# Probabilistic Demand Allocation for Cloud Service Brokerage

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

***Abstract*—Functioning as an intermediary between cloud tenants and providers, cloud service brokerages (CSBs) bring about great benefits to the cloud market. To maximize its own profit, a CSB is faced with a challenge: how to reserve servers and distribute tenant demands to the reserved servers such that the total reservation cost is minimized while the reserved servers can satisfy the tenant service level agreement (SLA)? Demand prediction and demand allocation are two steps to solve this problem. However, previous demand prediction methods cannot accurately predict tenant demands since they cannot accurately estimate prediction errors and also assume the existence of seasonal periods of demands. Previous demand allocation methods only aim to minimize the number of reserved servers rather than the server reservation cost, which is more challenging. To solve this challenge, we propose a *Probabilistic Demand Allocation* system (PDA). It predicts demands and more accurate prediction errors without the assumption of the existence of seasonal periods. It then formulates a nonlinear programming problem and has a decentralized method to find the problem solution. In addition to overcoming the shortcomings in previous methods, PDA is novel in that rather than separately conducting the prediction and demand allocation, it considers prediction errors in demand allocation in order to allocate demands with offsetting prediction errors (e.g., -1 and +1) to the same server, which helps find the problem solution. Both simulation and real-world experimental results demonstrate the superior performance of our system in reducing servers' reservation cost.**

## I. Introduction

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. In particular, if a cloud tenant wants to use the cloud resource from multiple cloud providers, it 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. 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 cloud market [1]–[4].

A CSB is a third-party individual or business that acts as an intermediary between the tenants and the cloud providers. A CSB reserves the cloud resources (e.g., servers) from the cloud providers and sells services (e.g. virtual machine (VM)) along with administration and security to the tenants with higher prices [5]. Usually, CSBs can make cloud services more valuable for cloud tenants, because CSBs work closely with cloud providers to get price breaks or access to more information about how much resources are required for a service [1]. In addition, CSBs can enhance the security of cloud services for tenants because they can monitor, track, protect and enforce company policies across all demands from different tenants [1]. Thus, CSBs can make it easier, less expensive, safer and more productive for tenants to use cloud resources, particularly when a tenant's demands span multiple cloud service providers.

To maximize its own profit, a CSB is faced with a challenge: *how to reserve servers and distribute tenant demands to the reserved servers such that the total reservation cost is minimized while the reserved servers can satisfy all the demands based on tenants' service level agreement (SLA) (i.e., satisfying all the demands with a given probability)?*
Demand prediction and demand allocation are two important functions to handle this challenge. Demand prediction predicts the amount of resources required by each demand in a future time period (e.g., one month) and demand allocation allocates demands to servers that can meet the demands. Previous demand allocation works [6]–[8] aim to find an allocation schedule that minimizes the number of allocated servers while satisfy all demands. However, these works are based on the assumption that the demands can be accurately predicted. Unfortunately, previous demand prediction methods [9]–[11] cannot accurately predict tenant demands since they neglect or cannot accurately estimate prediction errors and also assume the existence of seasonal periods of demands. Further, their methods to estimate the seasonal period are not time-efficient [12] and they are not suitable for predicting demands that do not exhibit seasonal periods (e.g. some demands in Google cluster [13]). A recent work, CloudScale [11], takes into account the underestimation of demands in prediction and adds padding on demands to avoid underestimation. However, it omits the case that the demands can be also overestimated, which leads to under-utilization of server resources.

Even if we can accurately estimate the demands, the previous demand allocation methods cannot solve the above-mentioned challenge because their objective is to minimize the number of reserved servers while our goal is to minimize the cost of the reserved servers, which however is more difficult.

This is because we need to consider not only which server can best fit a given demand (in order to minimize the number of servers) but also the reservation cost of the server. In most cases, the server that best fits a demand is not the server with the minimum reservation cost.

To solve the above-indicated challenge, we propose a *Probabilistic Demand Allocation (PDA)* system, which consists of demand prediction and demand allocation.

**Demand prediction.** We predict demands and more accurate prediction errors without the assumption of the existence of seasonal periods. Specifically, we model the historical data of tenants' demand by the seasonal autocorrelated moving average (SARMA) model for demand prediction. We estimate the prediction errors using the maximum likelihood estimation (MLE) [14], which determines the prediction errors that make the observed demands the most probable.

**Demand allocation.** We formulate a stochastic programming problem for the challenge and find the problem solution through theoretical work. The input of this problem is the predicted value of each demand and its estimation error in the next period and the output determines which servers should be reserved and how to allocate each demand to the reserved servers. Finally, we propose a decentralized method using the technique of Lagrangian dual decomposition [15] to find the solution for this problem.

In addition to overcoming the shortcomings in previous methods, PDA is novel in that rather than separately conducting the prediction and demand allocation, it considers prediction errors in demand allocation in order to allocate demands with offsetting prediction errors (e.g., -1 and +1) to the same server, which helps find the problem solution. Within our knowledge, this is the first work that provides guidance to CSBs in server reservation and demand allocation across multiple clouds to minimize the reservation cost while guarantee the reserved servers can satisfy demands with a given probability.

We test the performance of PDA in comparison with the previous algorithms by both trace-driven experiments and on Amazon EC2 [16]. The experimental results demonstrate the superior performance of our system. In summary, our contributions can be summarized as follows:

1. We design a prediction method that can predict the future demand cost based on the observed historical data. Different from previous prediction method that only considers the possibility of underestimation, we consider both cases of underestimation and overestimation.
2. Using the predicted data, we formulate a new demand allocation problem for the CSB, namely PDA, which has a different objective with the traditional demand allocation problem, i.e., the new problem's objective is to minimize the cost of the reserved servers, rather than to minimize the number of servers. As a solution, we also propose a gradient method based on Lagrangian dual decomposition, which can be implemented in a decentralized way.
3. Finally, we test the performance of PDA in comparison with the previous algorithms by both trace-driven experiments on a simulator and on Amazon EC2 [16]. The experimental results demonstrate the superior performance of our algorithms in the aspects of total reservation cost and resource utilization.

