This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data

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

Chenxi Qiu, Haiying Shen*, *Senior IEEE Member,* Liuhua Chen

Abstract—Functioning as an intermediary between tenants and cloud providers, cloud service brokerages (CSBs) can bring about great benefits to the cloud market. 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. 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 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 CSBs 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 and a decomposition-based solution 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 and the superior performance of our algorithms in energy efficiency and resource utilization over the previous algorithms.

Keywords—cloud computing, cloud service brokerage, pricing policy, demand allocation.

1 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 [7], [14], [15].

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 [8], [23] 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.

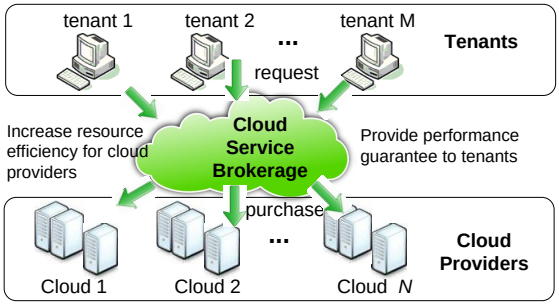


Fig. 1. Cloud service brokerage.

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 [23]. 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 [23]. The electricity consumption of clouds globally is 623 Billion kWh in 2007 and is projected to be 1,963 Billion

- * *Corresponding Author. Email: shenh@clemson.edu. Telephone number: (864) 6565931*
- *Chenxi Qiu is with the College of Information Science and Technology, Pennsylvania State University, State College, PA, 16801. E-mail: czq3@psu.edu.*
  Haiying Shen is with the Department of Computer Science, University of Virginia, Charlottesville, VA, 22904. E-mail: hs6ms@virginia.edu.
  Liuhua Chen is with the Department of Electrical and Computer Engineering, Clemson University, Clemson, SC, 29634. E-mail: liuhuac@clemson.edu

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data

kWh 2020, which will generate 1034 MtCO2e (Gigatonne Carbon Dioxide Equivalent) [19]. 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) [23]. 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 [28], [29]. However, MCD cannot simply be solved using the existing VBP solutions (e.g., Best Fit Decreasing algorithm (BFD) [28], [29]), 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 a decomposition-based algorithm for MCD, called Decomposition-based demand Fit Decreasing (DFD). First, we formulate a relaxation version of MCD, namely MCD-RL. Using Lagrangian dual decomposition, we divide MCD-RL into a set of subproblems, which can be computed in parallel in multiple machines. We prove that the decomposition-based method converges much faster than the centralized method and MCD and MCD-RL have a constant approximation ratio. We summarize our contributions in below. 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 a decomposition-based approximation algorithm (DFD) for a CSB to minimize cloud energy cost when allocating tenant demands to cloud servers, and analyze algorithm performance.
3. We test the performance of BaFD and DFD in comparison with the previous algorithms by both trace-driven experiments on a simulator and on Amazon EC2 [1]. The experimental results that our algorithms outperform the previous methods in the aspects of energy cost of servers and resource utilizations (e.g., CPU utilization and memory utilization).

The remainder of this paper is organized as follows. Section 2 and Section 3 study the pricing problem and demand allocation problem for CSBs, respectively. Section 4 evaluates the performance of our proposed schemes in comparison with other schemes. Section 5 presents related work. Section 6 concludes this paper with remarks on our future work.

2 Pricing Policy for CSBs

Our objective is to minimize the energy consumption of cloud servers. In reality, CSBs are usually profit-driven and they do not have to minimize the energy cost from cloud providers. To maximize its own profit, a selfish CSB may distribute tenants' demands to the clouds that cannot fully utilize the energy resources [23]. On the other hand, even if a CSB is willing to save energy for cloud providers, it has little information of servers' energy cost, which becomes an obstacle for the energy saving.

In this section, we discuss how to design a pricing policy from cloud providers to CSBs to incentivize profit-driven CSBs to minimize the total energy cost for cloud providers, even if CSBs are not aware of the energy cost of servers in clouds. In what follows, we first describe the system model as well as the notations that will be throughout the paper in Section 2.1. Then, we formulate the MCP and MCD problems in Section 2.2. Finally, we describe our pricing policies in Section 2.3.

2.1 System model

We consider a scenario composed of a CSB, M tenant demands $V = \{v_1, ..., v_M\}$, 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). 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[1]. We list the

1. In what follows, we normalize the entries of $\mathbf{w}_l$ and $\mathbf{b}_i$ by dividing each $w_{l,k}$ and $b_{i,k}$ by $w_{\max}$ and $b_{\max}$, respectively, where $w_{\max} = \max_{l,k} w_{l,k}$ and $b_{\max} = \max_{i,k} b_{i,k}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data

main notations that will be used throughout the paper in Table 1.

TABLE 1
Symbols and descriptions

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	Indicating whether s_i is purchased
$y_{i,l}$	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
L_n	The money that the CSB needs to pay to cloud provider n

Note that one of the major causes of energy inefficiency in data centers is the idle power wasted when servers run at low utilization. Even at a very low load, such as 10% CPU utilization, the power consumed is over 50% of the peak power [10]. Similarly, if the disk, network, or any such resource is the performance bottleneck, the idle power wastage in other resources goes up [27].

Since the relationship between server's utilization and energy consumption is too complex to model in a linear form [10], by considering the computational tractability of the approach design and analysis, we simplify the model by assuming that the energy consumption of each server is increased linearly with the increase of the utilization of each type of its resource [16]: $P_i = a_i + \sum_{l \in V_i} \sum_{k=1}^{K} w_{l,k}$, where a_i is the idle power of server i and V_i is the set of demands allocated to the ith server. Here, we need to note that when we sum the energy consumption of all the servers, we obtain

$$\sum_i P_i = \sum_i \left(a_i + \sum_{l \in V_i} \sum_{k=1}^{K} w_{l,k} \right) \quad (1)$$

$$= \sum_i a_i + \sum_i \sum_{l \in V_i} \sum_{k=1}^{K} w_{l,k} \quad (2)$$

$$= \sum_i a_i + \sum_{l=1}^{L} \sum_{k=1}^{K} w_{l,k} \quad (3)$$

where $\sum_{l=1}^{L} \sum_{k=1}^{K} w_{l,k}$ is given and can be consider as a constant. So, in what follows, we mainly consider how to minimize $\sum_i a_i$, and we call a_i the cost of each server s_i. As in [41], we also assume that the servers in each cloud provider are homogeneous. That is, they have the same capacity vector and energy cost. But servers from different cloud providers can be different.

2.2 Problem formulations

We assume that the CSB knows the consumption vectors of all the tenants and the tenants' consumption vectors are all fixed [23]. We use indicator variable x_i to represent whether s_i is purchased by the CSB:

$$x_i = \begin{cases} 1 & s_i \text{ is purchased by the CSB} \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

We use indicator variable $y_{i,l}$ to denote whether demand v_l is distributed to cloud provider c_i:

$$y_{i,l} = \begin{cases} 1 & v_l \text{ is distributed to server } s_i \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

Like [18], in this paper, we assume that 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, i.e., $z_n = \sum_{s_i \in S_n} x_i$. 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 and $L_n(0) = 0$. Cloud providers provide the information of the pricing policy and the capacity vectors of their servers to the CSB when the CSB buys servers [23]. 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 the *Maximum CSB Profit (MCP)* problem, as follows:

$$\min \quad f'(\mathbf{z}) = \sum_n L_n(z_n) \quad (6)$$

$$\text{s.t.} \quad g_{i,k}(\mathbf{x}, \mathbf{y}^i) = \sum_l y_{i,l} w_{l,k} - x_i b_{i,k} \leq 0, \forall i, k \quad (7)$$

$$h_l(\mathbf{y}_l) = \sum_i y_{i,l} - 1 = 0, \forall l \quad (8)$$

$$x_i \in \mathbb{N}, \forall i, \ y_{i,l} \in \{0,1\}, \forall i, l \quad (9)$$

$$z_n = \sum_{s_i \in S_n} x_i, \ \forall n. \quad (10)$$

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 $\mathbf{y}^i = [y_{i,l}, ..., y_{i,M}]$.

