the servers they need according to their needs, a tighter packing is possible and both the total number of allocated servers and the average cost would be reduced. VM collocation is an example of PCG.

Workflow Scheduling: A workflow represents a complex process composed of many subtasks. Workflows are commonly represented as directed-acyclic graphs, where nodes correspond to subtasks and edges to dependency relationships. A very prominent problem within the high-performance computing literature is that of minimizing the *makespan* of a collection of tasks, which again is supported on the assumption that all the tasks will cooperate by following the dictated optimal schedule. On the other hand, if tasks are controlled by independent selfish agents, this assumption no longer holds, and the collocation games offers a more appropriate model.

3 Analytical Results

Nash Equilibrium of GCG: The GCG does not necessarily have a Nash Equilibrium (NE), as illustrated by the examples in Figure 2.

Let us consider example (a). Here the hosting graph is an m -vertex ring. Each vertex is of unit capacity. Each of the n ($2 \leq n < m$) tasks consists of two connected vertices, with utilization

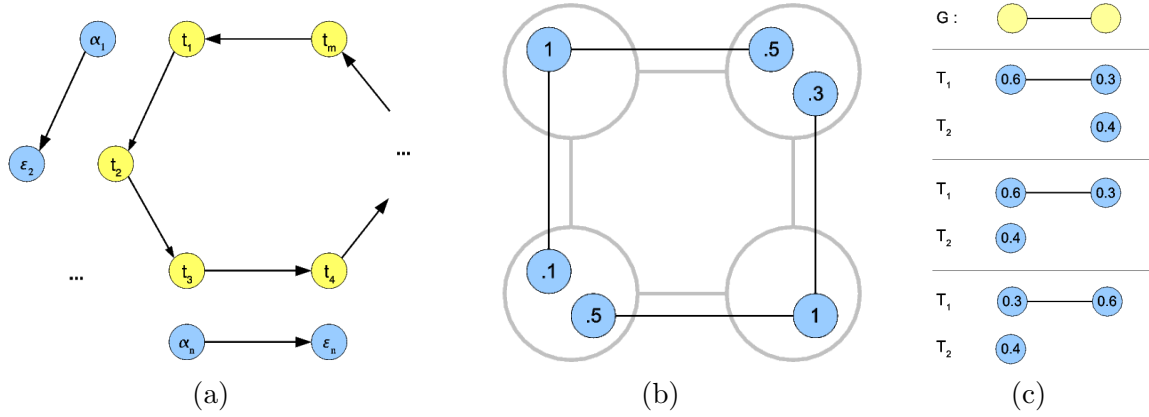


Figure 2: Examples of Collocation Games with no Nash Equilibria.

requirements $1/2 < \alpha_i \leq \alpha_{max} < 1$ and $0 < \epsilon_i < 1 - \alpha_{max}$, respectively.

Observe that two tasks, say T_1, T_2 , cannot be assigned to the same hosting nodes t_i, t_{i+1} , as the sum of two α nodes exceeds one. So, feasible configurations are consecutive or disjoint nodes. As $n < m$, there will always be at least one edge ($\alpha \rightarrow \epsilon$) connecting a pair of unmatched nodes, as some segment of the hosting graph will not be used. Without loss of generality, let T_1 be the task whose α node is free, and task T_2 be the task whose ϵ node is free. The GCG cost function implies that the best strategy for T_2 is to move so that its ϵ node shares the hosting node with the α node of T_1 . If we consider the set of strategies of all tasks, no matter what strategy T_i chooses, there will always be at least one free edge, and the task T_j whose ϵ node is free will be better off relocating. Since this holds for all possible strategies of all tasks, we conclude that there is no equilibrium.

Similarly, in example (b) the interests of both players conflict. In this example each player asks for three different CPUs, with the additional requirement that the CPUs must be adjacent. In the configuration shown the top player would benefit by swapping the position of tasks 0.5 and 0.1, as its cost goes from 1.79 to 1.75. After this move, the best response for the other player is to swap its 0.5 and 0.3 nodes as his cost reduces from 2.25 to 2.20. At this point, this configuration is the symmetric of the original one, and the game keeps going forever.

Example (c) illustrates even another case, where the small task (node of weight 0.4) prefers to join the node of value 0.6 of the large task, which gives a cost of 0.4 the minimum possible for the player. However, the large player prefers to match its 0.3 node to the other player's 0.4. This gives a minimum cost of 1.43 for the large player. As both players can keep switching positions forever, the game never reaches an equilibrium.

Theorem 1. *Determining whether a GCG has a Nash Equilibrium is NP-Complete.*

Proof. Consider some instance of the 3-SAT problem, with m clauses and n variables as given below:

$$\begin{aligned} \phi &= C_1 \wedge \dots \wedge C_m \\ C_k &= l_{k1} \vee l_{k2} \vee l_{k3} \end{aligned} \quad , \quad \text{where} \quad \begin{aligned} l_{ki} &\in \{x_j, \neg x_j\} \\ x_j &\in \{x_1, \dots, x_n\} \end{aligned}$$

We show how to construct a GCG corresponding to the 3-SAT problem as follows (Figure 3 illustrates this construction).

Consider a set of $m + 1$ task graphs of the form $\epsilon \rightarrow \alpha$, such that $\alpha > 1/2$ and $\epsilon < (1 - \alpha)/m$. There is also a second set of tasks of the form $\delta \rightarrow 1$ with $\delta = 1 - m\epsilon$. For this collection of task graphs, it is only feasible to share allocations for ϵ nodes.

Consider a hosting graph with n pairs of nodes x_j and $\neg x_j$, with pairs of edges $x_j \rightarrow \neg x_j$ and $\neg x_j \rightarrow x_j$, for $1 \leq j \leq n$, corresponding to the variables of the 3-SAT problem. The graph also has m nodes C_k , for $1 \leq k \leq m$, corresponding to the clauses of the 3-SAT problem, as well as an edge

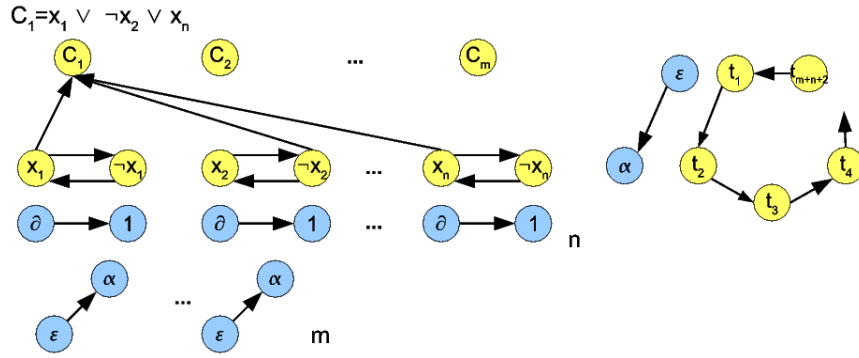


Figure 3: Construction used in proving that establishing NE for GCG is NP-Complete.

per clause term l_{ki} connecting each of the variables x_j and $\neg x_j$ in a clause to the corresponding C_k node. By definition of 3-SAT there will be $3m$ of these edges. Finally, the hosting graph includes a ring of $m + n + 2$ nodes $t_1, t_2, \dots, t_{m+n+2}$. Set the costs and capacities of all nodes and edges to one.