The remainder of this paper is organized as follows: Section II outlines the framework of PDA and introduces the system model used in this paper. Section III and Section IV describe the prediction part and demand allocation part of PDA, respectively. Section VI evaluates the performance of our proposed schemes in comparison with other algorithms. Section VII presents related work. Section VIII concludes this paper with remarks on our future work compared with previous methods.

## II. The Architecture of PDA

In this section, we will first briefly outline the architecture of the PDA system, which is composed of two parts: the *demand prediction* part and the *demand allocation* part. Also, we will describe the system model and some important concepts, notations, and assumptions that will be used in this paper.

First, we consider a scenario composed of multiple public cloud providers, tenants, and a CSB. The CSB reserves servers from cloud providers and allocates the demands from tenants to the reserved servers. We assume a dynamic system, i.e., the whole time span is partitioned into a number of periods (e.g., one month for a period) and the CSB needs to re-reserve servers and re-allocate the demands at the beginning of each period. The goal is to minimize the reservation cost while guarantee the reserved servers can still satisfy tenants' demands with a given probability in each period. Here, a demand can be a VM in IaaS (Infrastructure as a Service) model or a video game in SaaS (Software as a Service) model. Fig. 1 shows the architecture of PDA: At the beginning of a short-term period, the CSB analyzes the history of each demand consumption from cloud monitoring services by *monitor*, and then uses the historical data to predict the expected value of each demand by *predictor*. After that, the CSB delivers the predicted value to *allocator*, which is responsible for allocating the demands to different servers. While the CSB sells cloud service to tenants individually, it jointly reserves the different types of cloud resources from multiple cloud providers using the allocator.

More specifically, we assume there are totally $M$ demands $V = \{v_1, v_2, ..., v_M\}$ from tenants, where each demand is composed of $K$ different types of resources (e.g., CPU, memory, and storage). In reality, the resource consumption for each demand does not always remain the same level, but randomly fluctuates over time. As previous works in [5], [11], [17], we consider the case that the demands are predictable. We characterize the resource consumption of each demand $v_l$ by a *time series* $\{\mathbf{w}_t^l\}_{t\in\mathbb{Z}^+}$ $(l = 1, ..., M)$, where $\mathbf{w}_t^l = [w_t^{l,1} ... w_t^{l,k} ... w_t^{l,K}]$ and $w_t^{l,k}$ represents the consumption of type $k$ resource at time slot $t$ in demand $v_l$. At each period $t$, the CSB collects each $\{\mathbf{w}_1^l, ..., \mathbf{w}_{t-1}^l\}$ from the monitor, and then uses the predictor to predict $\mathbf{w}_t^l$. In addition, we don't consider the case that the demands can enter or leave the system.

Suppose that the CSB needs to allocate demands at beginning of period $T$. After getting $\mathbf{w}_T^l$ from the predictor, the CSB uses the allocator to allocate the demand of tenants

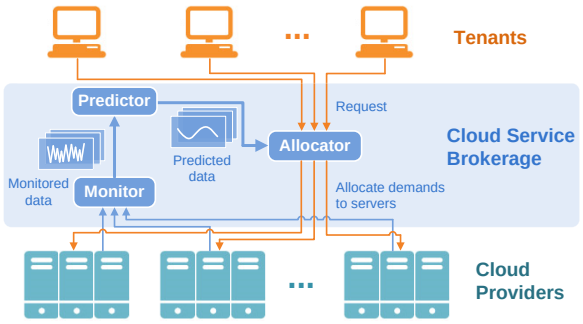



Fig. 1. Cloud service brokerage.

to different cloud providers. Similar to demand, each server $s_i$ can be characterized by a $K$-dimensional *capacity vector* $\mathbf{b}_i = [b_{i,1}, ..., b_{i,K}]^\top$, where each dimension $b_{i,j}$ represents the server's capacity on type-$j$ resource. We assume that there are $N$ heterogeneous servers $S = \{s_1, ..., s_N\}$. Here, "heterogeneous" means that the servers in $S$ have different reservation costs and different capacity vectors. We normalize the entries of each $\mathbf{w}_T^l$ and each $\mathbf{b}_i$ through dividing $w_T^{l,k}$ and $b_{i,k}$ by $b_{\max} = \max_{i,k} b_{i,k}$ in all demands and all servers, respectively. We assume that the reservation price of each server is fixed when it is used regardless of its resource utilization [8], and we use $c_i$ to denote the reservation price of each $s_i$ and $\mathbf{c} = [c_1, ..., c_N]^\top$.

We assume that cloud providers always have enough resource to sell to CSB [5], which is justified by the "illusion of infinite capacity". 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 server $s_i$: if yes, $y_{i,l} = 1$; otherwise $y_{i,l} = 0$. We define $\varepsilon$ as the *risk factor*. The objective of the allocator is to minimize the reservation cost while guaranteing all the demands can be satisfied by the reserved servers with a given probability $1 - \varepsilon$. We represent the allocator's objective formally by

$$\min \ \mathbf{c}^\top \mathbf{x} \tag{1}$$

$$\text{s.t. } \Pr\left(\sum_l w_T^{l,k} y_{i,l} > b_{i,k} x_i\right) \leq \varepsilon \ \forall i,k. \tag{2}$$

Here, Equ. (2) means that, for each server $s_i$, allocator requires the probability that the total consumption of the demands allocated to this server exceeds the server's capacity to be no higher than $\varepsilon$. In addition, each demand should be allocated to exactly one server, i.e.,

$$\mathbf{y}^j \mathbf{1} = 1, \tag{3}$$

where $\mathbf{1}$ is an $N$ dimensional vector with each entry equals to 1. Later on, we will prove that the problem with objective function (1) and constraints (2) (3) is NP-hard (Section V-A), and introduce how to solve this problem using the subgradient method [15] (Section V-B).