In the above problem formulation, constraint (7) ensures that for each cloud provider, the capacities of the purchased servers are enough for its allocated demands. Constraint (8) ensures that each demand is allocated to one server. In general, MCP is an integer non-linear programming problem.

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:

$$\min \quad f(\mathbf{x}) = \sum_i a_i x_i \quad (11)$$

$$\text{s.t.} \quad \text{Constraints } (7), (8), \text{ and } (9) \text{ in MCD.}$$

2.3 Pricing policies

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:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data

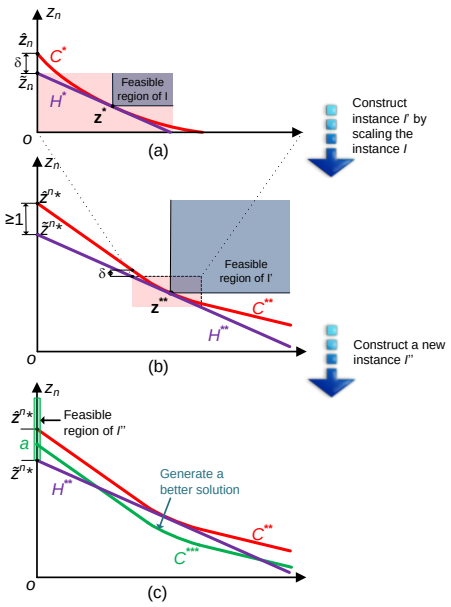


Fig. 2. Proof of Theorem 2.1.

what pricing policies (i.e., $L_n(\cdot)$) make MCP equivalent to MCD?

In what follows, we discuss the necessary and sufficient conditions that make the objective functions of MCD and MCP equivalent. First, it is trivial to find that when $L_n(z) = \beta a_n z_n \ \forall n$ (a_n is the cost of each server from provider n, as we assume the servers from the same provider have the same cost), the two problems are equivalent, since

$$f'(\mathbf{z}) = \sum_n L_n(z_n) = \sum_n \beta a_n z_n \tag{12}$$

$$= \sum_n \beta a_n \sum_{s_i \in S_n} x_i = \beta \sum_i a_i x_i = \beta f(\mathbf{x}), \tag{13}$$

which means they have the same objective functions.

However, it requires more discussion to find the necessary condition, as we need to not only prove that the policies under this condition meet the requirement, but also exclude all other possible pricing policies. In the following, Theorem 2.1 concludes that $L_n(z) = \beta a_n z_n \ \forall n$ is also necessary condition to make the two problems equivalent.

Lemma 2.1: Suppose that MCD and MCP are equivalent. Let $\mathcal{I}$ be an instance of both MCD and MCP, where the optimal point is $\mathbf{z}^*$, and let $H^*(\mathbf{z})$ and $C^*(\mathbf{z})$ be the hyperplane and the hypersurface passing through $\mathbf{z}^* = [z_1^*, ..., z_L^*]$, respectively:

$$H^*(\mathbf{z}): \quad \sum_n a_n z_n - c_A^* = 0 \tag{14}$$

$$C^*(\mathbf{z}): \quad \sum_n L_n(z_n) - c_L^* = 0, \tag{15}$$

where $c_A^* = \sum_n a_n z_n^*$ and $c_L^* = \sum_n L_n(z_n^*)$, then $H^*(\mathbf{z})$ and $C^*(\mathbf{z})$ coincide.

Proof: Since MCP and MCD have the same linear constraints, their feasible regions are the same convex polyhedron. As Fig. 2(a) shows, let each $\tilde{\mathbf{z}}^n = \tilde{z}^n \mathbf{e}^n$ be the point at which H^* intersects with z_n axis, where $\tilde{z}^n = \frac{\sum_i a_i z_i^*}{a_n}$, and $\mathbf{e}^n$ is a N dimension vector with all entries equaling 0 except the n^{th} entry equaling 1. Notice z_n does not have to be an integer. As $\mathbf{z}^*$ is in a hyperplane, $\mathbf{z}^*$ can be represented as a linear combination of $\tilde{\mathbf{z}}^1$, ..., $\tilde{\mathbf{z}}^L$:

$$\mathbf{z}^* = \sum_n \lambda_n \tilde{\mathbf{z}}^n, \tag{16}$$

where $\sum_n \lambda_n = 1$ and each $\lambda_n \geq 0$. Also, $f'(\mathbf{z})$ is concave, we can derive that

$$f'(\mathbf{z}^*) = f'\left(\sum_n \lambda_n \tilde{\mathbf{z}}^n\right) \geq \sum_n \lambda_n f'(\tilde{\mathbf{z}}^n), \tag{17}$$

which implies that $\sum_n \lambda_n f'(\tilde{\mathbf{z}}^n) \leq c_L^*$.

Similarly, let $\hat{\mathbf{z}}^n = \hat{z}^n \mathbf{e}^n$ be the point that C^* intersects with z_n axis ($n = 1, 2, ..., L$), respectively,

$$\hat{z}^n = L_n^{-1}\left(\sum_i L_n(z_i^*)\right). \tag{18}$$

On the other hand,

$$\sum_n \lambda_n f'(\hat{\mathbf{z}}^n) = \sum_n \lambda_n c_L^* = c_L^*. \tag{19}$$

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

Now, we need to prove that $\hat{z}^n = \tilde{z}^n \ \forall n$, which makes C^* a hyperplane and hence completes the proof. For the sake of contradiction, suppose that exists $\hat{z}^n \neq \tilde{z}^n$. Since f' is monotonically increasing, then $\exists \tilde{z}^n < \hat{z}^n$; otherwise, $\sum_n \lambda_n f'(\tilde{\mathbf{z}}^n) > \sum_n \lambda_n f'(\hat{\mathbf{z}}^n)$.

First, we consider moving a point from $\mathbf{z}^*$ towards the z_n axis in both $C^*(\mathbf{z})$ and $H^*(\mathbf{z})$. For simplicity, we let $\mathbf{v}$ denote the direction from $\mathbf{z}^*$ to the z_n axis and let $\bigtriangledown_{\mathbf{v}} f(\mathbf{z})$ denote the gradient of $f(\mathbf{z})$ along $\mathbf{v}$ at point $\mathbf{z}$. Then, $\bigtriangledown_{\mathbf{v}} H^*(\mathbf{z})$ is a constant. As $C^*(\mathbf{z})$ is convex, $C^*(\mathbf{z})$ is monotone. Hence, to make sure $\tilde{z}^n < \hat{z}^n$, $C^*(\mathbf{z})$ is monotonically increasing and $\bigtriangledown_{\mathbf{v}} C^*(\hat{\mathbf{z}}) > \bigtriangledown_{\mathbf{v}} H^*(\tilde{\mathbf{z}})$. We let

$$\delta = \bigtriangledown_{\mathbf{v}} C^*(\hat{\mathbf{z}}^n) - \bigtriangledown_{\mathbf{v}} H^*(\tilde{\mathbf{z}}^n) \tag{20}$$

And then, we construct a new instance $\mathcal{I}'$ by scaling the feasible region of instance $\mathcal{I}$ by $\left\lceil \frac{1}{\delta} \right\rceil$, as Fig. 2(b) shows. Let the optimal solution of both MCD and MCP in $\mathcal{I}'$ be $\mathbf{z}^{**}$, and we can move $H^*(\mathbf{z})$ and $C^*(\mathbf{z})$ from $\mathbf{z}^*$ to $\mathbf{z}^{**}$ to construct two new hypersurfaces $H^{**}(\mathbf{z})$ and $C^{**}(\mathbf{z})$: both of which pass through the optimal point $\mathbf{z}^{**} = [z_1^{**}, ..., z_L^{**}]$:

$$H^{**}(\mathbf{z}): \quad \sum_n a_n z_n - \sum_n a_n z_n^{**} = 0 \tag{21}$$

$$C^{**}(\mathbf{z}): \quad \sum_n L_n(z_n) - \sum_n L_n(z_n^{**}) = 0. \tag{22}$$

Let each $\tilde{\mathbf{z}}^{n*} = \tilde{z}^{n*} \mathbf{e}^n$ and $\hat{\mathbf{z}}^{n*} = \hat{z}^{n*} \mathbf{e}^n$ be the points at which H^{**} and C^{**} intersect with z_n axis, respectively. Note that the direction from $\mathbf{z}^{**}$ to the z_n axis is still $\mathbf{v}$.