Claim: If the 3-SAT problem instance is satisfiable, then there is a Nash equilibrium for the corresponding Colocation Game; otherwise there is no Nash equilibrium.

First, we consider the case when the 3-SAT problem is satisfiable. We assign the GCG tasks as follows: There is at least one l_{ki} edge used in the true assignment per clause. Place an $(\epsilon \rightarrow \alpha)$ task from the variable node, to the corresponding clause node. Map the n $(\delta \rightarrow 1)$ tasks to the n pairs of $(x_j \leftrightarrow \neg x_j)$ nodes by matching the ϵ nodes to the δ nodes. Because 3-SAT is satisfiable, this assigns all the $\delta \rightarrow 1$ tasks and m of the $\epsilon \rightarrow \alpha$ tasks. Place the last $(\epsilon \rightarrow \alpha)$ edge in the ring. This configuration is a Nash Equilibrium as it is easy to check that there are no other assignments that avoid placing edges in the ring (making it a non-stable configuration), and there is no task with an alternative cost-reducing position.

Now consider the case when the 3-SAT problem is unsatisfiable. From our previous ring example, no configuration with more than one $(\epsilon \rightarrow \alpha)$ tasks in the ring is stable. So, we place one in the ring and try to place the others outside. We can place n on the edges $(x_j \rightarrow \neg x_j)$, but still would have m to place elsewhere. Because 3-SAT is not satisfiable, there are less than m edges $l_{ki} \rightarrow C_k$ and we are forced to place at least one more task in the ring, giving a configuration with no equilibrium. \square

Nash Equilibrium of PCG: We define $P = (\dots, p_i = P_i/U_i, \dots, p_m)$ as the vector of ratios of prices to utilizations for all resources. In addition, we introduce a sorting function $\mathcal{L}(P) = (p^{(1)}, p^{(2)}, \dots, p^{(m)})$, whose image is the vector of sorted components of P , *i.e.*, $p^{(1)} \leq p^{(2)} \leq \dots \leq p^{(m)}$. Consider the evolution of $\mathcal{L}(P)$ as a result of making a valid move. In particular, define P the vector before the move, and P' the vector after the move. The following lemma shows that the succession of vectors $\mathcal{L}(P)$ is lexicographically ordered.

Lemma 1. *In a PCG, the ordered vector of unclaimed spaces after a valid move dominates the previous vector: $\mathcal{L}(P') < \mathcal{L}(P)$, where domination implies that $p^{(i)} \leq p^{(i)}$ for all $i = 1 \dots k$, with strict inequality for k , $p^{(k)} < p^{(k)}$.*

Proof. Assume task x_j is moving from a to b and that the sorted vectors of price ratios are

$$\mathcal{L}(P) = \left(\dots, \frac{P_a}{U_a}, \dots \right) \quad \text{and} \quad \mathcal{L}(P') = \left(\dots, \frac{P_b}{U'_b}, \dots \right)$$

By (5) we know that $P_b/U'_b < P_a/U_a$, which implies that P_b/U'_b will be either in the same position, or if it becomes smaller than one of the predecessors of P_a/U_a , it will move to the left. In either case, the element P_b/U'_b is strictly less than the corresponding element of $\mathcal{L}(P)$, and all other predecessors remain unchanged, therefore $\mathcal{L}(P')$ dominates $\mathcal{L}(P)$. \square

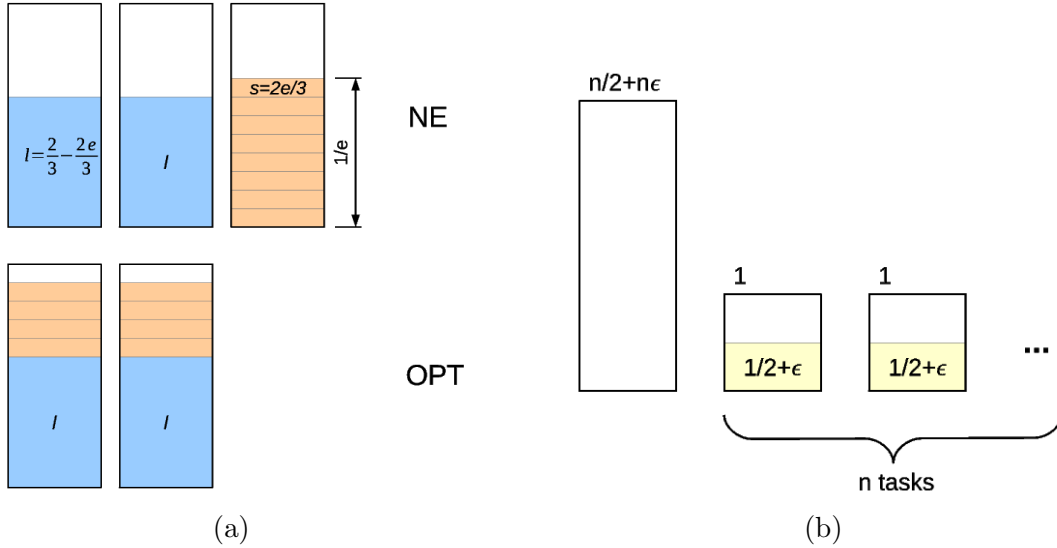


Figure 4: Examples illustrating PCG PoA for (a) homogeneous and (b) heterogeneous settings.

Theorem 2. *PCG converges to a Nash Equilibrium under better response dynamics.*

Proof. Lemma 1 defines a partial order in the set of vectors of price ratios. This partial order has one or more minimal elements and therefore the sequence of moves has to stop before, or at most when it reaches one of these minima. \square

Since PCG may not converge to a minimal element, it might be useful to know the *Price of Anarchy* (PoA), *i.e.* the ratio of the worst-case cost in an equilibrium to the cost in a socially-optimal solution. Figure 4 illustrates two examples, the first one for homogeneous resources and the second one for heterogeneous resources, where the equilibria are not optimal. In the example of part a) all the resources have capacity one, the top configuration is a NE and the bottom configuration is optimal (OPT). The large processes have utilizations (weights) $1/2 < l \leq 2/3 - 2e/3$, so they cannot be collocated. The small processes have size $s = 2e/3$ and there are $1/e \geq 8$ of them. So the utilization when they are all in the same bin is $2/3$. Individually, none will move with the large objects, as this does not reduce their cost, but when they all move with the large objects as in the bottom configuration, the cost of all players reduces and the social cost is minimum.