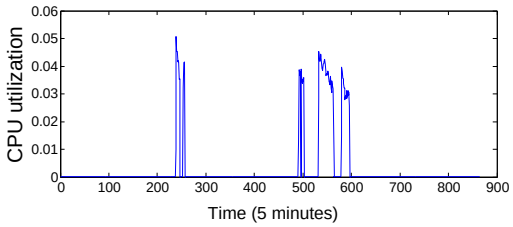



Fig. 2. The seasonal period varies in different demands.

## III. Demand Prediction

In this section, we will introduce how to predict demands of different types of resources for tenants. As a solution, we use the seasonal autoregressive moving-average (SARMA) model, which has been widely used for the prediction of time series [18]. In Section III-A, we will first introduce the problems we need to solve to predict demands more accurately and more efficiently. In Section III-B and Section III-C, we will introduce how we solve these problems.

### A. Accurate and Efficient Demand Prediction

Previous demand prediction methods assume the existence of seasonal period [19]. However, we observed from the trace in Google cluster [13], which records the CPU and memory resource utilization on a cluster of about 11000 machines from May 2011 for 29 days in every 5 minutes, which shows that not all the demands has seasonality. Specifically, we analyze all these demands and find that 32.7% demands do not have seasonality. For example, Fig. 2 shows that no seasonality exists for the CPU utilization of a demand that we picked up in Google cluster within 900 intervals, where each interval lasts for 5 minutes. We then use SARMA for prediction, which can describe not only the time series with seasonality but also the time series without seasonality. Specifically, we first need to estimate the parameters of SARMA using the historical demand consumption, and then based on the estimated parameters, we predict the future demands $\hat{w}_T^{l,k}$ $(1 \leq l \leq M, 1 \leq k \leq K)$ using SARMA. In SARMA, the parameter $d$ denotes the seasonal period. If no seasonality is found, the parameter $d = T$, which means seasonality does not exist. After estimating the seasonal period $d$, we can use the Innovation algorithm [18] to estimate all other parameters in SARMA. If a demand follows seasonality, we need to efficiently estimate the seasonal period of the demand. After the prediction, we also need to estimate the prediction errors for more accurate demand prediction. In particular, there are two problems we need to solve and we introduce the solutions in Sections III-B and III-C.

**Seasonal period estimation.** In our prediction method, we use Fast Hartley Transform (FHT) [12] to estimate the seasonal period [20], [21].

**Prediction error estimation.** Because the noise cannot be predicted even in a stationary time series, the prediction errors cannot be avoided, i.e., the actual demand is higher or lower than the expected demand. However, we need to determine

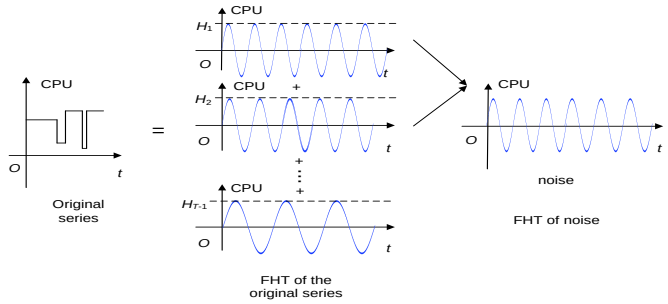



Fig. 3. FHT-based estimation of seasonal period in a demand.

the allocation of a demand by its predicted resource cost, prediction errors may lead to resource overload or under-utilization. Recall that in PDA, each tenant's demand must be satisfied with probability higher than $1-\varepsilon$, simply using the predicted value from SARMA cannot guarantee meeting such a requirement and also previous work [11] only considered underestimation. To solve this problem, we estimate the variance of prediction errors using maximum likelihood estimation (MLE) [14], which determines the parametric values that make the observed results the most probable.

### B. Seasonal Period Estimation

To estimate the seasonal period of each demand, we use a signal processing technique, called FHT [12], to discover the seasonal period. Given a series $\{w_1^{l,k},...,w_{T-1}^{l,k}\}$, FHT is defined as a linear and invertible function $H:\mathbb{R}^n\to\mathbb{R}^n$, which decomposes the $T-1$ elements in a resource consumption time series $\{w_1^{l,k},...,w_T^{l,k}\}$ into $T-1$ periodic series (as shown in Fig. 3). The $m^{\text{th}}$ period series has the period $\frac{T-1}{2\pi m}$ and the amplitude $H_m$, which can be calculated by

$$H_m=\sum_{t=1}^{T} w_t^{l,k}\left[\cos\left(\frac{2\pi}{T-1}tm\right)+\sin\left(\frac{2\pi}{T-1}tm\right)\right] \quad (4)$$

where $m=1,\ldots,T-1$. We employ FHT to calculate the dominant frequencies (whose $H_m$s are higher than a given threshold) of resource demand, and then pick the lowest dominant frequency. Suppose that the frequency we choose is $1/d'$, then the estimated seasonal period $d$ equals $d'$. Here, we don't choose the high dominate frequencies to determine the seasonal period because they cannot easily be differentiated from noise [14] (Fig. 3).

If there is no dominant frequency selected in FHT, which means no seasonality exists in the demand, then the seasonal period $d=T$ in SARMA. Using $d$ and and all other estimated parameters in SARMA, we finally get the expected value of all the demands using SARMA.