Similar to $C^*(\mathbf{z})$ and $H^*(\mathbf{z})$, we can infer that $\bigtriangledown_{\mathbf{v}} C^*(\hat{\mathbf{z}})$ is monotonically increasing and $\bigtriangledown_{\mathbf{v}} H^*(\tilde{\mathbf{z}})$ is a constant. Then,

$$\hat{\mathbf{z}}^{n*} - \tilde{\mathbf{z}}^{n*} \geq (\bigtriangledown_{\mathbf{v}} C^*(\hat{\mathbf{z}}^n) - \bigtriangledown_{\mathbf{v}} H^*(\tilde{\mathbf{z}}^n)) \left(\left\lceil \frac{1}{\delta} \right\rceil - 1\right) + \delta$$

$$= \delta \left(\left\lceil \frac{1}{\delta} \right\rceil - 1\right) + \delta \geq 1, \tag{23}$$

where indicates that there exists an integer, denoted by a, between $\hat{z}^{n*}$ and $\tilde{z}^{n*}$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data

Finally, we construct a new instance $\mathcal{I}''$ that only has the feasible points in the z_n axis with $z_n \geq a$, as Fig. 2(c) shows. Then, the optimal solution in MCP is $a \cdot \mathbf{e}^n$, but the optimal solution of MCD is $\tilde{\mathbf{z}}^{n*}$. Because MCD and MCP have different optimal solutions in $\mathcal{I}''$, it contradicts the assumption that MCP and MCD have the same optimal solution in all instances. □

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

Proof: Let $\mathcal{I}$ be an instance of both MCD and MCP, where the optimal point is $\mathbf{z}^*$, then hyperplane $H^*(\mathbf{z})$ and the hypersurface $C^*(\mathbf{z})$ (defined by Equ. (14) and Equ. (15)) coincide, indicating that $C^*(\mathbf{z})$ is a hyperplane:

$$C^*(\mathbf{z}) : \sum_n L_n(z_n) - c_L^* = \sum_n a'_n z_n - c_L^* = 0, \tag{24}$$

where each $a'_n = \frac{c_L^*}{c_A^*} a_n$, which implies that

$$\sum_n L_n(z_n) = \sum_n a'_n z_n \quad \forall \mathbf{z} \in C^*(\mathbf{z}). \tag{25}$$

For each n, let each $z_i = 0$ if $i \neq n$. Then, from Equ. (25) we can obtain

$$L_n(z_n) = \frac{c_L^*}{c_A^*} a_n z_n = \beta a_n z_n. \tag{26}$$

The proof completes. □

Theorem 2.1 suggests that: *to encourage a profit-driven CSB to minimize the total energy cost of multiple clouds, the price of each server should be proportional to its energy cost; If a pricing policy is not proportional to the energy cost, it cannot always 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 [7]. 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.

3 Min-energy CSB Demand Allocation

In this section, we address Q2 in Section 1, i.e., how a CSB distributes its tenants' requests to achieve the objective of minimizing the energy cost. As opposed to many existing works that target homogeneous servers (i.e., in terms of both capacity and cost), our demand allocation strategy considers the heterogeneous servers across multiple providers. We note that the capacity of different servers can be different (e.g., servers can be characterized with "higher CPU capacity" or "higher memory capacity"), and the selection of servers highly depends on the demands' requirement for different types of resources. The problem is an extension of well-known bin-packing problem, which is NP-hard.

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 [28], [29], and an approximation algorithm through linear programming (LP) relaxation and Lagrangian dual decomposition [6].

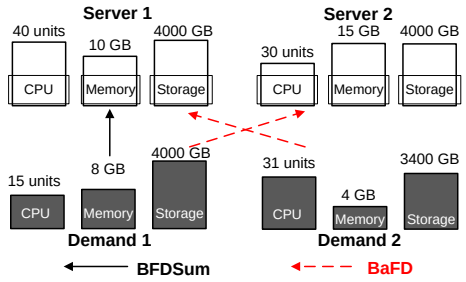


Fig. 3. Demand allocation in BFDSum and BaFD.

3.1 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 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 [27] (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. We use a BFD-based algorithm, BFDSum ([27]), as an example to illustrate this problem. In BFDSum, the volume of a server is the sum of all entries in the capacity vector, and the weight of a demand is the sum of all entries in the consumption vector. Before fitting the objects into bins, BFDSum first needs to map each capacity vector and consumption vector into single scalars (volume and weight) by summing all the entries in each vector. After that, BFDSum follows the same procedure of BFD algorithm based on the calculated volumes and weights. As shown in Fig. 3, the maximum CPU, memory, and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data

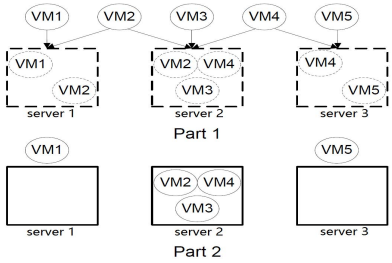


Fig. 4. An example of the BaFD algorithm.

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 $s^{(n)}$,

$$\min \ \mathrm{Var}(\mathbf{u}^{(n)}) = \sum_k \left(u_k^{(n)} - \overline{u^{(n)}}\right)^2, \tag{27}$$

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 $\mathbf{u}^{(n)} = [u_1^{(n)}, ..., u_K^{(n)}]$. 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:

$$e(s_i) = \frac{\sum_{v_l \in s_i} \mathrm{w}(v_l)}{a_i}. \tag{28}$$

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. (27)), 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.

Algorithm 1: Pseudocode of the BaFD algorithm.

```
1  // Initialization setup
2  Initialize V_a^(n) ← φ and V_u^(n) ← V, n = 1;
3  // Main step
4  while there are unallocated demands do
5      // Part 1
6      for each cloud c_i do
7          Pick one server s_i;
8          while s_i still has enough resource for v_l ∈ V_u^(n) do
9              j ← arg max_{v_l ∈ V_u^(n)} Var(u^(n));
10             Put v_j into s_i;
11     // Part 2
12     j ← arg max_i e(s_i);
13     s^(n) ← s_j // Select s_j as the n^th server;
14     V_a^(n) ← VMs in s_j;
15     Remove all the demands in V_a^(n) from V_u^(n);
16     n+ = 1;
```

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. (27) 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. 4 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.

Time complexity. We first look at the time complexity of each iteration, composed of part 1 and 2. Part 1 needs to put M demands into servers, which requires calculating Equ. (27) $O(M)$ times. Part 2 needs to picks up the server with the highest efficiency, which has time complexity $O(N)$. Hence, the time complexity of each iteration is $O(M + N)$. Since each iteration allocates at least 1 demand to a server, the algorithm has at most M iteration, implying the time complexity of BaFD is $O(M(M + N))$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data

3.2 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 to the optimal solution. Hence, in this section, we devise an approximation algorithm using LP-relaxation [6]. LP-relaxation is a technique that relaxes an NP-hard ILP 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 [17]. 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 3.2). For high algorithm scalability, we further propose a decomposition-based algorithm, called Decomposition-based demand Fit Decreasing (DFD) for MCD, which can be implemented in multiple machines in parrell. Finally, we prove that the decomposition-based algorithm also has an approximation ratio of factor 2 (Theorem 3.3).

3.2.1 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 (9) in MCD is 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

$$\sum_i a_i \overline{x}_i \leq \sum_i a_i x_i^*. \tag{29}$$

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, $\sum_i a_i x_i^* \leq \sum_i a_i \hat{x}_i$. Accordingly, we have the following relationship among $[\overline{\mathbf{x}},\overline{\mathbf{y}}]$, $[\hat{\mathbf{x}},\hat{\mathbf{y}}]$, and $[\mathbf{x}^*,\mathbf{y}^*]$:

Proposition 3.1: $[\overline{\mathbf{x}},\overline{\mathbf{y}}]$ and $[\hat{\mathbf{x}},\hat{\mathbf{y}}]$ provide a lower bound and an upper bound of the optimal solution $[\mathbf{x}^*,\mathbf{y}^*]$, respectively:

$$f(\overline{\mathbf{x}}) \leq f(\hat{\mathbf{x}}) \leq f(\mathbf{x}^*). \tag{30}$$

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 [6]. We also need to update $\overline{\mathbf{x}}$ to $\tilde{\mathbf{x}}$ to ensure that $\sum_l \hat{y}_{i,l} w_{l,k} - \tilde{x}_i b_{i,k} \leq 0, \forall i,k$ is satisfied (Constraint (7) in MCD). Thus, we update $\overline{\mathbf{x}}$ to $\tilde{\mathbf{x}}$ by

$$\tilde{x}_i = \max_k \left\{ \frac{\sum_l \hat{y}_{i,l} w_{l,k}}{b_{i,k}} \right\}. \tag{31}$$

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 (7) 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 $\hat{x}_i = \lceil x'_i \rceil \quad 1 \leq i \leq N$.

