Towards Green Cloud Computing: Demand Allocation and Pricing Policies for Cloud Service Brokerage

Chenxi Qiu, Haiying Shen and Liuhua Chen
Department of Electrical and Computer Engineering
Clemson University
Clemson, USA
{*chenxiq, shenh, liuhuac*}@*clemson.edu*

***Abstract*—Functioning as an intermediary between tenants and cloud providers, cloud service brokerages (CSBs) can bring about great benefits to the cloud market. CSBs buy the cloud resources, i.e., servers, with lower prices from cloud providers and sell the resources to the tenants with higher prices. To maximize its own profit, a CSB may distribute tenants' requests to the clouds that waste energy resources. However, as energy costs of cloud computing have been increasing rapidly, there is a need for cloud providers to optimize energy efficiency while maintain high service level performance to tenants, not only for their own benefit but also for social welfares (e.g., protecting environment). Thus, for green cloud companies, two questions have arisen: 1) under what pricing policies from the cloud providers to the CSB, a profit-driven CSB is willing to minimize the total cloud energy cost while satisfy tenant demands and 2) how should a CSB distribute tenants' demands to achieve this objective? To address question 1), we find a pricing policy for cloud providers such that maximizing CSB's profit is equivalent to minimizing cloud providers' energy cost. To address question 2), we first devise a greedy solution, and then propose an approximation algorithm with a constant approximation ratio. Both simulation and real-world Amazon EC2 experimental results demonstrate the effectiveness of our pricing policy to incentivize CSBs to save energy for cloud providers and the superior performance of our algorithms in energy efficiency and resource utilizations in comparison with the previous algorithms.**

***Keywords*-cloud service brokerages; pricing policy; approximation algorithm**

I. INTRODUCTION

Though cloud computing is still in its relative infancy, it has earned rapid interest and adoption due to its advantages. *Cloud tenants* (e.g., DropBox) purchase cloud computing services from *cloud providers* (e.g., Amazon, Microsoft Azure). As innovative approaches continue to emerge in cloud computing, it is becoming clear that simple cloud interoperability between cloud tenants and cloud providers is often neither realistic nor the most advantageous. Currently, if a cloud tenant wants to use the clouds in multiple cloud providers, the tenant needs to negotiate multiple contracts with the cloud providers, which results in multiple payments, multiple data streams, and multiple providers to check up on. Then, tenants are faced with a problem of how to make the services from multiple cloud providers' work together to gain maximum profit and efficiency. However, determining the most advantageous ways to procure, implement and manage cloud technologies to handle this problem presents complex issues to cloud tenants. Under this circumstance, *cloud service brokerages (CSBs)* have arisen in the market [6], [8], [9].

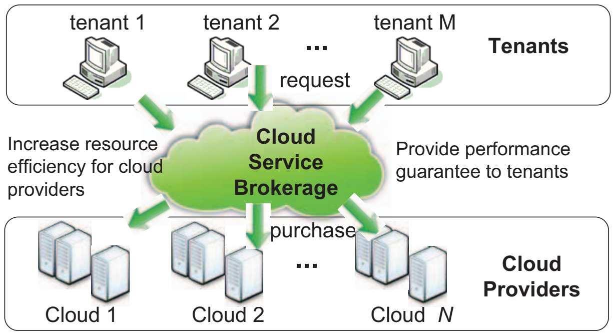


Figure 1. Cloud service brokerage.

As shown in Fig. 1, CSB is a third-party individual or business that acts as an intermediary between the tenants and the cloud providers. CSBs buy the cloud resources, i.e., servers, with lower prices from cloud providers [14] and sell the resources to the tenants with higher prices. In addition, CSBs can enhance the resource utilization of cloud services for tenants because they can monitor, track, protect and enforce company policies across all demands from different tenants (a demand can be a virtual machine (VM) in the IaaS (Infrastructure as a Service) model or a video game in the SaaS (Software as a Service) model). Thus, CSBs can make it easier, less expensive, safer and more productive for tenants to use cloud resources, particularly when tenants' requests span multiple and diverse cloud service providers.

To maximize its own profit, a CSB may distribute tenant requests to clouds which does not efficiently use cloud resources, since maximizing the CSB's profit does not mean minimizing the cloud providers' cost [14]. However, how to motivate a CSB to reduce energy cost (or cost for short) of cloud resources for green computing while satisfy all tenant requests has not been addressed. Indeed, energy consumption has been one of the most important issues in cloud computing [14]. The electricity consumption of clouds globally is 623 Billion kWh in 2007 and is projected to be 1,963 Billion kWh 2020, which will generate 1034 MtCO2e (Gigatonne Carbon Dioxide Equivalent) [12]. As energy costs of cloud computing have been increasing rapidly, there is a need for cloud providers to optimize energy efficiency while maintain high service level performance to tenants, not only for their own benefit but also for social welfares (e.g., protecting environment) [14]. To address this need, we attempt to explore the pricing policy of the cloud providers on CSBs to incentivize CSBs to save cloud energy cost and propose methods for a CSB to allocate tenants demands to the servers to minimize cloud energy cost. Specifically, we study two questions below:

Q1: *Under what pricing policies of cloud providers, when a CSB maximizes its profit, it can also minimize the total energy cost of all cloud providers.*

Q2: *How should a CSB distribute tenants demands to cloud providers to minimize total energy cost and meanwhile satisfy tenants demands?*

To address Q1, we first formulate two problems for CSBs: the Maximum CSB Profit problem (MCP) that aims to maximize a CSB's profit, and the Min-energy CSB Demand allocation problem (MCD) that aims to minimize cloud providers' energy cost when allocating tenant demands to servers. By analyzing these two problems, we find a pricing policy for cloud providers to CSBs, such that MCP is equivalent to MCD, i.e., maximizing a CSB's profit is equivalent to minimizing cloud providers' energy cost. In other words, under this pricing policy, even a profit-driven CSB will automatically save energy for cloud providers when maximizing its own profit.

To address Q2, we need to find the optimal solution of MCD, which can be regarded as a generalized version vector bin packing (VBP) problem [18], [19]. However, MCD cannot simply be solved using the existing VBP solutions (e.g., Best Fit Decreasing algorithm (BFD) [18], [19]), because unlike VBP, demands and servers (bins) in MCD have different requirements and capacities for each type of resource, and the energy costs for different types of servers are also different. To solve MCD, we first propose a greedy algorithm based on BFD, namely Balance Fit Decreasing algorithm (BaFD). It aims to balance each server's utilization on different types of resources when selecting a server for a demand because one resource bottleneck prevents fully utilizing other resources. We then propose an approximation algorithm for MCD using linear programming relaxation (or LP-relaxation). We summarize our contributions in below:
(1) We find a resource pricing policy for cloud providers to incentivize CSBs to minimize the cloud energy cost.
(2) We design a greedy algorithm (BaFD) and an approximation algorithm for a CSB to minimize cloud energy cost servers, and analyze algorithm performance.
(3) We test the performance of BaFD and the approximation algorithm in comparison with the previous algorithms by both trace-driven experiments on a simulator and on Amazon EC2 [1].

The remainder of this paper is organized as follows. Section II and Section III study the pricing problem and demand allocation problem for CSBs, respectively. Section IV evaluates the performance of our proposed schemes in comparison with other schemes. Section V presents related work. Section VI concludes this paper with remarks on our future work.