Theorem 3. *The price of anarchy for homogeneous PCG is $3/2$ while the price of the anarchy for heterogeneous PCG is 2 .*

Proof. Assuming resources of equal price and capacity (all normalized to 1), a lower bound for an optimal collocation is the summation of all process utilizations, and an upper bound is placing one process per resource. If there are n processes, we get:

$$\sum_{i=1}^n t_i \leq OPT \leq n \quad (9)$$

The PCG construction in figure 4(a) exemplifies a tight optimal collocation.

First, we observe that if there were more resources with processes of utilization $1/2 < l \leq 2/3 - 2e/3$, then the cost ratio between NE and OPT would be $(|L| + 1)/|L|$, where $|L|$ is the number of “large” processes. This ratio is maximized when $|L| = 2$. Alternatively, consider the case where there are more resources with low-utilization processes. This situation would not be an equilibrium as any process with utilization t_i in bin A would benefit by moving to bin B whenever $U_B + t_i \geq U_A$.

Therefore, the equilibrium would occur when all low-utilization processes have filled as much as possible some of the resources. In the worst case, it would have been possible to split such processes into two bins, with one large process of size $1/2 + \epsilon$ in each. This implies that a worst-case cost ratio of $3/2$ between NE and OPT.

As for the case where resources have non-uniform capacities (with proces proportional to capacity), the PoA can be bounded as illustrated by the example in Figure 4(b). The example shows a NE with cost n , where the optimal configuration has cost $n(1/2 + \epsilon)$, so this gives a lower bound $\Omega(2)$. Now, by contradiction assume an arbitrary collection of tasks such that $\sum t_i > n/2$ and whose PoA is greater than 2. By the definition of PoA and using (9) this would imply $n/\sum t_i > 2$, which is a contradiction of the fact that the initial set of tasks has $\sum t_i > n/2$. Therefore 2 is a tight bound on the PoA. \square

Convergence speed of PCG: It is possible to bound the number of moves that it takes for the game to reach a NE in the case of homogeneous resources (They all have the same capacity and the same price – without loss of generality we can set capacities and prices to be one). To so so, let's define the vector of unclaimed spaces

$$V = \mathbf{1}_n - U = (v_1, \dots, v_n)$$

then consider the following potential function

$$\phi(V) = v_{(1)} + \dots + nv_{(n)} \tag{10}$$

where n is the number of resources, and $v_{(i)}$ are the components of $\mathcal{L}(V)$. Let's also introduce the minimum allowed improvement ϵ as the threshold in the decrease of unclaimed space necessary to make a move. If the change is below this threshold, the move will not be performed. Then, when an object moves from j to i , the potential of resource i decreases by at least $\epsilon(n - i + 1)$ and the potential of j increases at least by $\epsilon(n - j + 1)$. The minimum change in potential is

$$\begin{aligned} \Delta\phi(V) &= \epsilon(n - i + 1) - \epsilon(n - j + 1) \\ &= \epsilon(i - j) \end{aligned}$$

which is minimal when moving between adjacent resources, in which case is ϵ .

At the beginning the potential is is at most

$$\phi(1, \dots, 1) = \sum_{i=1}^n i = \frac{n(n+1)}{2} \approx \frac{n^2}{2}$$

An upper bound on the number of moves m would then be the number of moves to go from the maximum potential to just below the threshold

$$\begin{aligned} m\epsilon &\geq \frac{n^2}{2} - \epsilon \\ m &\geq \frac{1}{\epsilon} \left(\frac{n^2}{2} \right) - 1 \\ m &= O(n^2) \end{aligned} \tag{11}$$

We now turn our attention to multidimensional PCG. According to (8), valid moves are defined by the ratios of price to utilization of the resources (a scalar quantity). Consequently, the exact same construction used for the unidimensional case describes the dynamics of price ratios vector for multidimensional PCG, yielding the following theorem.

Theorem 4. *Multidimensional PCG converges to a Nash equilibrium under better response dynamics.*

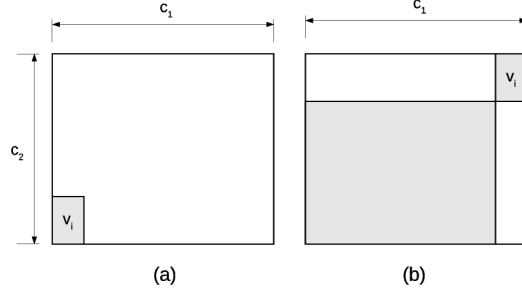


Figure 5: Worst and best cases for a player's cost

Lower bounding the worst-case cost paid by a player: As defined before, player i 's cost when placed in resource R is

$$P \cdot \frac{v_i}{\sum_{j \in R} v_j}$$

The worst-case for a player is when he is the only user of a resource (Figure 5a) and in this case he has to pay P . The best case is when the player only has to pay for its portion of the volume, and the volume taken by other players is maximum (Figure 5b). In this case the player has to pay

$$P_{best} = P \cdot \frac{v_i}{v_i + \prod_{k=1}^d (c_k - x_k)}$$

hence the ratio of the worst-case cost to the best case is:

$$\frac{worst}{best} = \frac{v_i + \prod_{k=1}^d (c_k - x_k)}{v_i} = 1 + \frac{\prod_{k=1}^d (c_k - x_k)}{\prod_{k=1}^d x_k}$$

In the continuous case it is unbounded, but considering a discrete representation of b -bits along each dimension, with a minimum permissible allocation of 1 unit, and a resource capacity along each dimension of $2^b - 1$, then when having d dimensions, the worst case ratio can be bounded as

$$\frac{worst}{best} = 1 + \frac{\prod_{k=1}^d (2^b - 2)}{\prod_{k=1}^d 1} = 1 + O(2^{bd}) \quad (12)$$

A practical consideration resulting from this analysis is that given the heterogeneity of the resources and of the task sizes, using a large number of bits to represent capacities and demands would severely hurt the worst-case cost for a player. Instead, subdividing the resources into classes and assigning tasks to the smallest class that can contain it, makes it possible to use a reduced number of bits per class. This subdivision also makes sense from the point of view of the cloud-resource provider: He will typically acquire machines by homogeneous batches, but each batch belonging to different hardware generations whose capacity typically corresponds to an exponential increments. Having various classes of homogeneous resources also brings the additional benefit that the price-of-anarchy is bounded (it is not for heterogeneous resources) and relatively small.

We conclude this section with equilibrium results for parallel PCG (PPCG).