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: The detailed proof is Appendix. □

Theorem 3.2: The approximation algorithm achieves a constant approximation ratio.

Proof: Detailed proof can be found in the appendix. □

3.3 Decomposition-based Fit Demand Algorithm

As a huge number of tenants are active in the system, the approximation algorithm introduced in Section 3.2 is not time-efficient. Considering the scalability of our system, we need to find a way to realize the algorithm in a more efficient way. In this section, we propose a decomposition-based algorithm for MCD-RL, called Decomposition-based demand Fit Decreasing algorithm (DFD). It relies on the Lagrangian decomposition, which is a classical method in combinatorial optimization and is widely applied to distributed and parallel computation [6].

From our observation, MCD-RL's dual problem is easier to decompose than MCD-RL itself. Hence, we first derive the dual problem, called dual MCD-RL (or DMCD-RL for short) and then decompose DMCD-RL. We use

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data

the technique called *Lagrangian dual decomposition*, which is a classical method in combinatorial optimization and has been widely applied to distributed and parallel computation [6]. After calculating the optimal solution of MCD-RL, DFD rounds this solution to form a solution of MCD, like the approximation algorithm in Section 3.2.1 does. To define DMCD-RL, specifically, we define the Lagrangian function $\Lambda : \mathbb{Z}^N \times \mathbb{Z}^{NM} \times \mathbb{R}^{NM} \times \mathbb{R}^M \to \mathbb{R}$ as follows.

In the function, the vectors $\lambda = [\lambda_{1,1}, \lambda_{1,2}, ..., \lambda_{N,K}]$ and $\nu = [\nu_1, ..., \nu_M]$ are called the Lagrange multiplier vectors ($\lambda \in \mathbb{R}^{NM}$, $\lambda \succeq \mathbf{0}$, and $\nu \in \mathbb{R}^M$) associated with MCD-RL:

$$\begin{aligned} & \Lambda(\mathbf{x}, \mathbf{y}, \lambda, \nu) \\ + & \sum_i \sum_k \lambda_{i,k} \left(\sum_l y_{i,l} w_{l,k} - x_i b_{i,k} \right) \\ = & \sum_i \left(a_i - \sum_k \lambda_{i,k} b_{i,k} \right) x_i \ [1] \\ + & \sum_l \left(\sum_i \left(\nu_l + \sum_k \lambda_{i,k} w_{l,k} \right) y_{i,l} - \nu_l \right) [2] \end{aligned} \quad (32)$$

Then, we define the Lagrangian dual function by

$$\Theta(\lambda, \nu) = \inf_{[\mathbf{x},\mathbf{y}] \in \mathcal{D}} \Lambda(\mathbf{x}, \mathbf{y}, \lambda, \nu). \quad (33)$$

where $\mathcal{D} = \{[\mathbf{x}, \mathbf{y}] : \mathbf{x} \in \mathbb{Z}^N \text{ and } \mathbf{y} \in \{0,1\}^{NM}\}$. Therefore, DMCD-RL is to find

$$\max \ \Theta(\lambda, \nu) \text{ s.t. } \lambda \succeq \mathbf{0} \quad (34)$$

The dual function $\Theta(\lambda, \nu)$ is always concave, because it is a point-wise infimum of affine functions [6]. Hence, the Lagrangian dual problem is a convex problem [6].

Lemma 3.2: Let $\mathcal{S}$ be the feasible region of MCD. Since MCD-RL is a convex problem, based on the *Strong duality theorem* [6], we can get

$$\inf \{ \ f(\mathbf{x}) : [\mathbf{x}, \mathbf{y}] \in \mathcal{S} \ \} = \sup \left\{ \begin{array}{c} \Theta(\lambda, \nu) : \lambda \in \mathbb{R}^{NM} \\ \lambda \succeq \mathbf{0}, \nu \in \mathbb{R}^M \end{array} \right\}. \quad (35)$$

That is, there is no gap between the optimal solutions of MCD-RL and its dual problem DMCD-RL.

Then, we attempt to find the optimal solution of MCD-RL by finding the optimal solution of DMCD-RL. To do this, we use the Lagrangian decomposition, which first decomposes the original problem into a set of subproblems, solves each subproblem, and finally combines the results to solve the original problem. We present these two steps below.

3.3.1 DMCD-RL decomposition

Note that the Lagrangian function in Equ. (32) has two parts (indicated by [1] and [2]). Accordingly, we decompose the problem to two parts denoted by $\Lambda_0(\mathbf{x}, \lambda, \nu)$ and $\Lambda_l(\mathbf{y}_l, \lambda, \nu_l)$ ($l = 1, ..., M$), respectively.

$$\Lambda_0(\mathbf{x}, \lambda, \nu) = \sum_i \left(a_i - \lambda_i^{\mathrm{T}} \mathbf{b}_i \right) x_i \quad (36)$$

$$\Lambda_l(\mathbf{y}_l, \lambda, \nu_l) \triangleq \sum_i \left(\nu_l + \lambda_i^{\mathrm{T}} \mathbf{w}_l \right) y_{i,l} - \nu_l \quad (37)$$

where $\lambda_i = [\lambda_{i,1}, ..., \lambda_{i,K}]^{\mathrm{T}}$ and $\mathbf{b}_i = [b_{i,1}, ..., b_{i,K}]^{\mathrm{T}}$, $\mathbf{y}_l = [y_{1,l}, ..., y_{N,l}]^{\mathrm{T}}$ and $\mathbf{w}_l = [w_{l,1}, ..., w_{l,K}]^{\mathrm{T}}$. Then, we define sub-problem functions $\Theta_0(\lambda)$ and $\Theta_l(\lambda, \nu_l)$, respectively.

$$\Theta_0(\lambda) \triangleq \inf_{\mathbf{x} \in \mathcal{D}_0} \{\Lambda_0(\mathbf{x}, \lambda, \nu)\}, \quad (38)$$

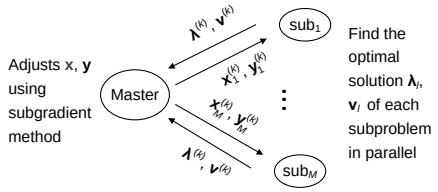


Fig. 5. Decomposition-based Fit Demand algorithm.

and $\Theta_l(\lambda, \nu_l) \triangleq \inf_{\mathbf{y}_l \in \mathcal{D}_l} \{\Lambda_i(\mathbf{y}_l, \lambda, \nu_l)\}$, where $\mathcal{D}_0 = \{\mathbf{x} : \mathbf{x} \in \mathbb{Z}^{NM}\}$ and $\mathcal{D}_l = \{\mathbf{y}_l : \mathbf{y}_l \in \{0,1\}^M\}$.