II. Pricing Policy for CSBs

Our objective in this paper is to minimize the energy consumption of cloud servers. However, in reality, CSBs are always profit-driven and they do not need to minimize the energy cost from cloud providers. To maximize its own profit, a selfish CSB may distribute tenants' demands to clouds, which cannot fully utilize the energy resources [14]. Even if a CSB is willing to save energy for cloud providers, it cannot get the information of servers' energy cost, which becomes an obstacle for the energy saving. In this section, we discuss how the cloud providers design a pricing policy to CSBs to incentivize profit-driven CSBs to minimize the total energy cost of multiple clouds while accommodate all tenant demands, even if CSBs are not aware of the energy cost of servers in clouds.

Consider a scenario composed of a CSB, M demands $V = \{v_1, ..., v_M\}$ from all the tenants, and N heterogenous servers $S = \{s_1, ..., s_N\}$ provided by L cloud providers $C = \{c_1, ..., c_L\}$. Here, a demand is defined as a tenant's request that can only be allocated to a single server. Hence, the applications that require multiple servers can be considered as the combinations of multiple demands. Since the resource consumptions for different demands are different, we can characterize each demand v_l by a K-dimensional vector $\mathbf{w}_l = [w_{l,1}, ..., w_{l,K}]$, called *consumption vector*, where K denotes the number of different types of resources (e.g., CPU, memory, and disk bandwidth). Here, each dimension $w_{l,j}$ represents the demand's consumption on type-j resource. A tenant may have multiple demands. Similarly, each server s_i can be characterized by a K-dimensional *capacity vector* $\mathbf{b}_i = [b_{i,1}, ..., b_{i,K}]^{\mathrm{T}}$, where each dimension $b_{i,j}$ represents the server's capacity on type-j resource. We normalize the entries of each $\mathbf{w}_l$ and each $\mathbf{b}_i$ through dividing $w_{l,k}$ and $b_{i,k}$ by $w_{\max} = \max_{l,k} w_{l,k}$ and $b_{\max} = \max_{i,k} b_{i,k}$ in all demands and all servers, respectively. We assume that the energy cost of each server is the same when it is used regardless of its resource utilization [17], and we denote the energy cost of each s_i by a_i. As in [27], we also assume that the servers in each cloud provider are homogeneous. That is, they have the same capacity vector and energy cost. We assume that the CSB knows the consumption vectors of all the tenants and the tenants' consumption vectors are all fixed [14]. We use indicator variable x_i to represent whether s_i is purchased by the CSB: if yes, $x_i = 1$; otherwise $x_i = 0$. We use indicator variable $y_{i,l}$ to denote whether demand v_l is distributed to cloud provider c_i: if yes, $y_{i,l} = 1$; otherwise $y_{i,l} = 0$. Like the pricing policy in IBM [11], in this paper, each cloud provider charges the CSB based on the number of servers. Let z_n denote the total number of servers that the CSB bought from cloud provider c_n, then $z_n = \sum_{s_i \in S_n} x_i$. Then, the money that the CSB needs to pay to c_n can be represented by $L_n(z_n)$, where each $L_n(\cdot)$ is a concave and twice differentiable pricing function [14]. Cloud providers provide the information of the pricing policy and the capacity vectors of their servers to the CSB when the CSB buys servers [14]. Recall that each demand has a consumption vector with the consumption for K types of resources. Then, we formally formulate the maximum profit problem of the CSB, namely *Maximum CSB Profit (MCP)* problem, as follows.

$$\begin{aligned} \min \quad & f'(\mathbf{z}) = \sum_n L_n(z_n) & (1)\\ \text{s.t.} \quad & g_{i,k}(\mathbf{x}, \mathbf{y}^i) = \sum_l y_{i,l} w_{l,k} - x_i b_{i,k} \le 0, \forall i, k & (2)\\ & h_l(\mathbf{y}_l) = \sum_i y_{i,l} - 1 = 0, \forall l & (3)\\ & x_i \in \mathbb{N}, \forall i, \ y_{i,l} \in \{0,1\}, \forall i, l & (4) \end{aligned}$$

where $\mathbf{z} = [z_1, ..., z_L]$, $\mathbf{x} = [x_1, ..., x_N]^{\mathrm{T}}$, $\mathbf{y}_l = [y_{1,l}, ..., y_{N,l}]^{\mathrm{T}}$ and $\mathbf{y}^i = [y_{i,l}, ..., y_{i,M}]$. Constraint (2) ensures that for each cloud provider, the capacities of the purchased servers are enough for its allocated demands. Constraint (3) ensures that each demand is allocated to one server. MCP is an integer non-linear programming problem, which can be solved by subgradient method or interior point method [5].

Similarly, we can formulate the problem that CSB aims to minimize the energy cost, called the Min-energy CSB demand allocation (MCD) problem, which is an integer linear programming (ILP) problem:

$$\begin{aligned} \min \quad & f(\mathbf{x}) = \sum_i a_i x_i & (5)\\ \text{s.t.} \quad & \text{Constraints } (2), (3), \text{ and } (4) \text{ in MCD.} & \\ & z_n = \sum_{s_i \in S_n} x_i, \ \forall n. & (6) \end{aligned}$$

If MCP is equivalent to MCD, then when the CSB strives to maximize its own profit, it also minimizes the total energy cost of all clouds to accommodate all tenant demands, i.e., the CSB becomes cooperative. However, MCD is not equivalent to MCP in general because their objective functions are different. This leads to a question: what pricing policies (i.e., $L_n(\cdot)$) make MCP equivalent to MCD?

In the following, we answer this question by Theorem 2.1, which gives a necessary and sufficient condition to make the objective functions of MCD and MCP equivalent. To help prove Theorem 2.1, we first introduce Lemma 2.1.

Lemma 2.1: The objective function $f'(\mathbf{z})$ is concave.

Proof: Recall that $f'(\mathbf{z}) = \sum_i L_n(z_n)$. Then, the Hessian matrix of $f'(\mathbf{z})$ is

$$\begin{aligned} H_{f'}(\mathbf{z}) &= \begin{bmatrix} \frac{\partial L^2(\mathbf{z})}{\partial z_1^2} & \cdots & \frac{\partial L^2(\mathbf{z})}{\partial z_1 \partial z_L} \\ \vdots & \ddots & \vdots \\ \frac{\partial L^2(\mathbf{z})}{\partial z_L \partial z_1} & \cdots & \frac{\partial L^2(\mathbf{x})}{\partial z_L^2} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial L^2(\mathbf{z})}{\partial z_1^2} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial L^2(\mathbf{z})}{\partial z_N^2} \end{bmatrix} \end{aligned}$$

which is a *diagonal matrix*. Since each $L_n(z_n)$ is concave, $\frac{\partial^2 L_i(z_n)}{\partial z_n^2} \le 0$. Hence, $H_{f'}(\mathbf{z})$ is *negative semi-definite* [5], which implies that $f'(\mathbf{z})$ is a concave function. ■

Theorem 2.1: MCP and MCD have the same optimal solution iff $L_n(z) = \beta a_n z, \ \forall i$ where $\beta > 0$ is a constant.