Theorem 5. *On the one hand, PPCG with uniform processes converges to a Nash equilibrium under better response dynamics. On the other hand, better response dynamics for PPCG with non-uniform processes may cycle.*⁴

⁴ This theorem can be extended to the multidimensional version of PPCG.

Proof. In uniform PPCG, consider a better response of task T_i . Since all of T_i 's node utilizations are the same, and since one cannot colocate any pair of these nodes on the same hosting node, the better response of T_i can be realized by a sequence of moves involving only one node. Therefore, we can apply the same potential function as in Lemma 1 and Theorem 2 to show that the better response dynamics converges. Examples (b) and (c) in Figure 2 illustrate the case in which non-uniform PPCG cycles. \square

4 Local better response algorithms for PCG

When a player is about to make a move, the best response would be to solve a knapsack problem including its own task and the tasks of all the players using a given resource. Then, it would choose the best one across all possible resources. Of course, doing this is not realistic as the knapsack problem is NP-hard, and doing this on all resources would take a long time. Instead we defined a local better response movement that is easily implemented in a distributed way. In a local better response a player picks a server at random, queries the tasks currently allocated in the server and uses a heuristic solution for the knapsack problem resulting of considering its own tasks and all the tasks already in the server. If a cost-reducing solution is found, the player executes the move, otherwise this trial is lost and the player has to wait until its next turn to try again.

The knapsack problem itself is NP-complete, but there is a dynamic programming algorithm for solving it in pseudo-polynomial time for reasonable size instances. Given that in practice it will be expected to find several tens, or at most a few hundreds of processes per resource, the problem is tractable for practical scenarios. In addition, we also considered a branch & bound algorithm that does either a BFS or a DFS search, but bounds the degree and the depth of the search to limit the amount of time spent by the search. This alternative does not guarantee an optimal solution, but as far as player keep executing *better moves* the game is guaranteed to reach an equilibrium.

4.1 Dynamic programming valid move search

Referring back to eq. (8), notice that the problem of minimizing the cost of a task in a resource is the problem of maximizing U'_b , or equivalently minimizing the slack (unused capacity) of a resource. For this analysis let us define a workload as a set of tasks $W = \{T_k, \dots\}$, and the slack as $s(W) = R - \sum_{T_i \in W} T_i$, where R is the capacity vector of the resource. The quantity we want to minimize is the volume of the slack $v(W) = \prod s(W)$. For the construction of the dynamic programming solution, we will assume demands and capacities are discrete b-bits quantities, in a d-dimensional space. In there are 2^b values per axis, the total number of discrete volumes is 2^{bd} .

The key observation in defining the dynamic programming algorithm is that the best solution that includes task T_i is the union of $\{T_i\}$ and the best solution for the resource of capacity $R - T_i$. If we proceed considering one additional task at a time, it may be the case that either the best solution for some capacity R is the same as when we considered only tasks T_1, \dots, T_{i-1} , or the alternative solution for $R - T_i$ union $\{T_i\}$. Therefore we can compute the overhead of the optimal solution by the recurrence:

$$A(i, j) = \min \{A(i-1, j), v(W_{(i-1,k)} \cup T_i)\} \quad (13)$$

where the column indexes (j, k) are over an enumeration of the capacity vectors whose increments are the discrete capacity steps, so in (13), k is the index of the column corresponding to the capacity of column j minus task T_i :

$$k = \text{column}(R_j - T_i)$$

As a boundary condition, for cases where $R_j - T_i \leq 0$ on any of its dimensions, the cost is ∞ , same for $A(0, j)$. Observe that the columns need not to be ordered. All is needed when computing row i is to have all the values of row $i-1$. Therefore the space requirement is limited to keeping in memory the current and the previous rows. If there are d dimensions and b bits per dimension, the total space required is $O(2^{bd})$. As for the time, the algorithm finishes when it reaches the last column of the last

```

function searchBetterMove(Set of tasks T)
  A(0,:) = Infinity
  tasksPerColumn(0,:) = {empty}
  maxColumn = 2^(b*d) // Bits times Dimensions
  foreach i in T
    for j=0 to maxColumn
      residualVolume = volume(j)-volume(i)
      if residualVolume>0
        k = columnOf(residualVolume)
        cost = volumen( tasksPerColumn(0,k) Union i )
        if cost<A(0,j)
          tasksPerColumn(1,k) = tasksPerColumn(0,k) Union i
          A(1,j) = cost
        endif
      endif
    endfor
    A(0,:) = A(1,:)
    tasksPerColumn(0,:) = tasksPerColumn(1,:)
  endfor
endfunction

```

Figure 6: Dynamic Programming Better Move Search Pseudocode

row, therefore it m is the number of tasks, the time is $O(m2^{bd})$. This solution suffers from the *curse of dimensionality* for large values of d or for high granularities though.

Figure 6 gives the pseudocode for this algorithm.

4.2 Branch and Bound valid move search

Figure 7 shows the B&B algorithm. It takes two parameters *degree* and *maxDepth* that limit the portion of the search space explored. If the task fits in the given resource, it returns immediately. Otherwise, it looks for the minimal set of tasks that need to be replaced in order for the new task to fit. In this pseudocode, *list* is a data structure that may be implemented as a FIFO queue to give a BFS search, or as a LIFO queue to give a DFS search. The algorithm keeps track of the best solution found so far and when it is done (either because it was able to explore the full search tree or it reaches the boundaries established by *degree* and *maxDepth*) it returns its current best solution.

4.3 Complete local better response procedure

Using one of the valid move search algorithms outlined above, the procedure is

1. Player k randomly picks a resource j of capacity R_j and price P_j . The set of tasks currently in j is V_j , and the new task k (with demand vector X_k , currently in a resource of price P_0 with total utilization U_0) that is probing if there is a move into j
2. Solve the following knapsack problem: The knapsack capacity is $R_j - X_k$ and the set of tasks is V_j . Say $S \subseteq V_j$ is the solution set. Let v_i be the volume of task i . Then, there is a valid move if

$$\begin{aligned}
\sum_{i \in S \cup \{k\}} X_i &\leq R_j \\
\sum_{i \in S \cup \{k\}} v_i &> \sum_{i \in V_j} v_i \\
\frac{P_j}{\sum_{i \in S \cup \{k\}} v_i} &< \frac{P_0}{U_0}
\end{aligned}$$

```

function initialize(Task T, Resource R)
    need = R.utilization()-(R.capacity()-T.volume())
    list = {}
    tasksToReplace = {}
    if need<0
        return tasksToReplace // Move found no task needs to be replaced
    for i=1 to degree
        chosen = random not chosen task from R.tasks()
        node.parent = null
        node.task = chosen
        node.sumVolume = chosen.volume
        node.depth = 1
        list.add(node)
    endfor
    bestSolution = search(list, T, R)
    if (bestSolution != null)
        tasksToReplace = set of tasks in the chain from
            bestSolution to the root
        return tasksToReplace
    else
        return "no better move found"
    endif
endfunction

function search(list, T, R)
    bestCost = infinity
    bestSolution = null
    while( ! list.isEmpty() )
        node = list.remove()
        newVolume = R.sumVolume() - node.sumVolume + T.volume()
        newCost = R.price()*T.volume()/newVolume
        if (newVolume>R.sumVolume() and newCost<T.cost()
            and newCost<bestCost)
            bestCost = newCost
            bestSolution = node
        else
            if (node.depth<maxDepth)
                for i=1 to degree
                    chosen = random not chosen task
                        from R.tasks()
                    child.parent = node
                    child.task = chosen
                    child.sumVolume = node.sumVolume
                        + chosen.volume
                    child.depth = node.depth+1
                    list.add(child)
                endfor
            endif
        endif
    endwhile
    return bestSolution
endfunction