Algorithm 2: Pseudocode of the decomposition-based subgradient method.

```
input  : ϵ;
output : x̄, ȳ;
// Initialization setup
Select a starting solution: [x^(1), y^(1)] = 0;
Let current incumbent solution [x*, y*] equal [x^(1), y^(1)];
Put k = 1;
// Main step
repeat
    Given x^(k) and y^(k), solve each sub_l in parallel:
    [λ_l^(k), ν_l^(k)] ← Subl(x^(k), y^(k));
    Each sub_l sends λ_l^(k) and ν_l^(k) to Master;
    Given λ^(k) and ν^(k), Master finds a subgradient
    ξ^(k) ∈ ∂Λ(x^(k), y^(k), λ^(k), ν^(k));
    if ξ^(k) = 0 then
        Break;
    else
        d^(k) ← ξ^(k) / ||ξ^(k)||;
        Select a step size α^(k) > 0 and compute
        [x^(k+1), y^(k+1)] ← [x^(k), y^(k)] + α^(k) ξ^(k);
    [x*, y*] ← [x^(k+1), y^(k+1)];
    Master sends x^(k+1) and y^(k+1) to each sub_l;
    k = k + 1;
until ξ^(k) < ϵ;
[x̄, ȳ] = [x^(k), y^(k)];
return x̄, ȳ;
```

Hence, the Lagrangian dual function can be written as:

$$\Theta(\lambda, \nu) = \inf_{[\mathbf{x},\mathbf{y}] \in \mathcal{D}} \Lambda(\mathbf{x}, \mathbf{y}, \lambda, \nu) = \Theta_0(\lambda) + \sum_l \Theta_l(\lambda, \nu_l). \quad (39)$$

Note that all $\Theta_l(\lambda, \nu_l)$ can be evaluated independently, e.g., in parallel. As for $\Theta_0(\lambda)$, if $\exists i$ s.t. $a_i - \lambda_i \mathbf{b}_i < 0$, then setting $x_i = -\infty$ and $x_j = 0$ ($j \neq i$) leads to $\Lambda_0(\mathbf{x}, \lambda, \nu) = -\infty$, which implies that $\Theta_0(\lambda) = -\infty$; if $\forall i$, $a_i - \lambda_i \mathbf{b}_i \geq 0$, $\Lambda_0(\mathbf{x}, \lambda, \nu)$ achieves its minimum value when $\mathbf{x} = \mathbf{0}$. In summary,

$$\Lambda_0(\mathbf{x}, \lambda, \nu) = \left\{ \begin{array}{ll} -\infty & \text{if } \exists i, a_i - \lambda_i \mathbf{b}_i < 0 \\ 0 & \text{otherwise} \end{array} \right. \quad (40)$$

Since when $\Lambda_0(\mathbf{x}, \lambda, \nu) \to -\infty$, $\Theta(\lambda, \nu) \to -\infty$, which cannot be the optimal solution of $\Theta(\lambda, \nu)$, in the following, we only consider the case of $a_i - \lambda_i \mathbf{b}_i \geq 0$, $\forall i$.

Then, we can decompose DMCD-RL to a set of sub-problems sub_l ($l = 1, ..., N$): $\max \ \Theta_l(\lambda, \nu_l)$.

3.3.2 Subproblem solution

To solve the subproblems to get the final problem solution, we use the *Master algorithm* [6], which is a method to solve dual decomposition problems. It adjusts and combines the solutions from all the subproblems. We define the objective function of *master algorithm* by:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data

$\max \sum_l \Theta_l(\lambda, \nu_l)$. The master algorithm collects and compares the subproblems' solutions and sends feedback to subproblems to adjust the solutions if conflicts exist. The output of the master algorithm, $[\overline{\mathbf{x}}, \overline{\mathbf{y}}]$, is the optimal solution of the DMCD-RL problem.

Algorithm 2 shows the pseducode of the decomposition-based subgradient method. To implement DFD, the broker needs to use resources from a cluster of computers to realize the algorithm. At the beginning, the algorithm initiates $[\mathbf{x}^{(1)}, \mathbf{y}^{(1)}]$ by $\mathbf{0}$ (line 1). It then enters the main step (lines 4-16), which iteratively moves point $[\mathbf{x}, \mathbf{y}]$ to search the optimal point. In the k^{th} iteration, the master algorithm sends $[\mathbf{x}^{(k)}, \mathbf{y}^{(k)}]$ to each sub_l; given $[\mathbf{x}^{(k)}, \mathbf{y}^{(k)}]$, each sub_l finds its own optimal solution $[\lambda_l^{(k)}, \nu_l^{(k)}]$ and sends back to the master algorithm. Then, the master algorithm calculates the subgradient $\xi^{(k)}$ according to $[\lambda_1^{(k)}, \nu_1^{(k)}]$, ..., $[\lambda_M^{(k)}, \nu_M^{(k)}]$ (line 7) and finds whether $\xi^{(k)} = 0$, which means that the optimal solution is found. If not, the master program figures out a direction to move to the next point $[\mathbf{x}^{(k+1)}, \mathbf{y}^{(k+1)}]$ by calculating $[\mathbf{x}^{(k)}, \mathbf{y}^{(k)}]$'s subgradient (line 7). Then, the master program sends $[\mathbf{x}^{(k+1)}, \mathbf{y}^{(k+1)}]$ to each subproblem (lines 14) and the process repeats. Otherwise, the algorithm completes (line 9) and the optimal solution is obtained (line 17).

Theorem 3.3: DFD has $O(1)$ approximation ratio.

Proof: According to Lemma 3.2, there is no gap between the optimal solutions of MCD-RL and DMCD-RL. Hence, DFD has the same performance guarantee with the centralized algorithm introduced in Section 3.2. According to Theorem 3.2, DFD has $O(1)$ approximation ratio. □

4 Performance Evaluation

In this section, we conducted both simulation and real-world experiments (on Amazon EC2 [1]) driven by the Google Cluster [13] and PlanetLab [9] real traces. 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. The PlanetLab trace contains the CPU utilization of VMs in PlanetLab every 5 minutes for 24 hours in 10 random days in March and April 2011. In both Google Cluster and PlanetLab traces, the capacity of the CPU and memory of all the servers are not provided [9], [13]. We evaluated the effectiveness of our demand allocation algorithms (BaFD and DFD) in comparison with two typical demand allocation algorithms, BFDSum [27] and Sandpiper [36]. 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 $\text{Wei}_l = \prod_k \frac{1}{1-w_{l,k}}$, where Vol_i and Wei_l represent the volume of s_i and the weight of demand v_l, respectively.

In our simulation, we read the utilization data from the trace every 10 seconds and consider this data as the resource consumption of demands. We set two types of servers and each type has 50 servers. The first type of servers has 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:

1. *Total server cost*. It is defined as the total energy cost of all the servers purchased by CSB.
2. *Average ratio of server overload*. A server's *ratio of server overload* is defined as the percentage of time points in a time period that it is overloaded. An overload in a server's any resource leads to the server overload. This metric is the average values of all servers.
3. *CPU/memory utilization.* It is defined as the percentage of a server's CPU/memory capacity that is actually consumed. Higher utilization of a server means that the server's resource is more fully utilized, which reduces the total cost of all servers required for a given amount of demands.

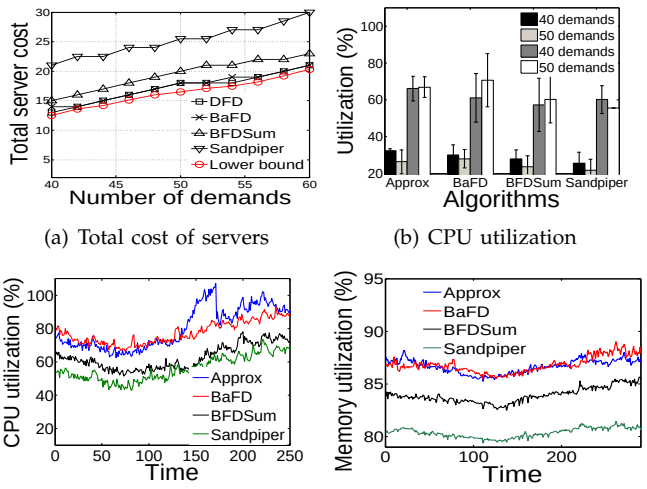
In both Google Cluster and PlanetLab traces, the CPU and memory consumption fluctuate over time, which implies that the resource requirement for each VM should be appropriately predetermined for demand allocation so that the real requirements can be satisfied but are close to the determined requirements 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%). According to the traces, for each user's demand, we first determine the consumption vector of each demand by taking the average value of the trace. Here, we use the following method [25] to determine the consumption vector. Let $\{w_{l,k}^t | t = 1, ..., T\}$ be the trace of type-k resource of demand v_l, then $w_{l,k}$ is given by: $w_{l,k} = \text{E}(w_{l,k}^t) + \eta\sigma(w_{l,k}^t)$, where $\text{E}(w_{l,k}^t)$ and $\sigma(w_{l,k}^t)$ represent the expectation and the standard deviation of $\{w_{l,k}^t\}$, respectively, and η is a coefficient to determine the percentage of trace data in $\{w_{l,k}^t\}$ that is lower than $w_{l,k}$. If $\eta = 1.28$, then about 80% trace data in $\{w_{l,k}^t\}$ is lower than $w_{l,k}$ [25], which means we set SLA to 80% demand satisfaction [25]. In all our experiments below, all algorithms achieve no less than 95% demand satisfaction.