Proof: Let H^* and C^* be the hyperplane and the hypersurface passing through the optimal point $\mathbf{z}^* = [z_1^*, ..., z_L^*]$, respectively: $H^* : \sum_n a_n z_n - c_A^* = 0$ and $C^* : \sum_n L_n(z_n) - c_L^* = 0$, where $c_A^* = \sum_n a_n z_n^*$ and $c_L^* = \sum_n L_n(z_n^*)$. Let $\mathcal{S}_{\mathbf{z}}$ denote the feasible region of $\mathbf{z}$. Because $\mathbf{z}^*$ is the optimal solution of MCD, then for each $\mathbf{z}' = [z_1', ..., z_L'] \in \mathcal{S}_{\mathbf{z}}$, we have $\sum_i a_n x_n' \ge c_A^*, \ n = 1, 2, ..., L$. Let $\mathbf{z}^n = z^n \mathbf{e}^n$ be the point that H^* intersects with z_n axis, respectively, where $\mathbf{e}^n$ is a N dimension vector with all entries equaling 0 except the n^{th} entry equaling 1. Then, $\mathbf{z}$ can be represented as a linear combination of $\mathbf{z}^1$, ..., $\mathbf{z}^L$: $\mathbf{z}^* = \sum_n \lambda_n \mathbf{z}^n$, where $\sum_n \lambda_n = 1$ and each $\lambda_n \ge 0$. Similarly, let $\hat{\mathbf{z}}^n = z^n \hat{\mathbf{e}}^n$ be the point that C^* intersects with x_n axis ($n = 1, 2, ..., L$), respectively. Since $f'(\mathbf{z})$ is concave (according to Lemma 2.1), we can derive that

$$\begin{aligned} f'(\mathbf{z}^*) &= f'\left(\sum_n \lambda_n \mathbf{z}^n\right) & (7)\\ &\ge \sum_n \lambda_n f'(\mathbf{z}^n), & (8) \end{aligned}$$

which implies that $\sum_n \lambda_n f'(\mathbf{z}^n) \le c_L^*$. On the other hand,

$$\begin{aligned} \sum_n \lambda_n f'(\hat{\mathbf{z}}^n) &= \sum_n \lambda_n c_L^* & (9)\\ &= c_L^*. & (10) \end{aligned}$$

Accordingly, $\sum_n \lambda_n f'(\mathbf{z}^n) \le \sum_n \lambda_n f'(\hat{\mathbf{z}}^n)$.

Now, we need to prove that $\hat{z}^n = z^n \ \forall n$, which makes C^* a hyperplane and $L_n(z) = \beta a_n z$, which completes the proof. For the sake of contradiction, suppose that $\exists n$ s.t. $\hat{z}^n \ne z^n$, then $\exists j$ s.t. $\hat{z}^j > z^j + \sigma$, where $\sigma > 0$. Then, we can always find a new instance of MCD that has feasible region $\lceil \frac{1}{\sigma} \rceil \cdot \mathcal{S}_{\mathbf{z}}$. In this newly constructed instance, the optimal solution of both MCD and MCP is $\lceil \frac{1}{\sigma} \rceil \mathbf{z}^*$, and $f'(\hat{\mathbf{z}}^n)$ intersects the z_j axis at point $\lceil \frac{1}{\sigma} \rceil \hat{z}_j \mathbf{e}^j$. Because $\hat{z}^j > z^j + \sigma$,

$$\left\lceil \frac{1}{\sigma} \right\rceil \hat{z}_j \mathbf{e}^j > \left\lceil \frac{1}{\sigma} \right\rceil x^j + \left\lceil \frac{1}{\sigma} \right\rceil \sigma \geq \left\lceil \frac{1}{\sigma} \right\rceil z^j + 1, \tag{11}$$

which implies there may exist a feasible point $\tilde{z}^j \mathbf{e}^j$ s.t.

$$\left\lceil \frac{1}{\sigma} \right\rceil z^j \leq \tilde{z}^j < \left\lceil \frac{1}{\sigma} \right\rceil \hat{z}_j \mathbf{e}^j \text{ and } z^j \in \mathbb{Z}. \tag{12}$$

Let $\tilde{z}^j \mathbf{e}^j$ be a feasible point in the newly constructed instance, then the second problem has a new optimal solution because

$$f_j(\tilde{z}^j) < f_j\left(\left\lceil \frac{1}{\sigma} \right\rceil \hat{z}^j\right) = \left\lceil \frac{1}{\sigma} \right\rceil c_L^*, \tag{13}$$

which is a contradiction. ■

According to Theorem 2.1, to encourage a profit-driven CSB to minimize the total energy cost of multiple clouds, the following pricing policy should be used: **the price of each server should be proportional to its energy cost.** Notice that Theorem 2.1 not only provides a feasible pricing policy, but also excludes all other pricing policies: If a pricing policy is not proportional to the energy cost, it cannot encourage a profit-driven CSB to minimize the energy cost.

Given the pricing policy, the MCD problem for CSB can be considered as a general version of the traditional demand allocation problem, which is equivalent to the VBP problem [6]. However, the methods for VBP, like FFD and BFD, cannot be directly applied to MCD, because in MCD the servers are chosen among multiple cloud providers, and it is non-trivial to find an optimal solution among heterogonous servers with different capacities and different prices. For example, some servers serve the requests more efficiently, but their prices are higher, so they may not be the optimal choice. In the following part, we will introduce how to solve this new demand allocation problem.

III. Min-energy CSB Demand Allocation

In this section, we address Q2 in Section I, i.e., how a CSB distributes its tenants requests to achieve the objective of minimizing the energy cost. Due to the hardness of this problem, we then propose two time-efficient algorithms: a greedy algorithm, called Balance Fit Decreasing (BaFD), based on the Best Fit Decreasing (BFD) algorithm [18], [19], and an approximation algorithm through linear programming (LP) relaxation.

A. A Heuristic: Balanced Best Fit Decreasing

When the servers in all different cloud providers have the same capacity for different resources, $b_{1,k} = b_{2,k} = \ldots = b_{N,k}$ $\forall k$, MCD maps to the classical optimization problem

Table I
Notations

Symbol	Description
C	The cloud provider set
c_m	The mth cloud provider
S	The server set
s_i	The ith server
V	The demand set
x_i	Variable indicating whether s_i is purchased
$y_{i,l}$	Variable indicating whether v_l is allocated to s_i
M	The number of demands
v_l	The lth tenant demand
N	The number of servers
a_i	The energy cost of s_i
$b_{i,k}$	The capacity of resource k of s_i
$w_{l,k}$	The demand of resource k of v_l

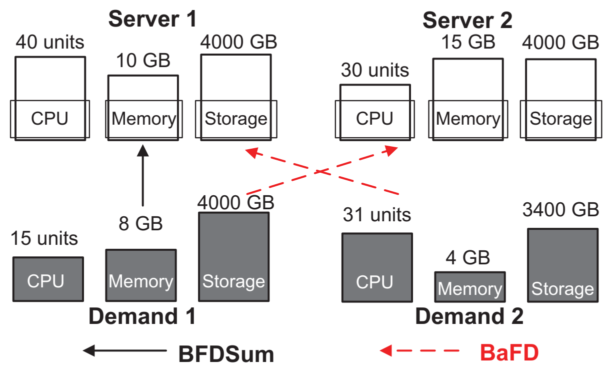


Figure 2. Demand allocation in BFDSum and BaFD.

called *vector bin packing (VBP)* [3], where the servers are conceived as bins and the demands as objects that need to be packed into the bins. Since vector bin packing is NP-hard, MCD is also NP-hard. Due to the hardness of MCD, we cannot find an optimal solution for this problem. Hence, we turn our attention to designing a time-efficient heuristic algorithm for MCD, called Balance Fit Decreasing (BaFD) algorithm. BaFD is improved from the existing algorithm Best Fit Decreasing (BFD) algorithm, a natural heuristic for one-dimensional bin packing problem. BFD orders the objects in decreasing order of size. Starting from the first object, it iterates over the bins, finds out the bin that has the least amount of space left after accommodating the object. It then proceeds to the second object, and repeats the same procedure until all the objects are packed.