```

Figure 7: Branch and Bound Randomized Better Move Pseudocode

where the first condition is the feasibility and it is implicit in any knapsack solution, the second condition is the total utilization increase that assures that the cost of the remaining tasks will decrease, and the third condition is the cost reduction for the player that is moving. The difference set $V_j \setminus S$ are the tasks that will execute a *replacement move*.

3. If there was a valid move, then execute it. The set of tasks set $V_j \setminus S$ can be temporarily placed in an empty resource, and upon their turn they can improve their position too.

In addition, to reduce the total number of moves, we also implemented a cost threshold heuristic: a task will only move if the cost improvement is at least $p\%$ of its current cost. From a practical point of view it is worthless executing a move if the cost savings are too small.

5 PCG: Experimental Evaluation

To explore the potential from using collocation games as the underlying mechanism for enabling rational, selfish users to make cost-effective use of a distributed (cloud) infrastructure available through an independent provider who has no economic incentive to do so, we conducted a series of experiments for variants of the process collocation game, which are natural candidates for resource management in cloud computing environments. In this section, we summarize results from these experiments.

Data Sets and PCG Variants: In our experiments, we used synthetically-generated as well as trace-driven (PlanetLab) workloads, which we applied to unidimensional and multidimensional variants of PCG under both a homogeneous and heterogeneous resource model.

Our *synthetic workloads* give us the flexibility of varying the number of resources, number of tasks, and dimensionality. More importantly, it enables us to experiment with workloads for which we know (by construction) that a socially-optimal solution exists. We do so by repeatedly fragmenting the full capacities of a set of resources and then using the resulting fragments as a representation of the utilization (demand) of a set of tasks. By construction, we know that a “perfect” collocation exists (with every resource being fully utilized). Next, we place the resulting tasks in a much larger set of resources, which constitute our initial configuration. We have generated and experimented with a number of datasets for different versions of PCG. Here we present results for unidimensional and 3-dimensional PCG workloads when resources are homogeneous (of uniform capacities) as well as heterogeneous.

Our *trace-driven workloads* were constructed using publicly available PlanetLab traces of CoMon [21, 10]. PlanetLab is an example of a hosting infrastructure (much like the ones we entertain in this paper) which allows applied networking researchers to submit tasks of various resource utilization needs (demand) for hosting on PlanetLab servers. The traces we used give us snapshots of PlanetLab server capacities as well as the utilization of the slices assigned to the various tasks collocated on each server. The main advantage of this dataset is that it gives us a realistic distribution of typical task utilizations on a fairly large scale. Its disadvantage is that one cannot compare the configurations resulting from the collocation game to socially-optimal configurations – the latter being infeasible to compute due to PlanetLab’s scale. Thus for these workloads, we report the worst/best cost ratios for a series of runs of the same game instance. In PlanetLab, server capacities as well as slice utilizations (corresponding to the utilizations of a slice’s CPU, memory, uplink, and downlink). This makes PlanetLab an example of an infrastructure on top of which a multidimensional PCG game may be implemented. As with synthetic workloads, our PlanetLab experiments considered both homogeneous and heterogeneous resource models.

Player Response: As computation of a player’s best response is an NP-complete problem, in our implementation, we settled on and implemented three better response strategies: (1) A dynamic-programming knapsack solution (DPKP), (2) A depth-first search (DFS) with branch-and-bound pruning, and (3) a breadth-first search (BFS) also with branch-and-bound pruning (see §4 for more details).

Metrics: To characterize the PCG game dynamics, we evaluated a number of metrics, namely: (1) the *collocation ratio* which we define to be the ratio of average/optimal social cost (for synthetic

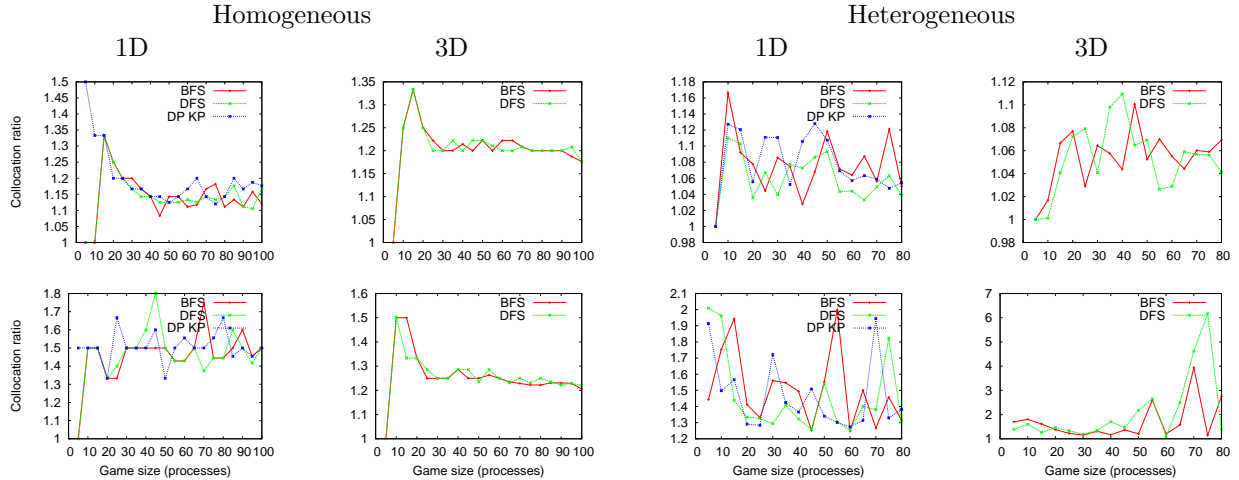


Figure 8: Median (top) and worst-case (bottom) collocation ratio for synthetic workloads.

workloads) and of worst/best social cost (for trace-driven workloads); (2) the total number of move *trials* until an equilibrium is reached; and (3) the total number of actual *moves* until an equilibrium is reached. The collocation ratio characterizes the (in)efficiency of the collocation resulting from playing the game, and is bounded by the price of anarchy (PoA), which we derived analytically for unidimensional PCG. The number of trials gives us insight as to the total time it takes for the game to reach an equilibrium. The number of moves gives us insight in the overhead involved in relocating players (tasks) since each relocation involves migration costs, *etc.*

To compute all three metrics, we need to establish whether an equilibrium has been reached. As shown earlier in the paper, verifying that the game is at a NE is an NP-hard problem. Thus, the criterion we used to declare that an equilibrium has been reached was to set a threshold on the number of consecutive trials attempted without resulting in a move. The thresholds we used were such that doubling them (*e.g.*, from 500 to 1,000) did not change our metrics much.