4.1 Trace-driven Simulation

4.1.1 Google Cluster trace

First, we evaluate the performance of the four algorithms in simulation using the Google cluster trace. Fig 6(a) shows the total server cost of different algorithms 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 also draw the total server cost of MCD-RL's optimal solution, which provides a lower bound of the optimal total server cost (according to Proposition 3.1). We see that the result follows Sandpiper>BFDSum> BaFD $\approx$ DFD. 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 3.1,

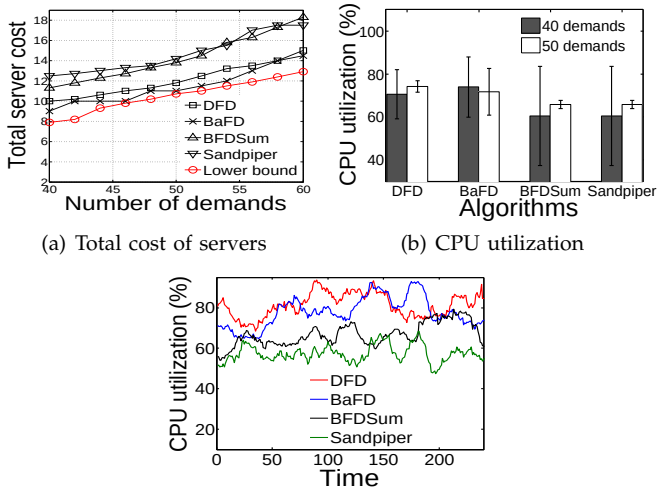
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data



(a) Total cost of servers (b) CPU utilization

(c) CPU utilization vs. time (sec) (d) Mem utilization vs. time (sec)

Fig. 6. Simulation using the Google Cluster trace.



(a) Total cost of servers (b) CPU utilization

(c) CPU utilization. vs time

Fig. 7. Simulation using the PlanetLab trace.

balancing the resource utilization for each server can increase each server's remaining capacity for allocating more demands. As for DFD, it sets the total costs of all used servers as its objective function, implying that DFD aims to search the demand allocation such that the total server cost is minimized. To achieve this goal, DFD 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 DFD 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. Finally, we see that BaFD and DFD's server cost is close to the lower bound derived by MCD-RL. As the optimal solution must be higher than this lower bound, it indicates that both BaFD and DFD are close to the optimal solution.

We measured the CPU and memory utilizations every 10 seconds for all servers. Fig 6(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 $\approx$ DFD, 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 6(a). We also compare the CPU and memory utilization of a randomly selected server under different algorithms over time in Fig 6(c) and Fig 6(d), respectively. Comparing these two figures, we find that for both CPU and memory resource utilizations, Sandpiper $\approx$ BFDSum $<$ BaFD $\approx$ DFD, which is consistent with the results in Fig. 6(b) due to the same reasons.

4.1.2 PlanetLab trace

We also evaluate the performance of the different algorithms using the PlanetLab trace. Fig 7(a) shows the total cost of servers of the four algorithms when the number of demands was varied from 40 to 60 with 2 increase in each step. From the figure, we can find that the total cost of servers follows Sandpiper $\approx$ BFDSum $>$ BaFD $\approx$ DFD. It demonstrates that BaFD and DFD generate less energy cost than Sandpiper and BFDSum. Besides balancing the resource utilization, both BaFD and DFD take into account the servers' cost when they allocate demands to servers. More specifically, BaFD always tries to allocate each demand to the server with the highest ratio of the sum weight of demands to the server's cost. DFD sets the total costs of all used servers as its objective function, which means that DFD aims to search the demand allocation such that the total cost of all servers is minimized. In contrast, BFDSum and Sandpiper do not consider the energy cost of servers in demand allocation. Finally, we can observe that both BaFD and DFD have total server cost close to the lower bound derived from MCD-RL, indicating BFD and DFD are close to the optimal solution.

Fig 7(b) shows the median, 5th percentile and 95th percentile of the CPU utilizations at all time points of all servers of the four algorithms with 40 and 50 demands. The CPU utilization data is collected from all the servers at each time slot. We observe that the average utilization of CPU follows Sandpiper $\approx$ BFDSum $<$ BaFD $\approx$ DFD, which indicates that our two algorithms can more fully utilize the resources of servers. The reason is the same as in Fig 7(a). We also compare the CPU utilization of one server under different algorithms over time in Fig 7(c). We can have a similar observation as Fig. 6(c): Sandpiper $\approx$ BFDSum $<$ BaFD $\approx$ DFD, which is also consistent to the results in Fig. 7(b) due to the same reasons. The results confirm that BaFD and DFD more fully utilize resources in each server and hence reduce the number of servers and save energy cost.

4.1.3 Comparison of BaFD and DRR

The idea of balancing servers' utilization on different types of resources has been investigated by many existing works. Here, we pick up one method, namely *Dominant-Residual Resource aware FFD* [40] (or simply DRR), as a compared method. Similar with BaFD, the basic idea of DRR is to balance the resource

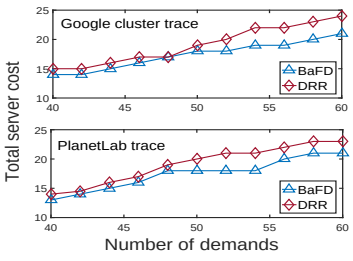
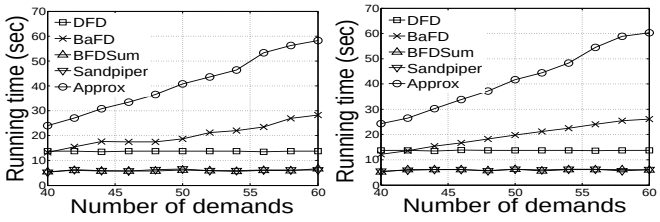


Fig. 8. BaFD vs. DRR.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data



(a) Google Cluster trace (b) PlanetLab trace

Fig. 9. Running time for different pricing functions.

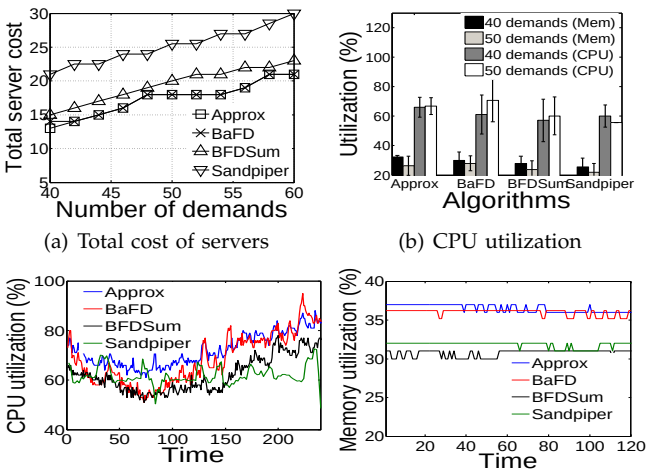
utilizations across all dimensions by matching up the server's residual resource capacity with the next demand to be allocated. However, DRR assumes homogenous servers and hence cannot achieve high energy efficiency when it is applied heterogenous servers with different cost. We compare the total server cost of BaFD and DRR-FFD by using both Google cluster trace and PlanetLab trace in Fig. 8. From both figures, we can observe that the cost of DRR-FFD is lightly higher than that of BaFD.