As mentioned above, MCD is a generalized form of the bin packing problem, and MCD's problem instance is constrained by more than one dimension and each bin (server) has different capacity vector and each object (demand) has different consumption vector. Hence, we need some generalization of BFD for multiple dimensions. A traditional approach such as BFDSum [17] (or BFDProd) is to map capacity vector into a single scalar (called *volume*) and map consumption vector into a single scalar (called *weight*) using the sum (or product) function, and then perform a one-dimensional BFD algorithm on the volumes and weights. However, a single scalar cannot accurately reflect a server's ability to fit a demand because one type of resource may become the bottleneck of a server which makes other available resources unable to be used. As shown in Fig. 2, the maximum CPU, memory, and storage of all the servers are 40 units, 15GB, and 4000GB, respectively. The normalized capacity vectors of s_1 and s_2 are $[1, 0.6, 1]$ and $[0.75, 1, 1]$, respectively (the three entries in the vector denote CPU, memory, and storage, respectively). The volume of Server1 and Server2 are 2.6 and 2.75, respectively, and the weight of Demand1 and Demand2 are 1.905 and 1.895, respectively. BFDSum allocates Demand1 first since Demand1 has a higher weight than Demand2, and then selects Server1 because it has less volume left after accommodating Demand1 than Server 2. Then, Demand2 cannot be allocated to either Server1 or Server 2. Obviously, a better schedule is to allocate Demand1 to Server 2 and Demand 2 to Server1. BFDSum has worse performance because it ignores that Server2's CPU is the bottleneck for running a demand.

To avoid the bottleneck of one type of resource in each server, BaFD aims to balance the resource utilizations of different resources in each server to increase the server's "volume" for allocating more demands. To this end, in each iteration, say the n^{th} iteration, BaFD attempts to minimize the variance of the allocated resources of the selected server, denoted by $s^{(n)}$,

$$\min\ \text{Var}(\mathbf{u}^{(n)}) = \sum_k \left(u_k^{(n)} - \overline{u^{(n)}}\right)^2, \qquad (14)$$

where $u_k^{(n)}$ represents the utilization of type-k resource of server $s^{(n)}$, $\overline{u^{(n)}}$ represents the mean value of $u_1^{(n)}$, ..., $u_K^{(n)}$, and $\mathbf{u}^{(n)} = [u_1^{(n)}, ..., w_K^{(n)}]$. Accordingly, when selecting an object for a bin, BaFD tries to place the object (demand) that can balance the resource of a given bin (server).

Before introducing the details of BaFD, we first present some definitions. Let $V_{\text{a}}^{(n)}$ and $V_{\text{u}}^{(n)}$ represent the set of demands allocated in the n^{th} iteration and the set of demands unallocated after the n^{th} iteration, respectively. We define the *efficiency* of a server s_i, denoted by $e(s_i)$, by the ratio of the sum of the weights of all VMs placed in this server to its energy cost: $e(s_i) = \frac{\sum_{v_l \in s_i} \text{w}(v_l)}{a_i}$. A higher efficiency of a server means it can support more VM resource consumption per unit energy cost. To minimize the total energy cost when allocating a given group of VMs, BaFD tries to find the servers with higher efficiency values.

The basic idea of BaFD is to iteratively find the "best fit" demands that make each server's resource utilization be most balanced (according to Equ. (14)), and then picks up the server (along with the demand allocation) with the highest efficiency among all servers. Here, we pick up the server with the highest efficiency, because our goal is to minimize the total server cost, thus in each iteration we try to accommodate the demands by the server that has the smallest cost per unit weight. To find the "best fit" demands for a given server, BaFD iteratively picks up the demand that leads to the highest efficiency of the server until its remaining resources cannot hold any existing unallocated demand. In each iteration, there are two parts. Part 1 temporarily determines the VMs that can be allocated to each server to fully utilize its resources. Part 2 chooses the server that leads to the highest efficiency value for the actual demand allocation.

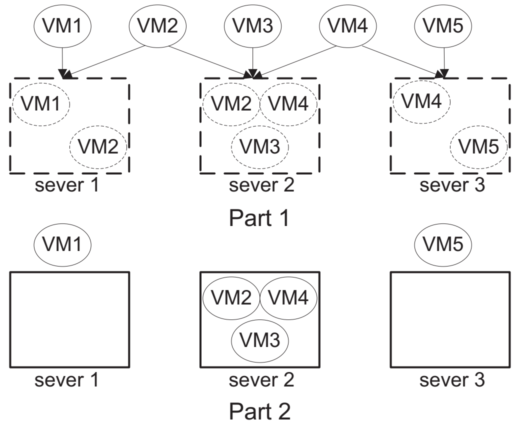


Figure 3. An example of the BaFD algorithm.

Part 1 (line 5-10). For each server s_i, iteratively choose the demand from $V_{\text{u}}^{(n)}$ that best balances s_i's resource utilization by Equ. (14) and temporarily allocate it to s_i. Repeat this process until no demand can be fitted into s_i due to its resource limit. Go to Part 2.
Part 2 (line 11-16). Among all the servers with the temporary demand allocation in Part 1, select the server s_i with the highest efficiency $e(s_i)$. Then, s_i is selected as $s^{(n)}$ and $V_{\text{a}}^{(n)}$ equals all the demands fit into s_i in Part 1. Remove all the demands in $V_{\text{a}}^{(n)}$ from $V_{\text{u}}^{(n)}$. Go to the next iteration.

As Fig. 3 shows: in part 1, Demand1 and Demand2 are iteratively selected among all demands that lead to the highest efficiency of server 1 in each step, so they are put into server 1. Similarly, Demand2, Demand3, and Demand4 are put into server 2; Demand4 and Demand5 are put into server 3. In part 2, we find that server 2 has the highest efficiency among all servers. Then, Demand2, Demand3, and Demand4 are allocated to server 2, while Demand1 and Demand5 remain unallocated.