### C. Prediction Error Estimation

As we mentioned before, prediction errors cannot be avoided during the predicting process. Hence, we need to estimate the distribution of these errors, such as variance. We implement the estimation of prediction error variance using maximum likelihood estimation [14], which determines the prediction error variance, which is defined by

$$\sigma^{l,k}=\frac{\sum_{t=1}^{T-1}\left(\hat{w}_t^{l,k}-w_t^{l,k}\right)^2}{T-1} \quad (5)$$

where $w_t^{l,k}$ and $\hat{w}_t^{l,k}$ represent the actual value and expected value of $v_l$ on resource $k$. In the following we use vector $\mathbf{v}^{l,k}$ to represent the errors $[w_1^{l,k}-w_1^{l,k},...,w_{T-1}^{l,k}-w_{T-1}^{l,k}]$. Then, MLE tries find the value of $\sigma_t^{l,k}$, i.e., the prediction error variance, that makes the observed results $\mathbf{v}^{l,k}$ the most probable. That is, given a predicted value, we estimate the predicted error variance. To do this, we find the value of $\sigma_t^{l,k}$ that maximizes the likelihood function $L^{l,k}\left(\mathbf{v}^{l,k}|\sigma^{l,k}\right)$,

$$L^{l,k}\left(\mathbf{v}^{l,k}|\sigma^{l,k}\right)=\ln f\left(\sigma^{l,k}\right) \quad (6)$$

which represents the probability that $\mathbf{v}^{l,k}$ happens given $\sigma^{l,k}$, that is its partial derivative, i.e., $\frac{\partial L^{l,k}\left(\sigma^{l,k}|\mathbf{v}^{l,k}\right)}{\partial\sigma_t^{l,k}}$, should be equal to 0. In the following, we represent how to calculate $L^{l,k}\left(\mathbf{v}^{l,k}|\sigma^{l,k}\right)$, and then estimate the prediction error variance $\sigma_t^{l,k}$, when the partial derivative equals 0.

To calculate $L^{l,k}\left(\mathbf{v}^{l,k}|\sigma^{l,k}\right)$, we first need to specify the joint PDF for $\mathbf{v}^{l,k}$. Here, we use a widely used statistic model to describe $\mathbf{v}^{l,k}$ in time series, which assumes that the demands $\mathbf{v}^{l,k}$ follows multi-variant Gaussian distribution, which is a widely used statistic model to describe the noise, i.e., prediction errors, [14], [18], [22]

$$f\left(\sigma^{l,k}\right)=\frac{1}{\sqrt{(2\pi)^T|\Gamma_n^{l,k}|}}\exp\left(-\frac{1}{2}\mathbf{v}^{l,k\top}\left(\Gamma_n^{l,k}\right)^{-1}\mathbf{v}^{l,k}\right), \quad (7)$$

where $\Gamma_n^{l,k}$ is the covariance matrix of $\mathbf{v}^{l,k}$. Also, from Fig. 4(b), we observe that this assumption is suitable to describe the trace in Google cluster. According to Equ. (6) and Equ. (7), we can derive that

$$L^{l,k}\left(\mathbf{v}^{l,k}|\sigma^{l,k}\right)=-\frac{1}{2}\left(T\ln 2\pi+\ln\left|\Gamma_n^{l,k}\right|\right)-\frac{1}{2}\mathbf{v}^{l,k\top}\left(\Gamma_n^{l,k}\right)^{-1}\mathbf{v}^{l,k} \quad (8)$$

Take the partial derivative of $L^{l,k}\left(\mathbf{v}^{l,k}|\sigma^{l,k}\right)$, we obtain

$$\frac{\partial L^{l,k}\left(\mathbf{v}^{l,k}|\hat{\sigma}^{l,k}\right)}{\partial\hat{\sigma}_t^{l,k}}=0,\ t=1,2,...,T. \quad (9)$$

or equivalently,

$$\frac{1}{\left|\Gamma_n^{l,k}\right|}\frac{\partial\left|\Gamma_n^{l,k}\right|}{\partial\hat{\sigma}_t^{l,k}}+\frac{\mathbf{v}^{l,k\top}}{\left(\Gamma_n^{l,k}\right)^2}\frac{\partial\Gamma_n^{l,k}}{\partial\hat{\sigma}_t^{l,k}}\mathbf{v}^{l,k}=0,\ t=1,2,...,T. \quad (10)$$

After solving Equ. (10), we can obtain the value of $\sigma_T^{l,k}$

$$\hat{\sigma}_T^{l,k}=\frac{\sum_{t=1}^{T}\left(\mathbf{v}^{l,k}\right)^2}{T} \quad (11)$$

which is the prediction error variance for the demand $v_l$'s $k^{\text{th}}$ resource.

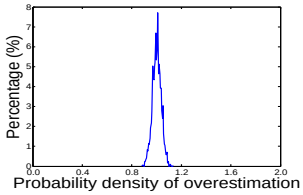



Fig. 4. Overestimation and underestimation are equally likely to happen.

## IV. Demand Allocation

Based on the predicted demands and the prediction errors calculated in Section III, in this section, we focus on deciding which servers to reserve and how to allocate demands to handle the challenge indicated in Section I, that is, minimizing the total reservation cost and guaranteeing that the reserved servers can satisfy the requirement of each demand with the probability no smaller than $1-\varepsilon$. In Section V, we introduce the rationale of our demand allocation method. In Section V-A, we formally formulate this demand allocation problem, namely the *probabilistic demand allocation (PDA)* problem and prove the NP-hardness of this problem. In Section V-B, we introduce how to solve this problem.

## V. Rationale of the Demand Allocation Method

Using the previous demand prediction method, we first study whether demand overestimation and underestimation co-exist and have similar probability to occur.

We run our prediction method on the CPU utilization of 3000 demands in Google cluster [13], and measure the ratio of overestimation over underestimation in the prediction of each demand. From the test, we found that this ratio is no larger than 2. Then, we partition the interval $[0,2]$ to 200 intervals: $[0,0.01)$, $[0.01,0.02)$, ..., $[1.99,2.00]$. Fig. 4 shows the percentage of demand ratios in each interval. We find that over 95% of the demands have the ratios in $[0.95,1.05)$, which means that overestimation and underestimation are equally likely to happen for each demand in most cases. Then, we test whether the distributions of the magnitude of overestimation and underestimation of each demand are similar. Specifically, we partition the prediction error interval $[-0.15,0.15]$ to 300 intervals evenly. For each interval, we count the percentage of the prediction errors that reside in each interval for randomly chosen demands. We observe that the magnitudes of overestimation and underestimation have similar distributions over these intervals for each demand as shown in Fig. 4(b).