4.2 Computing time

Fig. 9(a) and Fig. 9(b) compare the running time of different algorithms using Google Cluster trace and PlanetLab trace, respectively. Besides the four algorithms, we also ran the approximation algorithm introduced in Section 3.2 (denoted by Approx) and depicted its running time in both figures, to measure how much time DFD can save compared with Approx. Because the computing time of the five algorithms are the same in both simulation and Amazon EC2, we will not show these results for Amazon EC2. From both figures, we find that the computation time of DFD is much lower than that of Approx. The figures also show that, as the number of demands increases, the computing time of DFD, BFDSum, and Sandpiper remains at the same level, while the computing time of BaFD and Approx increases. Different from BFDSum and Sandpiper that only greedily select the server with the minimum remaining capacity in each iteration, BaFD needs to 1) first iteratively fits all the unallocated demands into each server, and then 2) selects the server with the highest efficiency. To obtain the solution of MCD-RL, Approx needs to first run the simplex method, of which the computation time increases with the increase of the problem's scale. Therefore, both BaFD and Approx's computing time is more likely to be affected by the number of demands. The computing time of DFD does not increase with the increasing number of demands because DFD divides the problem into a certain number of subproblems, and figures out each subproblem in parallel.

4.3 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 and PlanetLab 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 4.1, and then compare the results with the simulation results in Section 4.1.



(a) Total cost of servers (b) CPU utilization

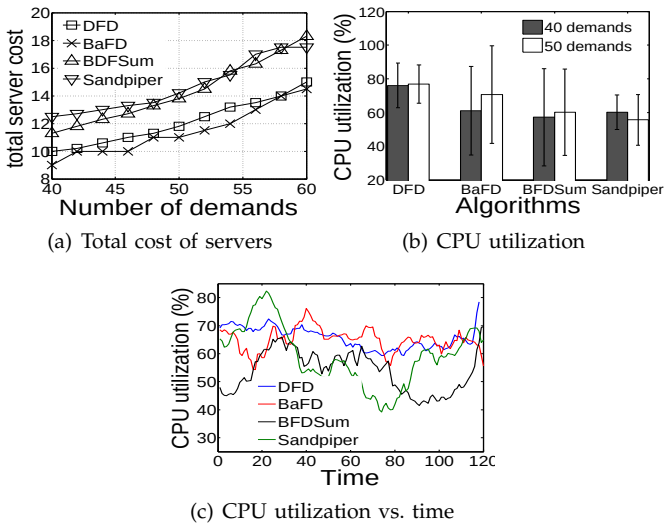
(c) CPU utilization vs. time (d) Mem utilization vs. time

Fig. 10. Exp. on Amazon EC2 (Google Cluster trace).

4.3.1 *Google Cluster trace*

Fig. 10 shows the performance of the four algorithms implemented in Amazon EC2 using Google Cluster trace. Comparing Fig. 6 and Fig. 10, we have the following observations: (1) in both Fig. 6(a) and Fig. 10(a), the total cost of servers follows: Sandpiper $\approx$ BFDSum $>$ BaFD $\approx$ DFD, and (2) in all Fig. 6(b)-(d) and Fig. 10(b)-(d), both CPU utilization and memory utilization follow Sandpiper $\approx$ BFDSum $<$ BaFD $\approx$ DFD, and (3) all the algorithms have higher ratio of CPU overloads in Amazon EC2 than in simulation. Observations (1) and (2) are caused by the same reasons as in Fig. 6(a) and Fig. 6(b). BaFD and DFD have lower servers' cost and higher resource utilization due to the same reasons as in Fig. 6(a) and Fig. 6(b), because, as we mention before, both of them jointly take into account servers' cost and the balance of CPU utilization and memory utilization, while BFDSum and Sandpiper simply map all the capacity vectors and consumption vectors to single scalars, without considering the balance between different types of resources' utilization and the servers' cost. For observation (3), the experiments in Amazon EC2 have more CPU overloads because, besides the process generated by the Google Cluster trace, there are some other background process in Amazon EC2's servers, which introduce more CPU utilization. The results in Fig. 10 demonstrate that

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TBDATA.2018.2823330, IEEE Transactions on Big Data



(a) Total cost of servers (b) CPU utilization

(c) CPU utilization vs. time

Fig. 11. Experiment on Amazon EC2 (PlanetLab trace).

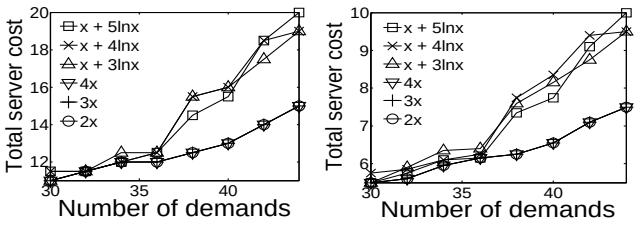
BaFD and DFD more fully utilize resources in each server and hence save energy cost in Amazon EC2.

4.3.2 PlanetLab

Fig. 11 shows the performance of the four algorithms implemented in Amazon EC2 using the PlanetLab trace. Similarly, by comparing Fig. 11 with Fig. 7, we find that (1) the total cost of servers follows: Sandpiper $\approx$ BFDSum $>$ BaFD $\approx$ DFD, and (2) the CPU utilization for the four algorithms follows: Sandpiper $\approx$ BFDSum $<$ BaFD $\approx$ DFD. These results are consistent to the results using Google Cluster (Fig. 10) and due to the same reasons. The results in Fig. 11 confirm that BaFD and DFD cost less energy and can more fully utilize server resources.

4.4 Comparison of Different Pricing Policies

Recall that we proved in Section 2 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. 12(a) and Fig. 12(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 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. 6) in the MCP problem in Section 2, which further determines the allocation strategy of the CSB.



(a) Google Cluster trace (b) PlanetLab trace

Fig. 12. Comparison of total cost of servers for different pricing functions.

The experimental results in Fig 12(a) and Fig. 12(b) confirm the effectiveness of our pricing policy to incentivize CSBs to be cooperative in minimizing the energy cost while satisfy tenants' demands.

5 Related Work

5.1 Demand allocation

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 vSphere 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 [27], [30], [36]. Sandpiper [36] 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.* [30] 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.* [27] proposed to use Euclidean distance between resource demands and residual capacity as a metric for consolidation, a heuristic analogous to Norm-based Greedy. Lee [20] *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. By considering the problem of resource over-provisioning that may lead to low resource utilization, Chen and Shen designed a system that excludes bursts in demand prediction and handles bursts to avoid resource over-provisioning in [11], and also presented three VM resource utilization pattern refinement algorithms to improve the original predicted utilization pattern in [12]. 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. Though [22], [37], and [40] have the strategies with the consideration of resource utilization balance of each server, they assumed homogeneous physical machines with the same cost and capacity, which still cannot be directly applied in our scenario. Also, their algorithms can only be implemented in a centralized

way, leading to high delay when the number of tenants is large.

In addition, since the electricity bill of a datacenter constitutes a significant portion of its overall operational costs, intensive research efforts have been devoted to the study of energy efficiency for datacenters [5], [21], [24], [31]. Some of these works [21], [24], [31] use energy storage device (e.g., battery or uninterrupted power supply (UPS) units) while other works [5] arrange data to the servers that have the highest energy efficiency. All the above works do not handle demand allocation to servers for energy-efficiency, which is the focus of our work. For example, using the technique of Lyapunov optimization, Urgaonkar *et al.* [31] developed an online control algorithm that can optimally exploit uninterrupted power supply (UPS) units to reduce the energy cost in a datacenter. Wang *et al.* [32] presented a theoretical framework for multiple type energy storage devises (ESD) with a multiple-level power delivery hierarchy. They also developed a generalized optimization platform for ESD placement and control in the datacenter.

5.2 Pricing problem in clouds

A number of studies apply auctions policies to price computing resources in a cloud system [33], [35], [38]. Wang *et al.* [33] 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.* [35] 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.* [23] considered a model of cloud bandwidth allocation and pricing when explicit bandwidth reservation is enabled. Shen and Li [26] 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 [34] [39]. Wang *et al.* [34] studied how a cloud should allocate its resources between the on-demand market and the auction market. Zhang *et al.* [39] 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.

6 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? For high scalability, we also proposed a decomposition-based algorithm to implement the approximation algorithm. 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.