B. Approximation Algorithm

The greedy algorithm can efficiently get a suboptimal solution for MCD, but it has no performance guarantee for its solution, i.e., how the worst result can be compared

Algorithm 1: Pseudocode of the BaFD algorithm.

```
// Initialization setup
Initialize $V_{\mathrm{a}}^{(n)} \leftarrow \phi$ and $V_{\mathrm{u}}^{(n)} \leftarrow V$, $n = 1$;
while there are unallocated demands do
    for each cloud $c_i$ do
        Pick one server $s_i$;
        while $s_i$ still has enough resource for $v_l \in V_{\mathrm{u}}^{(n)}$ do
            $j \leftarrow \arg\max_{v_l \in V_{\mathrm{u}}^{(n)}} \mathrm{Var}(\mathbf{u}^{(n)})$;
            Put $v_j$ into $s_i$;
    $j \leftarrow \arg\max_i e(s_i)$;
    $s^{(n)} \leftarrow s_j$ // Select $s_j$ as the $n$th server;
    $V_a^{(n)} \leftarrow$ VMs in $s_j$;
    Remove all the demands in $V_{\mathrm{a}}^{(n)}$ from $V_{\mathrm{u}}^{(n)}$;
    $n+ = 1$;
```

to the optimal solution. Hence, in this section, we devise an approximation algorithm using LP-relaxation [5]. LP-relaxation is a technique that relaxes an NP-hard LIP into a related LP problem that is solvable in polynomial time, and the solution to the relaxed LP can be used to gain the solution to the original ILP. To be more specific, first, we get a relax-version of MCD, called MCD-relaxation (MCD-RL) by relaxing the feasible region of MCD's solution from integers ($[\mathbf{x}, \mathbf{y}] \in \{0,1\}^{N+MN}$) to real numbers ($[\mathbf{x}, \mathbf{y}] \in [0,1]^{N+MN}$). Then, MCD-RL becomes an LP problem, which can be solved efficiently by simplex method [10]. By *combining* and *rounding* the solution of MCD-RL, we get the solution of MCD. We also show that the solution of MCD-RL has a constant approximation ratio to the solution of the MCD problem? (Theorem 3.1).

1) MCD-Relaxation and its Rounded Solution: In this part, we propose our centralized approximation algorithm.

Definition 3.1: *MCD-RL* is defined as MCD such that Constraints (4) and (4) in MCD are relaxed to (1) $x_i \in [0,1]\ \forall i$ and (2) $y_{i,l} \in [0,1]\ \forall i, l$, respectively.

In the following, we use $[\overline{\mathbf{x}}, \overline{\mathbf{y}}]$, $[\hat{\mathbf{x}}, \hat{\mathbf{y}}]$, and $[\mathbf{x}^*, \mathbf{y}^*]$ to distinguish the optimal solution of MCD-RL, the rounded solution of MCD-RL, and the optimal solution of MCD. Since MCD-RL is an LP problem, using simplex method, we can get MCD-RL's optimal solution $[\overline{\mathbf{x}}, \overline{\mathbf{y}}]$. Note that $[\overline{\mathbf{x}}, \overline{\mathbf{y}}]$ is not necessarily integral. Since the feasible region of MCD-RL is larger than the feasible region of MCD, we have

$$\sum_i a_i \overline{x}_i \le \sum_i a_i x_i^*. \tag{15}$$

The rounded solution $[\hat{\mathbf{x}}, \hat{\mathbf{y}}]$ is derived from $[\overline{\mathbf{x}}, \overline{\mathbf{y}}]$ (the optimal solution of MCD-RL), hence $[\hat{\mathbf{x}}, \hat{\mathbf{y}}]$ is not necessarily optimal for MCD. Therefore, $\sum_i a_i x_i^* \le \sum_i a_i \hat{x}_i$. Accordingly, we have the following relationship among $[\overline{\mathbf{x}}, \overline{\mathbf{y}}]$, $[\hat{\mathbf{x}}, \hat{\mathbf{y}}]$, and $[\mathbf{x}^*, \mathbf{y}^*]$:

$$\sum_i a_i \overline{x}_i \le \sum_i a_i x_i^* \le \sum_i a_i \hat{x}_i. \tag{16}$$

Now, we turn our attention to getting the rounded solution $[\hat{\mathbf{x}}, \hat{\mathbf{y}}]$ from $[\overline{\mathbf{x}}, \overline{\mathbf{y}}]$. Rounding $\overline{\mathbf{y}}$ is straightforward: for each vector $\overline{\mathbf{y}}_l = [\overline{y}_{1,l},...,\overline{y}_{N,l}]$, we set each $\hat{y}_{i,l}$ by 1 if $\overline{y}_{i,l} = \max\{\overline{y}_{1,l}, ..., \overline{y}_{N,l}\}$, or 0 elsewhere [5]. We also need to update $\overline{\mathbf{x}}$ to $\tilde{\mathbf{x}}$ to ensure that $\sum_l \hat{y}_{i,l} w_{l,k} - \tilde{x}_i b_{i,k} \le 0, \forall i, k$ is satisfied (Constraint (2) in MCD). Thus, we update $\overline{\mathbf{x}}$ to $\tilde{\mathbf{x}}$ by

$$\tilde{x}_i = \max_k \left\{ \frac{\sum_l \hat{y}_{i,l} w_{l,k}}{b_{i,k}} \right\}. \tag{17}$$

What remains to be done is to round the entries in $\tilde{\mathbf{x}}$ to generate $\hat{\mathbf{x}}$. A typical approach to get $\hat{\mathbf{x}}$ is directly rounding up $\tilde{\mathbf{x}}$. For example, if $\tilde{x}_1 = 0.3$, then $\hat{x}_1 = \lceil 0.3 \rceil = 1$. However, directly rounding up $\tilde{\mathbf{x}}$ may be wasteful if each $\tilde{x}_i$ is much smaller than 1. Consider the following scenario: s_1 and s_2 have the same capacity vector and cost a_1, and $\tilde{x}_1 = 0.3$ and $\tilde{x}_2 = 0.4$. Then, by up rounding, we get $\hat{x}_1 = \lceil 0.3 \rceil = 1$ and $\hat{x}_2 = \lceil 0.4 \rceil = 1$, which implies the cost is $2a_1$. However, since $\tilde{x}_1 + \tilde{x}_2 = 0.7 < 1$, we also combine the demands of s_2 and s_1, and put the demands into s_1, which implies the cost is a_1, better than directly rounding up. Hence, combing entries in $\tilde{\mathbf{x}}$ can further decrease the cost than directly rounding up. Note that if s_i and s_j are two different types of servers, we cannot combine $\tilde{x}_i$ and $\tilde{x}_j$ because the combined value cannot reflect the resource utilization of either s_i or s_j if we put all demands in one of them (Constraint (2) in MCD). Thus, the combination can only be executed among the same type of servers. Therefore, we first partition $\tilde{\mathbf{x}}$ into a set of subvectors $\tilde{\mathbf{x}}_1, ..., \tilde{\mathbf{x}}_L$, such that the servers corresponding to the entries in each $\tilde{\mathbf{x}}_n$ $(n = 1, 2, ..., L)$ are from the same cloud provider c_n, i.e., have the same vector capacity and cost. Then, in each $\tilde{\mathbf{x}}_n$, we combine as more non-zero entries as possible with the condition that the sum of the combined entries does not exceed 1. The combining process can also be considered as putting a set of objects with size ranging from $[0, 1]$ into minimum number of bins with size 1, which is a typical *bin packing* problem, and can be solved by BFD efficiently. After combining the entries in $\tilde{x}$, we get a new vector $\mathbf{x}' = [x_1', ..., x_N']$. Consequently, we round up each entry in $\mathbf{x}'$ to get our rounded solution $\hat{x} = [\hat{x}_1, ..., \hat{x}_N]$, in which $\hat{x}_i = \lceil x_i' \rceil \quad 1 \le i \le N$.