Collocation Efficiency for Synthetic Workloads: Figure 8 shows the median (over 100 samples) of the collocation ratios for synthetically-generated workloads in both homogeneous and heterogeneous settings. Recall that in a homogeneous (heterogeneous) setting all resources are (not) of equal capacities. These results show that the PoA bound (of $3/2$ for homogeneous cases and 2 for heterogeneous cases) for the collocation ratio holds for the median (1D). In the worst-case, there were a few samples above the bound, which we attribute to the approximate better-response computation and the threshold for detecting an equilibrium.

The results in Figure 8 show that the collocation ratio tends to decrease (*i.e.*, better efficiency) as the number of players increases, which bodes well for large-scale deployments. Also, our results show that collocation efficiency was basically independent of the better-response heuristic in use. Here we note that the branch-and-bound BFS and DFS heuristics we used were much faster than the dynamic programming (DPKP) heuristic. Indeed, DPKP was not able to handle many of our 3D and PlanetLab instances of PCG (hence the omitted results in Figure 8 for 3D cases).

Comparing our results for homogeneous versus heterogeneous settings, we note that the median collocation ratio in the latter setting was only slightly larger.

Collocation Ratio for PlanetLab Workloads: Figure 9 shows the median collocation ratio (ratio of worst/best social cost) using the task specifications derived from the PlanetLab traces. These results imply a relatively small collocation ratio, which as with synthetic workloads seems to be independent of the better response heuristic used. As shown in Figure 9, the worst-case collocation ratios were not too far off.⁵

⁵ Here we note that the abundance of small tasks in the trace significantly improve the chances of achieving better placements.

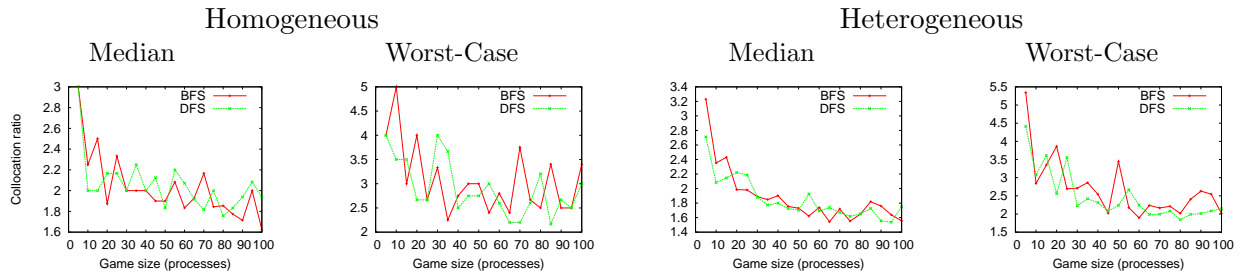


Figure 9: Median and worst-case collocation ratio for PlanetLab workloads.

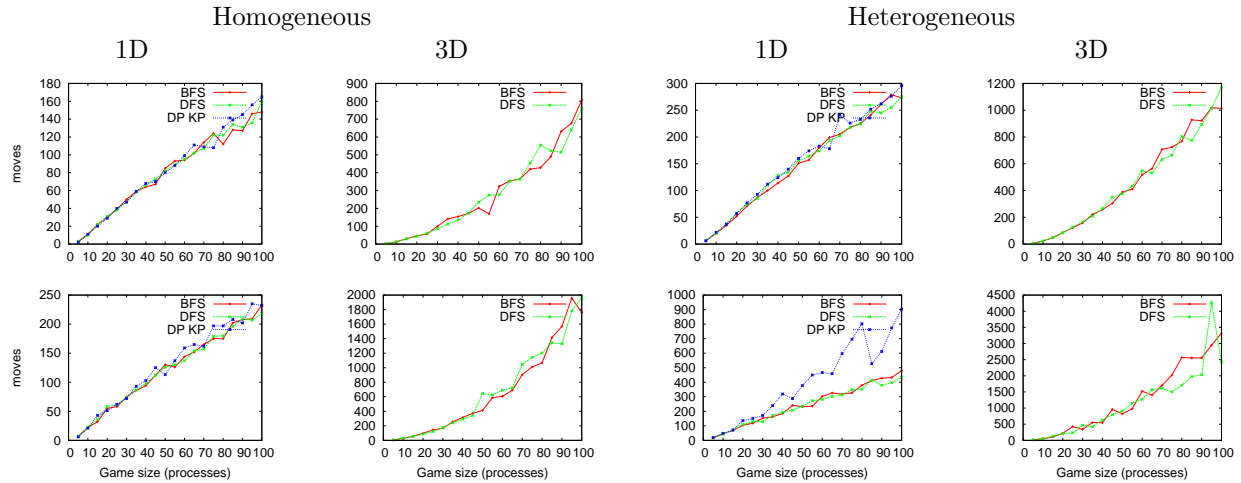


Figure 10: Median (top) and worst-case (bottom) number of moves for synthetic workloads.

Convergence Speed and Overhead: Figures 10 and 11 show the number of moves it takes to reach equilibrium. They indicate that the number of moves is directly related to the number of tasks in the system and fairly independent by the heuristic used to compute the better response. The relation is essentially linear for unidimensional PCG and follows a power law for higher dimensionality PCGs.

Figures 12 and 13 show the number of trials to reach equilibrium. Similar to the number of moves, the number of trials follows a power law dependent on the dimensionality of the problem, almost linear for 1D settings.