Previous methods (e.g., CloudScale) achieve high SLA guarantee at the cost of under-utilization of resources. They always add a padding on predicted demands to handle underestimation. However, they omit the impact of overestimation. PDA considers both underestimation and overestimation. It aims to allocate demands with similar underestimation and overestimation together in a server, so that the underestimation and overestimation can offset each other and hence the server resources are less likely to be either overutilized or underutilized.

### A. Problem Formulation and Analysis

Recall that the scenario we have considered is composed of a CSB, $M$ demands $V=\{v_1,v_2,...,v_M\}$, and $N$ heterogeneous servers $S=\{s_1,...,s_N\}$. At time $T$, the allocator of each CSB needs to allocate the demands to the servers. Using the prediction method described in Section III, CSBs get the *estimated demand vector* and the *prediction error vector* for each demand $v_l$, represented by $\hat{\mathbf{w}}_T^l=[w_T^{l,1},...,w_T^{l,K}]^\top$ and $\hat{\sigma}_T^l=[\hat{\sigma}_T^{l,1},...,\hat{\sigma}_T^{l,K}]^\top$, respectively.

Recall that $y_{i,l}$ represents whether demand $v_l$ is distributed to server $s_i$. Then, the sum demand consumption for resource $k$ in $s_i$ is calculated by $\omega_{k,T}^\top \mathbf{y}_i$, where $\omega_{k,T}=[w_T^{1,k},...,w_T^{M,k}]^\top$. Also, let $\hat{\omega}_{k,T}=[\hat{w}_T^{1,k},...,\hat{w}_T^{M,k}]^\top$ and $\hat{\sigma}_{k,T}=[\hat{\sigma}_T^{1,k},...,\sigma_T^{M,k}]^\top$. We use $\mathbf{c}^\top\mathbf{x}=\sum_i c_i x_i$ to represent the total reservation cost of servers. Our objective is to minimize the cost of all the reservation cost of servers and satisfy each demand with probability no smaller than $1-\varepsilon$. Accordingly, we formulate the PDA problem as follows:

$$\min \quad \mathbf{c}^\top\mathbf{x} \tag{12}$$

$$\text{s.t.} \quad \Pr\left(\omega_{k,T}^\top\mathbf{y}_i \le b_{i,k}x_i\right) \ge 1-\varepsilon,\ \forall i,k \tag{13}$$

$$\mathbf{y}^j\mathbf{1}=1 \tag{14}$$

where $\omega_{k,T}^\top$ is random vector that follows normal distribution with mean 0 and variance $[\hat{\sigma}_T^{1,k},...,\hat{\sigma}_T^{M,k}]$. The first constraint (Equ. (13)) means that, for each server $s_i$ and resource $k$, the probability that the resource consumption of demands assigned to $s_i$ does not exceed $s_i$'s capacity on resource $k$ is no smaller than $1-\varepsilon$. The second constraint (Equ. (14)) means that each demand should be allocated to exact one server.

The input of the problem is the predicted demand at period $T$, say $\hat{\mathbf{w}}_T^l$ and $\hat{\sigma}_T^l$, and capacity vector of each server $s_i$, say $\mathbf{b}_i$. The output of the algorithm is the solution of PDA $\mathbf{z}$, where $\mathbf{z}=[\mathbf{z}_1,...,\mathbf{z}_N]^\top$ and $\mathbf{z}_i=[x_i,y_{i1},y_{i2},...,y_{iM}]^\top$, which contains all the information of $x_i$ and $y_{i,l}$. Here $x_i$ indicates whether $s_i$ is reserved and $y_{i,l}$ indicates whether $v_l$ is allocated to $s_i$. Then the allocator determines which server should be reserved and how to allocate the demands to reserved servers according to $x_i$ and $y_{i,l}$.

**Proposition 5.1:** The PDA problem can be expressed as the SOCMIP problem [22]

$$\min \quad f(\mathbf{z})=\sum_{i=1}^{N}\mathbf{c}_i^\top\mathbf{z}_i \tag{15}$$

$$\text{s.t.} \quad g_i(\mathbf{z})=\phi(\varepsilon)\|\Sigma_k'^{1/2}\mathbf{z}_i\|-\mathbf{b}_{i,k}^\top\mathbf{z}_i\le 0, \tag{16}$$

$$i=1,...,N,\ k=1,...,K \tag{17}$$

$$h(\mathbf{z})=E^\top\mathbf{z}-\mathbf{1}=0, \tag{18}$$

where each $\Sigma_k'$ is determined by the input $\hat{\sigma}^{k,T}$ and each $\mathbf{b}_{i,k}$ is determined by the input $\omega_{k,T}$, $\mathbf{c}_i$ is an $N$ dimensional vector such that the first entry equals $c_i$ $(i=1,...,N)$ and all other entries equals 0, $E=[\mathbf{e}_1,...,\mathbf{e}_M]$, and $\mathbf{e}_j$ is a $(M+1)N$ dimensional vector such that the $j$th entry equals 1 and all other entries equal to 0.

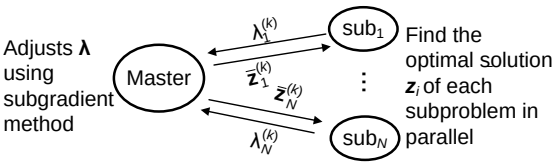



Fig. 5. Decentralized subgradient method at the $k^{\text{th}}$ iteration.