Lemma 3.1: $\sum_{s_i \in S_n} \hat{x}_i < 2 \sum_{s_i \in S_n} \tilde{x}_i + 1$, where S_n denotes the set of all the servers in c_n.

Proof: First, we claim that no pair x_i' and x_j' s.t. $x_i' \le \frac{1}{2}$ and $x_j' \le \frac{1}{2}$ exist in $\mathbf{x}'$; otherwise we can combine x_i' and x_j'. For each $x_i' > \frac{1}{2}$, we have $\hat{x}_i = \lceil x_i' \rceil < 2x_i'$. For each $\tilde{\mathbf{x}}_n$, combining its entries does not change the sum of the entries, i.e., $\sum_{s_i \in S_n} x'_i = \sum_{s_i \in S_n} \tilde{x}_i$. When there is no $x'_i \geq \frac{1}{2}$

$$\begin{aligned}\sum_{s_i \in S_n} \hat{x}_i &= \sum_{s_i \in S_n} \lceil x'_i \rceil &(18)\\ &< 2 \sum_{s_i \in S_n} x'_i &(19)\\ &= 2 \sum_{s_i \in S_n} \tilde{x}_i. &(20)\end{aligned}$$

When there exists $x'_j \geq \frac{1}{2}$, then

$$\begin{aligned}\sum_{s_i \in S_n} \hat{x}_i &= \sum_{s_i \in S_n} \lceil x'_i \rceil &(21)\\ &< 2 \sum_{s_i \in S_n \setminus s_j} x'_i + \lceil x'_j \rceil &(22)\\ &= 2 \sum_{s_i \in S_n \setminus s_j} \tilde{x}_i + 1 &(23)\\ &< 2 \sum_{s_i \in S_n} \tilde{x}_i + 1. &(24)\end{aligned}$$

■

Theorem 3.1: The approximation algorithm achieves a constant approximation ratio.

Proof: First, by Lemma 3.1

$$\begin{aligned}\sum_i a_i \hat{x}_i &= \sum_n \left(a_n \sum_{s_i \in S_n} \hat{x}_i \right) &(25)\\ &< \sum_n \left(a_n \left(2 \sum_{s_i \in S_n} \tilde{x}_i + 1 \right) \right) &(26)\\ &= 2 \sum_i a_i \tilde{x}_i + C &(27)\end{aligned}$$

where $C = \sum_n a_n$ is a constant. Also, based on Equ. (17), we can derive that

$$\begin{aligned}\sum_i a_i \tilde{x}_i &= \sum_i a_i \max_k \left\{ \frac{\sum_l \hat{y}_{i,l} w_{l,k}}{b_{i,k}} \right\}\\ &\leq \sum_i a_i \sum_l \delta_{i,l} \hat{y}_{i,l} \quad (\delta_{i,l} = \max_k \left\{ \frac{w_{l,k}}{b_{i,k}} \right\})\\ &\leq \sum_l \beta_l \sum_i \hat{y}_{i,l} \quad (\beta_l = \max_i \{a_i \delta_{i,l}\})\\ &\leq \Delta \sum_i a_i \sum_l \frac{\overline{y}_{i,l} w_{l,1}}{b_{i,1}} \ (\Delta = \max_{i,l,k} \left\{ \frac{b_{i,1} \beta_l}{w_{l,1} a_i} \right\})\\ &\leq \Delta \sum_i a_i \overline{x}_i &(28)\end{aligned}$$

Based on Equ. (25), when $\sum_i a_i \tilde{x}_i$ is large, $\sum_i a_i \hat{x}_i / \sum_i a_i \tilde{x}_i$ is asymptotically approximate to 2, and based on Equ. (25) and Equ. (28), then we get $\sum_i a_i \hat{x}_i < 2 \sum_i a_i \tilde{x}_i \leq 2\Delta \sum_i a_i \overline{x}_i$. Consequently, from $\sum_i a_i \overline{x}_i \leq \sum_i a_i x^*_i \leq \sum_i a_i \hat{x}_i$ (Equ. (16)), we can derive that the approximation ratio is upper bounded by 2Δ, which is a constant. ■

IV. Performance Evaluation

In this section, we conducted both simulation and real-world experiments (on Amazon EC2 [1]) driven by the Google Cluster [7] real trace. The Google Cluster trace records the CPU and memory resource utilization on a cluster of about 11000 VMs from May 2011 for 29 days in every 10 seconds. In Google Cluster trace, the capacity of the CPU and memory of all the servers are not provided [7].

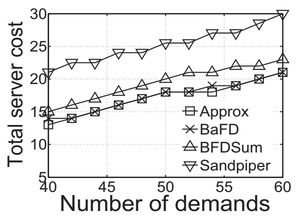
We evaluated the effectiveness of our demand allocation algorithms (BaFD and the approximation algorithm (or Approx)) in comparison with two typical demand allocation algorithms, BFDSum [17] and Sandpiper [24]. In Sandpiper, all servers' capacity vectors and demands' consumption vectors are mapped into singular scales, called volumes and weights, respectively. More specifically, Sandpiper calculates each server's volume and each demand's weight by $\text{Vol}_i = \prod_k \frac{1}{1-b_{i,k}}$ and $\text{Wei}_l = \prod_k \frac{1}{1-w_{l,k}}$, respectively.

We observed that the memory requirements from demands do not exceed servers' memory capacities in our experiments. We set two types of servers and each types has 50 servers. The first type of servers have the same CPU and memory capacity as in the traces. The memory capacity, CPU capacity and energy cost of the second type of servers are 1, 1.2 times and 1.1 times of those of the first type, respectively. We normalize the CPU capacity, memory capacity and energy cost of the first type of server to 1, and hence the second type of server have CPU capacity, memory capacity and energy cost equal to 1.2, 1, and 1.1, respectively. The metrics we measured include:

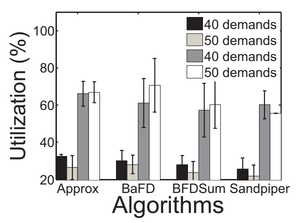
In the Google Cluster trace, the CPU and memory consumption fluctuate over time, which implies that the resource requirement for each VM should be appropriately predetermined for demand allocation so that the real requirements can be satisfied but are close to the determined requirements most of the time. As Service-Level Agreement (SLA) usually specifies a high probability that the real demands must be satisfied, we aim to satisfy all the demands with a high probability (i.e., above 95%). Here, we use the following method [15] to determine the consumption vector. Let $\{w^t_{l,k} | t = 1, ..., T\}$ be the trace of type-k resource of demand v_l, then $w_{l,k}$ is given by: $w_{l,k} = \text{E}(w^t_{l,k}) + \eta\sigma(w^t_{l,k})$, where $\text{E}(w^t_{l,k})$ and $\sigma(w^t_{l,k})$ represent the expectation and the standard deviation of $\{w^t_{l,k}\}$, respectively, and η is a coefficient to determine the percentage of trace data in $\{w^t_{l,k}\}$ that is lower than $w_{l,k}$. If $\eta = 1.28$, then about 80% trace data in $\{w^t_{l,k}\}$ is lower than $w_{l,k}$ [15], which means we set SLA to 80% demand satisfaction [15]. In all our experiments below, all algorithms achieve no less than 95% demand satisfaction.