We also experimented with various heuristics for determining the player who attempts the next move. These are 1) random, chooses the player who is going to make the trials uniformly at random, 2) round-robin, all players go around taking turns, and 3) worst-located first, players who are currently paying higher overhead costs get more chances to move. To evaluate the effect of the selection heuristic on the convergence of the game, Figure 14 shows the number of moves and trials under the various strategies. In general the effect is minimal suggesting that the convergence speed

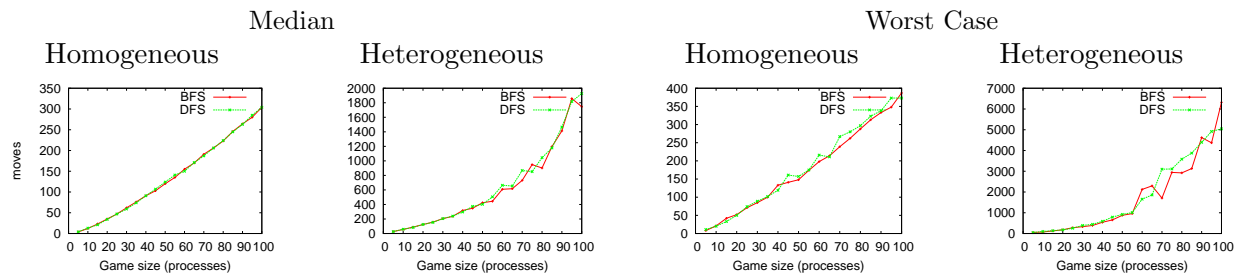


Figure 11: Number of moves for PlanetLab workloads.

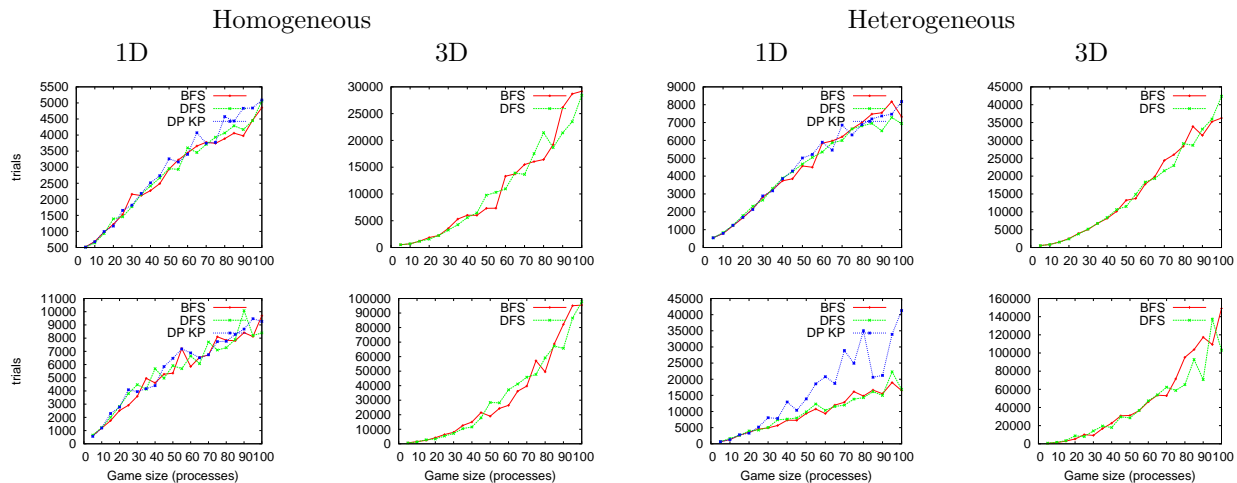


Figure 12: Median (top) and worst-case (bottom) number of trials for synthetic workloads.

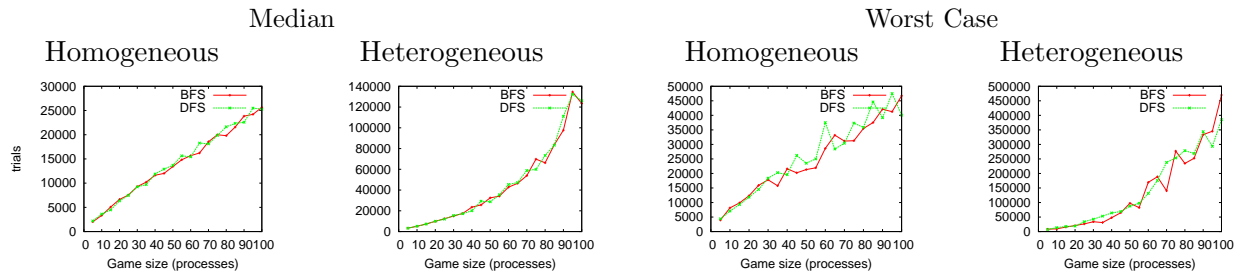


Figure 13: Number of trials for PlanetLab workloads.

is independent of the selection policy.

6 Conclusion

Collocation Games offer a natural paradigm for modeling and analyzing the dynamics that are likely to result when rational, selfish parties (users) interact in an attempt to minimize the individual costs they incur to secure the shared infrastructure resources necessary to support the QoS or SLA requirements of their applications. Collocation games offer an attractive alternative to approaches that require such parties to trust infrastructure providers (who have no vested interest in minimizing user costs, and may indeed have the exact opposite incentive) or those that expect such parties to be altruistic or to accept best-effort (as opposed to reservation-based) approaches that do not guarantee performance isolation.

In this paper we introduced the general collocation game (GCG) as well as the process collocation game (PCG), a more restricted version of GCG, along with variants of PCG that allow for unidimensional or multidimensional resources, homogeneous or heterogeneous resource capacities, and scalar or parallel resource configurations. We presented analytical results related to the existence and complexity of Nash equilibria for a number of these games. Also, using both synthetic and trace-driven workloads, we presented results from extensive empirical performance evaluation of practical and scalable implementations of the strategies underlying these games.

Collocation games are not only valuable as modeling and analysis tools, but also they provide a solid framework upon which purely distributed resource acquisition and management protocols may be conceived for emerging cloud computing, grid, and peer-to-peer overlays. In this paper we have shown that although best response computation may be expensive, computationally-efficient better-response heuristics are quite promising. We are currently exploring protocol designs that will enable