7 Acknowledgement

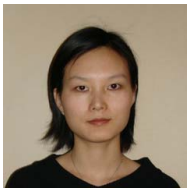
This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, CNS-1249603, and Microsoft Research Faculty Fellowship 8300751

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] H. Amur, J. Cipar, V. Gupta, G. R. Ganger, M. A. Kozuch, and K. Schwan. Robust and flexible power-proportional storage. In *Proc. of SoCC*, 2010.
[6] M. Bazaraa, H. Sherali, and C. Shetty. Nonlinear programming: Theory and algorithms. *Wiley Interscience*, 2006.
[7] 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.
[8] H. S. C. Qiu and L. Chen. Probabilistic demand allocation for cloud service brokerage. In *Proc. of IEEE INFOCOM*, 2016.
[9] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *SPE*, 2011.
[10] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *Proc. of USENIX NSDI*, 2008.
[11] L. Chen and H. Shen. Towards resource-efficient cloud systems: Avoiding over-provisioning in demand-prediction based resource provisioning. In *Proc. of IEEE INFOCOM*, 2016.
[12] L. Chen and H. Shen. Considering resource demand misalignments to reduce resource over-provisioning in cloud datacenters. In *Proc. of IEEE INFOCOM*, 2017.
[13] G. cluster data. https://code.google.com/p/googleclusterdata/.
[14] A. C. Compute. http://aws.amazon.com/ec2/hpc- applications/., 2011.
[15] Z. C. Computing. http://www.zimory.com/.
[16] M. Dayarathna, YonggangWen, and R. Fan. Data center energy consumption modeling: A survey. 2016.
[17] F. S. Hillier. *Linear and Nonlinear Programming*. Stanford University, 2008.
[18] IBM. http://www.ibm.com/cloud-computing/us/en/products/dedicated-bare-metal-servers.html/.
[19] IBM. Make IT Green - Cloud Computing and its Contribution to Climate Change.
[20] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder. Validating heuristics for virtual machines consolidation, 2011.
[21] Z. Liu, M. Lin, A. Wierman, and S. H. Low. Greening geographical load balancing. In *Proc. of Sigmetrics*, 2011.

[22] Z. A. Mann. Approximability of virtual machine allocation: much harder than bin packing. In *Proc. of Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, 2015.
[23] D. Niu, C. Feng, and B. Li. A theory of cloud bandwidth pricing for video-on-demand providers. In *Proc. of INFOCOM*, 2012.
[24] D. S. Palasamudramy, R. K. Sitaramanz, B. Urgaonkary, and R. Urgaonkar. Using batteries to reduce the power costs of internet-scale distributed networks. In *Proc. of SoCC*, 2012.
[25] S. M. Ross. *Introduction to Probability Models, 8th Edition*. Amsterdam: Academic Press, 2003.
[26] 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.
[27] S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. In *Proc. of HotPower*, 2008.
[28] F. TA and R. MG. A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, 1989.
[29] F. TA and R. MG. Greedy randomized adaptive search procedures. *Journal of global optimization*, 1995.
[30] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici. A scalable application placement controller for enterprise data centers. In *Proc. of WWW*, 2007.
[31] R. Urgaonkar, B. Urgaonkar, M. J. Neely, and A. Sivasubramanian. Optimal power cost management using stored energy in data centers. In *Proc. of Sigmetrics*, 2011.
[32] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy. Energy storage in datacenters: What, where, and how much? In *Proc. of Sigmetrics*, 2012.
[33] Q. Wang, K. Ren, and X. Meng. When cloud meets ebay: Towards effective pricing for cloud computing. In *Proc. of INFOCOM*, 2012.
[34] W. Wang, B. Li, and B. Liang. Towards optimal capacity segmentation with hybrid cloud pricing. In *Proc. of ICDCS*, 2012.
[35] W. Wang, B. Liang, and B. Li. Revenue maximization with dynamic auctions in iaas cloud markets. In *Proc. of IWQoS*, 2013.
[36] 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.
[37] M. G. S. Zaourar. Variable size vector bin packing heuristics application to the machine reassignment problem. *Distributed, Parallel, and Cluster Computing*.
[38] 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.
[39] Q. Zhang, E. Grses, R. Boutaba, and J. Xiao. Dynamic resource allocation for spot markets in clouds. In *Proc. of Hot-ICE*, 2011.
[40] Y. Zhang and N. Ansari. Heterogeneity aware dominant resource assistant heuristics for virtual machine consolidation. In *Proc. of IEEE Globecom*, 2013.
[41] 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.

Chenxi Qiu Chenxi Qiu received the BS degree in Telecommunication Engineering from Xidian University, China, in 2009 and the Ph.D. degree in Electrical and Computer Engineering in Clemson University in 2015. He currently is a Post-doc scholar in the College of Information and Science at Pennsylvania State University, PA, United States. His research interests include cyber security, cyber physical systems, and cloud computing.



Haiying Shen Haiying Shen received the BS degree in Computer Science and Engineering from Tongji University, China in 2000, and the MS and Ph.D. degrees in Computer Engineering from Wayne State University in 2004 and 2006, respectively. She is currently an Associate Professor in the Department of Computer Science at University of Virginia. Her research interests include distributed computer systems and computer networks, with an emphasis on P2P and content delivery networks, mobile computing, wireless sensor networks, and grid and cloud computing. She was the Program Co-Chair for a number of international conferences and member of the Program Committees of many leading conferences. She is a Microsoft Faculty Fellow of 2010 and a member of the IEEE and ACM.

Liuhua Chen received both his BS degree and MS degree from Zhejiang University, in 2008 and 2011, and is currently working toward the PhD degree in the Department of Electrical and Computer Engineering at Clemson University. His research interests include distributed and parallel computer systems, and cloud computing.

APPENDIX

.1 Proof of Lemma 3.1

Proof: First, we claim that no pair x'_i and x'_j s.t. $x'_i \leq \frac{1}{2}$ and $x'_j \leq \frac{1}{2}$ 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 $x'_i \geq \frac{1}{2}$

$$\sum_{s_i \in S_n} \hat{x}_i = \sum_{s_i \in S_n} \lceil x'_i \rceil < 2 \sum_{s_i \in S_n} x'_i = 2 \sum_{s_i \in S_n} \tilde{x}_i. \tag{41}$$

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

$$\begin{aligned} \sum_{s_i \in S_n} \hat{x}_i &= \sum_{s_i \in S_n} \lceil x'_i \rceil < 2 \sum_{s_i \in S_n \setminus s_j} x'_i + \lceil x'_j \rceil & (42) \\ &= 2 \sum_{s_i \in S_n \setminus s_j} \tilde{x}_i + 1 & (43) \\ &< 2 \sum_{s_i \in S_n} \tilde{x}_i + 1. & (44) \end{aligned}$$

□

.2 Proof of Theorem 3.2

Proof: First, by Lemma 3.1

$$\begin{aligned} \sum_i a_i \hat{x}_i &= \sum_n \left(a_n \sum_{s_i \in S_n} \hat{x}_i \right) < \sum_n \left(a_n \left(2 \sum_{s_i \in S_n} \tilde{x}_i + 1 \right) \right) \\ &= 2 \sum_i a_i \tilde{x}_i + C & (45) \end{aligned}$$

where $C = \sum_n a_n$ is a constant. Also, based on Equ. (31), we can derive that

$$\begin{aligned} \sum_i a_i \tilde{x}_i &= \sum_i a_i \max_k \left\{ \frac{\sum_l \hat{y}_{i,l} w_{l,k}}{b_{i,k}} \right\} \\ &\leq \sum_i a_i \sum_l \delta_{i,l} \hat{y}_{i,l} \quad (\delta_{i,l} = \max_k \left\{ \frac{w_{l,k}}{b_{i,k}} \right\}) \\ &\leq \sum_l \beta_l \sum_i \hat{y}_{i,l} \quad (\beta_l = \max_i \{a_i \delta_{i,l}\}) \\ &\leq \Delta \sum_i a_i \sum_l \frac{\overline{y}_{i,l} w_{l,1}}{b_{i,1}} \quad (\Delta = \max_{i,l,k} \left\{ \frac{b_{i,1} \beta_l}{w_{l,1} a_i} \right\}) \\ &\leq \Delta \sum_i a_i \overline{x}_i & (46) \end{aligned}$$

Based on Equ. (45), 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. (45) and Equ. (46), 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 $\sum_i a_i \overline{x}_i \leq \sum_i a_i x^*_i \leq \sum_i a_i \hat{x}_i$ (Proposition (3.1)), we can derive that the approximation ratio is upper bounded by 2Δ, which is a constant. □