A. Trace-driven Simulation

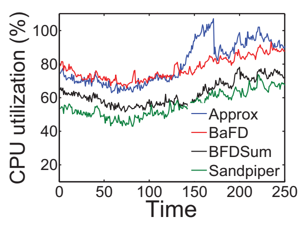
Fig 4(a) shows the total server cost when the number of demands that are required to allocated to servers was varied from 40 to 60 with 2 increase in each step. We see that



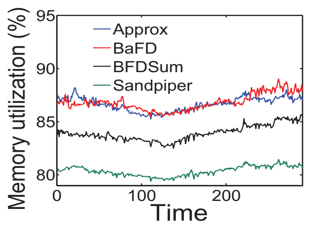
(a) Total cost of servers



(b) CPU utilization



(c) CPU utilization vs. time



(d) Mem utilization vs. time

Figure 4. Comparison of server cost, CPU utilization and memory utilization (simulation)

the result follows Sandpiper>BFDSum> BaFD $\approx$ Approx. Recall that, when selecting a server, BaFD jointly takes into account server cost and the balance of the utilization of different resources, while BFDSum and Sandpiper simply map all the capacity vectors and consumption vectors to single scalars without considering the balance. As we have analyzed in Section III-A, balancing the resource utilization for each server can increase each server's remaining capacity for allocating more demands. As for Approx, it sets the total costs of all used servers as its objective function, implying that Approx aims to search the demand allocation such that the total server cost is minimized. To achieve this goal, Approx first calculates the optimal solution of the relaxed problem, and then rounds the optimal solution to integers, which is still near to the optimal. Since BaFD and Approx use less energy cost and fewer servers to support a given amount of demands, we are interested in checking if they generate many server overload occurrences.

We measured the CPU and memory utilizations every 10 seconds for all servers. Fig 4(b) shows the median, 5th percentile and 95th percentile of these CPU and memory utilizations, respectively, of the four algorithms with 40 and 50 demands. The utilization data is collected from all the servers at each time slot. In both figures, we observe that the median utilization follows: Sandpiper $\approx$ BFDSum $<$ BaFD $\approx$ Approx, which indicates our two algorithms can more fully utilize the resources of servers and hence save the energy cost. The reason is the same as in Fig 4(a). We also compare the CPU and memory utilization of a randomly selected server under different algorithms over time in Fig 4(c) and Fig 4(d), respectively. Comparing these two figures, we find that for bot CPU and memory resource utilizations, Sandpiper $\approx$ BFDSum $<$ BaFD $\approx$ Approx, which is consistent with the results in Fig. 4(b) due to the same reasons.

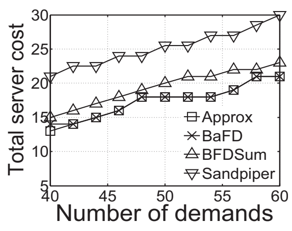
B. Trace-driven Real-world Experiments on Amazon EC2

In this section, we conducted trace-driven experiments on Amazon EC2 [1], which is a web service that provides resizable computing capacity in the cloud [1]. In the simulation, we calculated each server's utilization by summing all the traces' resource consumption in the server. However, in reality, the resource utilization of a server is not simply the linear combination of the programs' load in the server. Hence, in this part, we ran real programs in each server and observe the servers' resource utilization, which more practically reflects the performance of our methods. To generate CPU and memory load of these programs in the servers in Amazon EC2, we use a generator to read the computation utilization data from the trace (Google Cluster trace) every 10 seconds. During each CPU spinning interval (e.g. 10 seconds), the generator generates approximately the same CPU utilization with the value of trace data (e.g. 80%). In the following, we measure the performance of the four algorithms implemented in Amazon EC2 with the same metrics from the simulation in Section IV-A, and then compare the results with the simulation results in Section IV-A.

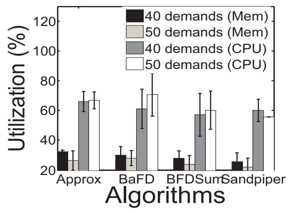
Fig. 5 shows the performance of the four algorithms implemented in Amazon EC2 using Google Cluster trace. Comparing Fig. 4 and Fig. 5, we have the following observations: (1) in both Fig. 4(a) and Fig. 5(a), the total cost of servers follow: Sandpiper $\approx$ BFDSum $>$ BaFD $\approx$ Approx, and (2) in all Fig. 4(b)-(d) and Fig. 5(b)-(d), both CPU utilization and memory utilization follow Sandpiper $\approx$ BFDSum $<$ BaFD $\approx$ Approx. The results in Fig. 5 demonstrate that BaFD and Approx more fully utilize resources in each server and hence save energy cost in Amazon EC2.

C. Comparison of Different Pricing Policies

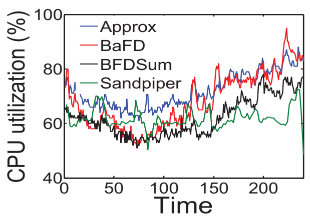
Recall that we proved in Section II that in order to incentivize a CSB to minimize the total server cost for cloud providers, a cloud provider must set the price of each server to be proportional to its cost of the total server cost. We then measure the effectiveness of the pricing policies on the incentives. We compare the total server cost of three different strictly concave (non-linear) pricing functions $L(x) = x + \beta \log(1 + x)$ with our linear pricing function $L(x) = \beta x$ ($\beta = 2, 3, 4$) in Fig. 6(a) and Fig. 6(b) using two traces respectively. We find that the linear pricing function generates the least total server cost and the cost remains nearly the same regardless of the β. It is because that as long as the pricing function is linear, CSB always set its



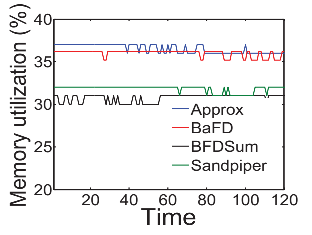
(a) Total cost of servers



(b) CPU utilization



(c) CPU utilization vs. time



(d) Mem utilization vs. time

Figure 5. Experiment on Amazon EC2.

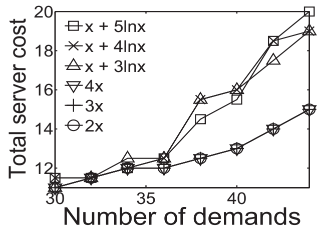


Figure 6. Comparison of total cost of servers for different pricing policies.

goal to minimize the cost of cloud providers (Theorem 2.1), and hence achieve the same demand allocation and the same total energy cost of servers. We also observe that for total cost of servers, $L(x) = x + 5\log(1 + x)$ is better than $L(x) = x + 4\log(1 + x)$, which is better than $L(x) = x + 3\log(1 + x)$. The pricing policy effect the total server cost that the CSB uses because that the pricing function determines the objective function of CSB (Equ. 1) in the MCP problem in Section II, which further determines the allocation strategy of the CSB. The experimental results in Fig 6(a) and Fig. 6(b) confirm the effectiveness of our pricing policy to incentivize CSBs to be cooperative in minimizing the energy cost while satisfy tenants' demands.