### *B. Decentralized Subgradient Algorithm*

SOCMIP has been proved NP-hard [15]. Hence, it is impossible to get the optimal solution of SOCMIP within polynomial time. As a solution, we first relax the feasible region of SOCMIP from integers to real numbers. However, even for the relaxed SOCMIP, the subgradient algorithm is still not time-efficient. Considering the scalability of our system, we need to find a way to realize the algorithm in a decentralized way. In the following, we design a decentralized method based on the Lagrangian dual decomposition, which is a classical method in combinatorial optimization and has been widely applied to distributed and parallel computation [15]. More specifically, the decentralized method is composed of three steps, where the first two steps (Lagrangian dual decomposition) decompose the problem and the third step solves the problem:

1. We derive a dual problem of the relaxed SOCMIP, denoted by *DSOCMIP* (Equ. (19) - Equ. (22)).
2. We decompose the dual problem into a set of subproblems (Equ. (23) - Equ. (28)).
3. We use subgradient to get the solutions of all the subproblems, and combine and adjust all these solutions to get the dual solution and primal solution, and then round the primal solution to get $\mathbf{z}$.

**Step 1. Creating a dual problem of the relaxed SOCMIP.** To create the dual problem of the relaxed SOCMIP, we follow the steps defined by the Lagrangian dual decomposition (Equ. (19) - Equ. (28)). We first define the Lagrangian function $\Lambda : \mathbb{Z}^{N(M+1)} \times \mathbb{R}^N \times \mathbb{R} \to \mathbb{R}$ as follows. In this function, the vector $\lambda = [\lambda_1, ..., \lambda_N]$ is called the Lagrange multiplier vector and $\nu$ is called Lagrange multiplier ($\lambda \in \mathbb{R}^N$, $\lambda \succeq \mathbf{0}$ and $\nu \in \mathbb{R}$) associated with SOCMIP:

$$\begin{aligned}\Lambda(\bar{\mathbf{z}}, \lambda, \nu) &= \sum_{i=1}^{N} \mathbf{c}_i^\top \bar{\mathbf{z}}_i + \sum_{i=1}^{N} \lambda_i g_i(\bar{\mathbf{z}}) + \nu h(\bar{\mathbf{z}}) \\ &= \sum_{i=1}^{N} \left( \mathbf{c}_i^\top \bar{\mathbf{z}}_i + \lambda_i \left( \|\Sigma_i^{1/2} \bar{\mathbf{z}}_i\| \right) - \mathbf{b}_i^\top \bar{\mathbf{z}}_i \right) \quad [i] \\ &+ \nu \left( E^\top \bar{\mathbf{z}} - \mathbf{1} \right) \quad [ii] \end{aligned} \tag{19}$$

Then, we define the Lagrangian dual function by

$$\Theta(\lambda, \nu) = \inf \Lambda(\bar{\mathbf{z}}, \lambda, \nu). \tag{20}$$

Therefore, DSOCMIP is to find the solution

$$\begin{aligned} \max \quad & \Theta(\lambda, \nu) && (21) \\ \text{s.t.} \quad & \lambda \succeq \mathbf{0} && (22) \end{aligned}$$

**Step 2. Decomposing the dual problem of the relaxed SOCMIP.** Note that the Lagrangian function in Equ. (19) has two parts (indicated by [i] and [ii]). Accordingly, we decompose the problem to two parts denoted by $\Lambda_0(\bar{\mathbf{z}}, \nu)$ and $\Lambda_i(\bar{\mathbf{z}}_i, \lambda_i)$ $(i = 1, ..., N)$, respectively.

$$\Lambda_0(\bar{\mathbf{z}}, \nu) \triangleq \nu \left( E^\top \bar{\mathbf{z}} - \mathbf{1} \right) \tag{23}$$

$$\Lambda_i(\bar{\mathbf{z}}_i, \lambda_i) \triangleq \mathbf{c}_i^\top \bar{\mathbf{z}}_i + \lambda_i \left( \|\Sigma_i^{1/2} \bar{\mathbf{z}}_i\| - \mathbf{b}_i^\top \bar{\mathbf{z}}_i \right) \tag{24}$$

Then, we define sub-problem functions $\Theta_0(\lambda)$ and $\Theta_i(\lambda_i)$ by $\Theta_0(\nu) \triangleq \inf\{\Lambda_0(\bar{\mathbf{z}}, \nu)\}$ and $\Theta_i(\lambda_i) \triangleq \inf\{\Lambda_i(\bar{\mathbf{z}}_i, \lambda_i)\}$. Hence, the Lagrangian dual function can be written as:

$$\Theta(\lambda, \nu) = \inf_{\bar{\mathbf{z}} \in \mathscr{D}} \Lambda(\bar{\mathbf{z}}, \lambda, \nu) = \Theta_0(\nu) + \sum_{i=1}^{N} \Theta_i(\lambda_i). \tag{25}$$

where $\mathscr{D} = \{\bar{\mathbf{z}} \mid 0 \le \bar{x}_i \le 1, 0 \le \bar{y}_{ij} \le 1, 1 \le i \le N, 1 \le j \le M\}$. As for $\Theta_0(\nu)$, if $E^\top \bar{\mathbf{z}} - 1 < 0$, then setting $\nu = \infty$ leads to the minimum value of $\Theta_0(\nu)$ to be $-\infty$; if $E^\top \bar{\mathbf{z}} - 1 > 0$, then setting $\nu = -\infty$ leads to the minimum value of $\Theta_0(\nu)$ to be $-\infty$. Hence, in the above two cases, it is impossible for $\Theta_0(\nu)$ to get its maximum value. Accordingly, we only consider the case that $E^\top \bar{\mathbf{z}} - 1 = 0$, where

$$\Theta_0(\nu) \triangleq \inf_{\bar{\mathbf{z}} \in \mathscr{D}_0} \{\Lambda_0(\bar{\mathbf{z}}, \nu)\} = \inf_{\bar{\mathbf{z}} \in \mathscr{D}_0} \{E^\top \bar{\mathbf{z}} - 1\} = 0 \tag{26}$$

where $\mathscr{D}_0 = \mathscr{D} \cap \{\bar{\mathbf{z}} \mid \bar{\mathbf{z}} - 1 = 0\}$. Then, $\Theta(\lambda, \nu) = \sum_{i=1}^{N} \Theta_i(\lambda_i)$. Note that all $\Theta_i(\lambda_i)$ can be evaluated independently, i.e., in parallel. Hence, DSOCMIP can be decomposed into a set of subproblems $\mathsf{sub}_i$ $(i = 1, ..., N)$:

$$\begin{aligned} \max \quad & \Theta_i(\lambda_i) && (27) \\ \text{s.t.} \quad & \lambda_i \succeq \mathbf{0} && (28) \end{aligned}$$