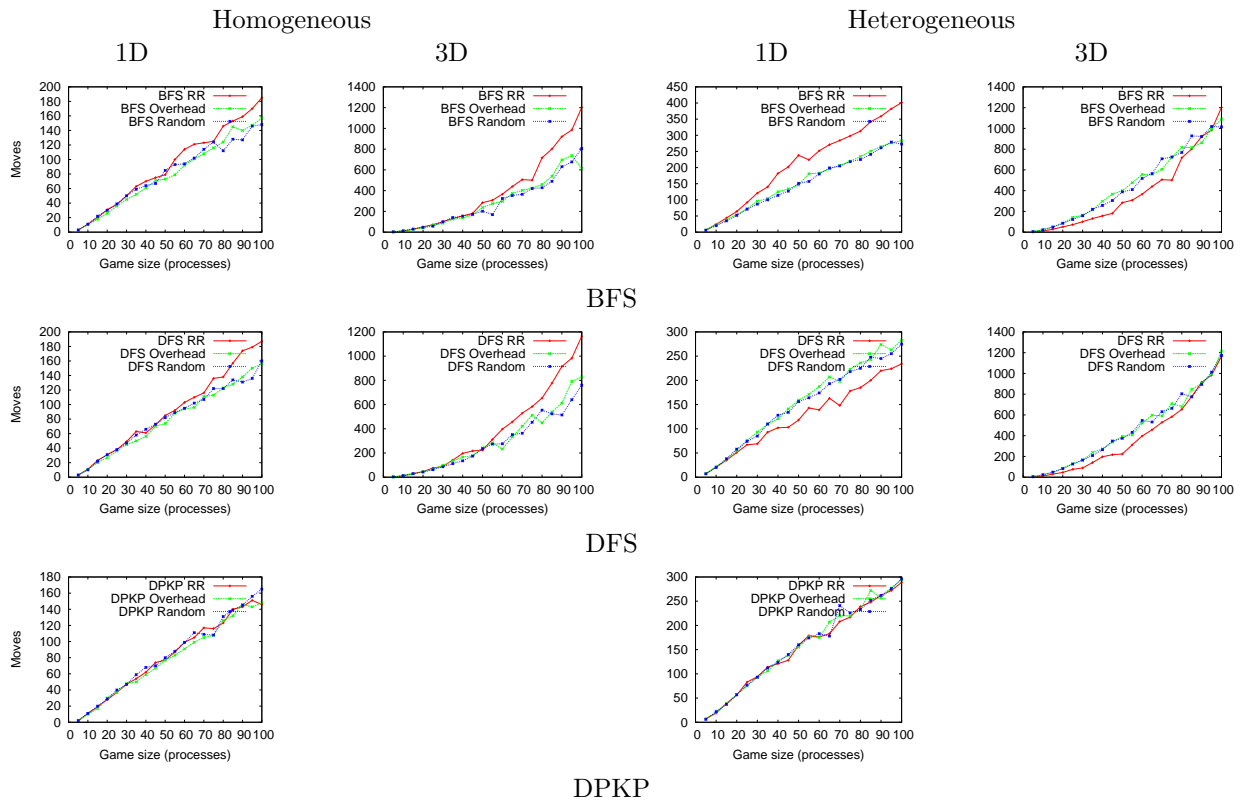


Figure 14: Comparing player selection strategies – Median number of Moves – Synthetic dataset

peers (*e.g.*, in a P2P system) to use collocation games to efficiently self-organize and to rationally and selfishly optimize their individual resource acquisition costs without reliance on any central authority.

References

- [1] J. Albrecht, D. Oppenheimer, D. Patterson, and A. Vahdat. Design and Implementation Tradeoffs for Wide-Area Resource Discovery. *ACM Transactions on Internet Technology (TOIT)*, 8(2), May 2008.
- [2] Amazon.com, Inc. Amazon CloudFront. <http://aws.amazon.com/cloudfront/>, Feb 2009.
- [3] Amazon.com, Inc. Amazon Elastic Computing Cloud (Amazon EC2). <http://aws.amazon.com/ec2/>, Feb 2009.
- [4] S. Annapureddy, M. J. Freedman, and D. Mazières. Shark: Scaling file servers via cooperative caching. In *NSDI'05*, May 2005.
- [5] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *Foundations Of Computer Science (FOCS04)*, 2004.
- [6] A. AuYoung, B. Chun, A. Snoeren, and A. Vahdat. Resource allocation in federated distributed computing infrastructures. In *Proceedings of the 1st Workshop on Operating System and Architectural Support for the On-demand IT Infrastructure*, October 2004.
- [7] B. Awerbuch, Y. Azar, and A. Epstein. The price of routing unsplittable flow. In *STOC'05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 57–66, New York, NY, USA, 2005. ACM.
- [8] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In *HPC ASIA 2000*, 2000.
- [9] Y. Chawathe. Scattercast: an adaptable broadcast distribution framework. *Multimedia Systems*, 9(1):104–118, July 2003.
- [10] CoMon. CoMon: A monitoring infrastructure for PlanetLab. <http://comon.cs.princeton.edu/>.
- [11] P. Crosman. Cloud Computing Begins to Gain Traction on Wall Street. <http://www.wallstreetandtech.com/it-infrastructure/showArticle.jhtml?articleID=212700913>, Jan 2009.
- [12] A. Czumaj, P. Krysta, and B. Vöcking. Selfish traffic allocation for server farms. In *STOC '02: Proceedings of the thirty-fourth annual ACM symposium on Theory of computing*, pages 287–296, New York, NY, USA, 2002. ACM.
- [13] M. J. Freedman, E. Freudenthal, and D. Mazières. Democratizing content publication with coral. In *1st USENIX/ACM Symposium on Networked Systems Design and Implementation*, March 2004.
- [14] M. J. Freedman, K. Lakshminarayanan, and D. Mazières. Oasis: Anycast for any service. In *3rd USENIX/ACM Symposium on Networked Systems Design and Implementation*, 2006.
- [15] Gartner, Inc. Gartner Says Cloud Computing Will Be As Influential As E-business. <http://www.gartner.com/it/page.jsp?id=707508>, Jun 2008.
- [16] J. Gomoluch and M. Schroeder. Performance evaluation of market-based resource allocation for grid computing. *Concurrency and Computation: Practice and Experience*, 16(5):469–475, 2004.
- [17] X. Gu, K. Nahrstedt, and B. Yu. Spidernet: An integrated peer-to-peer service composition framework. In *IEEE HPDC-13*, June 2004.
- [18] D. Helder and S. Jamin. End-host multicast communication using switch-trees protocols. In *Symposium on Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International*, pages 419–419, May 2002.
- [19] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Lecture Notes in Computer Science*, 1563:404–413, 1999.
- [20] N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 1st edition, 2007.
- [21] K. Park and V. S. Pai. CoMon: a mostly-scalable monitoring system for planetlab. *SIGOPS Oper. Syst. Rev.*, 40(1):65–74, 2006.
- [22] K. Park and V. S. Pai. Scale and performance in the coblitz large-file distribution service. In *NSDI'06*, May 2006.
- [23] A. Ricadela. Computing Heads for the Clouds. http://www.businessweek.com/technology/content/nov2007/tc20071116_379585.htm, Nov 2007.
- [24] T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the ACM*, 49(2):236–259, 2002.
- [25] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. G-commerce: Market formulations controlling resource allocation on the computational grid. In *IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 46, Washington, DC, USA, 2001. IEEE Computer Society.
- [26] B. Zhang, S. Jamin, and L. Zhang. Host multicast: a framework for delivering multicast to end users. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1366–1375, June 2002.