V. Related Work

There have been rich literatures studying how to allocate multiple demands into fewer servers to save energy. For example, commercial products such as the VMware v-Sphere Distributed Resource Scheduler [4] (DRS), Microsoft System Center Virtual Machine Manager [3] (VMM), and Citirix XenServer [2] offer VM consolidation as their chief functionality. Research on demand allocation has generated several clever heuristics for resource efficiency [17], [20], [24]. Sandpiper [24] enables live migration of VMs from overloaded hosts by taking the product of CPU, memory, and network loads and migrating VMs to servers based on the First Fit Decreasing (FFD) heuristic. Tang *et al.* [20] proposed a demand allocation method that combines CPU and memory consumption into a singular scalar by calculating the ratio of these two metrics. Srikantaiah *et al.* [17] proposed to use Euclidean distance between resource demands and residual capacity as a metric for consolidation, a heuristic analogous to Norm-based Greedy. Lee [13] *et al.* addressed two fundamental issues that are critical to the design and use of VM consolidation heuristics: 1) how resource utilization and performance aggregate when demands are co-hosted, and 2) how resource demands and scarcities that span across different dimensions should be treated. All these approaches map the capacity vector into a single scalar without considering the resource utilization balance of each server. Thus, they neglect the case that one type of resource may become the bottleneck in a server, which prevents from fully utilizing other types of resources.

A number of studies apply auctions policies to price computing resources in a cloud system [21], [23], [25]. Wang *et al.* [21] proposed an auction-style pricing mechanism, which enable users to compete for cloud resources and cloud providers to increase their own benefits. Wang *et al.* [23] modeled a dynamic auction, where bidders request to occupy a VM for more than one period, such that the auction in one decision interval is correlated with that in another period. Niu *et al.* [14] considered a model of cloud bandwidth allocation and pricing when explicit bandwidth reservation is enabled. Shen and Li [16] proposed network bandwidth pricing policies to create a win-win situation, where tenants strive to increase their own benefits in bandwidth sharing, which also increases the utilities of cloud provider and other tenants. Another group of works studied cloud resource scheduling under given pricing strategies [22] [26]. Wang *et al.* [22] studied how a cloud should allocate its resources between the on-demand market and the auction market. Zhang *et al.* [26] proposed a dynamic scheduling and consolidation mechanism that allocate VM resources to each spot market, in which VMs are traded for immediate delivery to maximize the cloud provider's total revenue. Differently, our work jointly models dynamical resource pricing and scheduling.

VI. Conclusions

It is critical for a Cloud Service Broker (CSB) to guarantee the high service level performance for their cloud tenants and meanwhile minimize the total energy cost of clouds for green computing when it strives to maximize its own profit. To this end, our research is driven by two intriguing questions: 1) under what pricing policies of the cloud providers, a CSB is willing to achieve the above objective when trying to maximizing its own profit, and 2) how should a CSB distribute tenants' demands to multiple cloud providers to minimize cloud providers' energy costs and also satisfy all tenants' demands? To answer the first question, we found a pricing policy from cloud providers to the CSBs, such that maximizing a CSB's profit is equivalent to minimizing the energy cost of cloud providers. To answer the second question, we formulated a demand allocation problem, namely MCD, and proved its NP-hardness. We then devised a greedy algorithm and further proposed an approximation algorithm using LP-relaxation, which was proved to have constant performance guarantee. The experimental results demonstrated the superior performance of our algorithms in both energy efficiency and resource utilizations, and the effectiveness of our pricing policy to make CSBs cooperative in achieving the objective. In our future work, we will consider the scenario where tenants' demands change over time and each cloud provider has heterogeneous servers with different capacities and prices. Also, we will discuss how to apply our technique directly between the cloud providers and the tenants, who do not have enough information of cloud providers (i.e., servers' capacity vectors) as CSBs.

VII. Acknowledgement

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, CNS-1249603, and Microsoft Research Faculty Fellowship 8300751

References

[1] Amazon EC2. http://aws.amazon.com/ec2.

[2] Citrix XenServer. http://www.citrix.com/xenserver.

[3] Microsoft Systems Center Virtual Machine Manager. http://www.microsoft.com/systemcenter.

[4] VMware DRS - dynamic scheduling of system resources. http://www.vmware.com/products/vi/vc/drs.html.

[5] M. Bazaraa, H. Sherali, and C. Shetty. Nonlinear programming: Theory and algorithms. *Wiley Interscience*, 2006.

[6] R. Buyya, C. S. Yeo, J. B. S. Venugopal and, and I. Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 2009.

[7] G. cluster data. https://code.google.com/p/googleclusterdata/.

[8] A. C. Compute. http://aws.amazon.com/ec2/hpc- applications/., 2011.

[9] Z. C. Computing. http://www.zimory.com/.

[10] F. S. Hillier. *Linear and Nonlinear Programming*. Stanford University, 2008.

[11] IBM. http://www.ibm.com/cloud-computing/us/en/products/dedicated-bare-metal-servers.html/.

[12] IBM. Make IT Green - Cloud Computing and its Contribution to Climate Change.

[13] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder. Validating heuristics for virtual machines consolidation, 2011.

[14] D. Niu, C. Feng, and B. Li. A theory of cloud bandwidth pricing for video-on-demand providers. In *Proc. of INFOCOM*, 2012.

[15] S. M. Ross. *Introduction to Probability Models, 8th Edition*. Amsterdam: Academic Press, 2003.

[16] H. Shen and Z. Li. New bandwidth sharing and pricing policies to achieve a win-win situation for cloud provider and tenants. In *Proc. of Infocom*, 2014.

[17] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proc. of HotPower*, 2008.

[18] F. TA and R. MG. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 1989.

[19] F. TA and R. MG. Greedy randomized adaptive search procedures. *Journal of global optimization*, 1995.

[20] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *Proc. of WWW*, 2007.

[21] Q. Wang, K. Ren, and X. Meng. When cloud meets ebay: Towards effective pricing for cloud computing. In *Proc. of INFOCOM*, 2012.

[22] W. Wang, B. Li, and B. Liang. Towards optimal capacity segmentation with hybrid cloud pricing. In *Proc. of ICDCS*, 2012.

[23] W. Wang, B. Liang, and B. Li. Revenue maximization with dynamic auctions in iaas cloud markets. In *Proc. of IWQoS*, 2013.

[24] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proc. of NSDI*, 2007.

[25] H. Zhang, B. Li, H. B. Jiang, F. M. Liu, A. V. Vasilakos, and J. C. Liu. A framework for truthful online auctions in cloud computing with heterogeneous user demands. In *Proc. of INFOCOM*, 2013.

[26] Q. Zhang, E. Grses, R. Boutaba, and J. Xiao. Dynamic resource allocation for spot markets in clouds. In *Proc. of Hot-ICE*, 2011.

[27] J. Zhao, H. Li, C. Wu, Z. Li, Z. Zhang, and F. C. Lau. Dynamic pricing and profit maximization for the cloud with geo-distributed data centers. In *Proc. of INFOCOM*, 2009.