**Step 3. Finding the solution of SOCMIP**. To solve the subproblems to get the final problem solution, we use a decentralized subgradient method which finds the result by combining and adjusting the solutions from all the subproblems and is guaranteed to converge to the optimal values provided that the step sizes of subgradient are sufficiently small [15]. We first define the objective function of the *master* problem by $\max \sum_i \Theta_i(\lambda_i)$. As shown in Fig. 5, the decentralized algorithm collects and compares the subproblems' solutions and sends feedback to subproblems to adjust the solutions if conflicts exist. After getting the solution of the relaxed SOCMIP $\bar{\mathbf{z}}$, we obtain $\mathbf{z}$ by rounding $\bar{\mathbf{z}}$.

## VI. Performance Evaluation

We conducted both simulation and real-world experiments (on Amazon EC2 [16]) driven by the Google Cluster [13] (introduced previously). We first evaluated the effectiveness of our prediction algorithm with the prediction method in CloudScale [11], which is the most advanced prediction method for cloud computing. CloudScale estimates the magnitude of underestimation from prediction using FFT and adds a padding on predicted demands to avoid the impact of underestimation. We then evaluated the effectiveness of our demand allocation algorithm in comparison with two typical demand allocation algorithms, BFDSum (or BFD for short) and FFDSum (or FFD

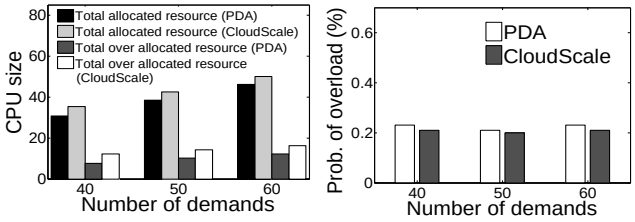
for short) [8]. In both BFD and FFD, all servers' capacity vectors and demands' consumption vectors are mapped into singular scales, called volumes and weights, respectively. Both BFD and FFD sort the demands in decreasing order of size at the beginning, and then start the allocation from the first demand. The difference is that, given a demand to allocate, BFD iterates over all the servers and allocates the demand into the server with the least remaining volume, while FFD just allocates the demand into the first server in the server set $S$ with sufficient volume. The objective of both FFD and BFD is to minimize the number of servers reserved rather than the total server reservation cost. We extended them to *weighted BFD (BFD-w)* and *weighted FFD (FFD-w)*, respectively, that consider servers' different weights (prices) when allocating a demand to a server. Specifically, for BFD-w, we define a metric $\beta_i$ for each server $s_i$, which equals the product of $s_i$'s remaining volume and $s_i$'s price. In each iteration, BFD-w chooses the server with the minimum $\beta_i$ to allocate the demand. FFD-w always allocates the demand to the server that has the lowest price among the servers with sufficient remaining volume. All these compared methods use CloudScale without padding (i.e., FFT) as their demand prediction method. For each demand in the trace of Google Cluster, we used first two days' utilization records for prediction and the utilization records of the first five minutes in the third day for testing. Finally, we assume that there are three types of servers for the simulation and the experiments on Amazon EC2, as shown in Table I.

TABLE I
THREE TYPES OF SERVERS

| Type | Memory | CPU | Price | Quantity |
|---|---|---|---|---|
| Standard | 15 GB | 8 units | 0.7 | 20 |
| High-Memory | 17.1 GB | 6.5 units | 0.6 | 20 |
| High-CPU | 7 GB | 20 units | 1.0 | 20 |

For prediction algorithms, the metrics we measured include: 1) *Total allocated resource*, which is defined as the total amount of all the allocated resource (i.e., capacity of reserved servers). 2) *Total over allocated resource*, which is calculated as the total allocated resource minus the total amount of predicted demands. It includes padding and reserved resource that may not be used in the server and a smaller value implies higher utilization of the resources of reserved serves. 3) *Probability of overload*, which is defined as the percentage of periodical recoding times that the actual total demand exceeds the total allocated resource. In addition, in the following experiments, we normalized the demands and servers by dividing the CPU and memory by 20 units and 17.1GB, respectively, which are the maximum CPU and maximum memory of all the servers (shown in Table. I). In the following, the metrics of total allocated resource and total over allocated resource are normalized.

For demand allocation algorithms, the metrics we measured include: 1) *Total reservation cost*, which is defined as the total fee that the CSB needs to pay to all the cloud providers; 2) *CPU overload rate*, which is defined as the percentage of time that a server's CPU is overloaded; 3) *SLA violation rate*, which is defined as the percentage of demands that are not satisfied with probability $1-\varepsilon$ during the testing time, and 4) *CPU/memory utilization*, which is defined as the percentage of a server's CPU/memory capacity that is actually consumed.




(a) Prediction for CPU (b) Prob. of CPU overload

Fig. 6. Demand prediction for Google Cluster

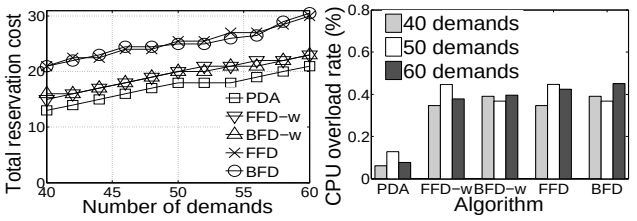
## A. Demand Prediction

We first compare the performance of the prediction methods in PDA and CloudScale. CloudScale determines the servers and allocates the demands to the servers using FFD based on the sum of its predicted value and the padding value for each demand. To compare the prediction methods of PDA and CloudScale, we also let PDA use FFD to choose servers and allocate demands to the servers. In particular, PDA first sorts the demands to be inserted in decreasing order by their expected capacity, and then inserts each demand into the first server in the list with sufficient remaining space.

Fig. 6(a) and Fig. 6(b) show the performance of PDA and CloudScale using the Google cluster CPU trace. In particular, Fig 6(a) compares the total allocated resource and the total over allocated resource and Fig 6(b) shows the probability of overload. From the figures, we find that 1) PDA needs less total allocated resource than CloudScale, 2) the total over allocated resource of PDA is smaller than that of CloudScale, and 3) the probabilities of overload of PDA and CloudScale are similar. Even though PDA aims to minimize the reservation cost of servers rather than minimizing the total allocated resource, which is the goal of FFD used in CloudScale, PDA still generates less total allocated resource while produces similar probability of overload as CloudScale. This is because PDA can more accurately estimate the prediction errors. Specifically, when allocating demands, PDA considers both overestimation and underestimation from the predictions, which can offset each other and lead to less prediction errors. However, CloudScale neglects the impact of overestimation, leading to larger prediction errors. This is why PDA produces smaller total over allocated resource (hence higher server resource utilization) than CloudScale as shown in Fig 6(a)-(d).

## B. Demand Allocation

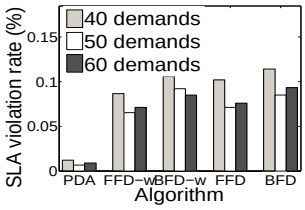
*1) Trace-driven Simulation:* Fig 7(a) shows the total server reservation cost when the number of demands is varied from 40 to 60 with 2 increase in each step. We observe that the result follows FFD≈BFD>FFD-w≈BFD-w>PDA. Recall that, when selecting a server, all BFD-w, FFD-w, BFD, and FFD simply map all the capacity vectors and consumption



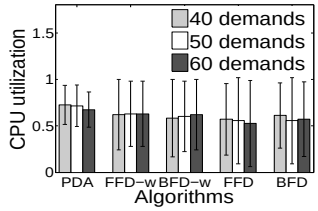

(a) Total reservation cost



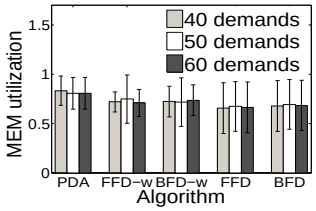

(b) CPU overload rate




(c) SLA violation rate

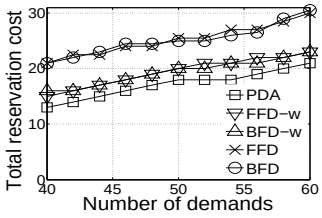



(d) CPU utilization

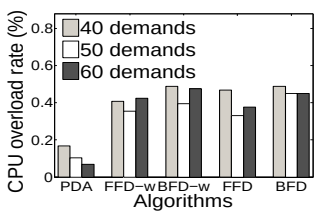



(e) Memory utilization

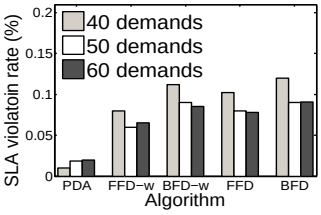
Fig. 7. Demand allocation for Google cluster (Simulation).



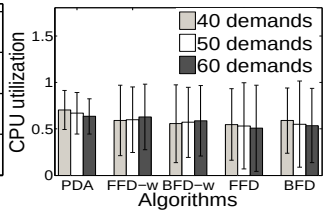

(a) Total reservation cost



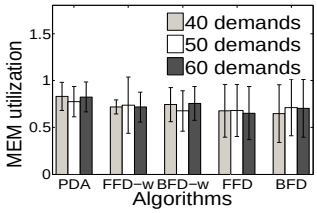

(b) CPU overload rate




(c) SLA violation rate




(d) CPU utilization




(e) Memory utilization

Fig. 8. Demand allocation for Google cluster (Amazon EC2).

vectors to single scalars without considering the bottleneck of the resource utilization for each server, which decreases each server's remaining capacity for allocating more demands. PDA has better performance because it aims to search the demand allocation such that the total server cost is minimized. Also, the prediction method in BFD, FFD, BFD-w, and FFD-w does not consider the overestimation of prediction, and hence they cannot fully utilize the resource of reserved servers comparing to PDA, which can estimate the prediction errors before the allocation. Comparing to FFD and BFD, FFD-w and BFD-w have smaller total reservation cost because FFD-w and BFD-w always choose the server with lower price to allocate each demand, while FFD and BFD do not consider the server prices at all. Since PDA uses the least server reservation cost to support a given amount of demands, we are interested in checking if it generates many server overload occurrences. Fig. 7(b) and Fig. 7(c) show the CPU's overload rate and the SLA violation rate of different algorithms, respectively. We observe that for these two metrics, the result follows FFD≈BFD≈FFD-w≈BFD-w>PDA. PDA has much smaller CPU's overload rate because when allocating a demand to a server, PDA allocating the demands with the consideration of prediction errors. In contrast, all other four methods predict the demand using FFT without considering prediction errors, which generates larger number of CPU overload. PDA has almost zero SLA violation rate because it satisfies the constraint (13) when allocating demands, which guarantees that the probability of overload rate is lower than $\varepsilon$, i.e., the SLA requirement. Other four algorithms have much higher SLA violation rates because they do not have such constraint when allocating demands and they do not consider the prediction errors, which leads to more resource overloads.

Furthermore, we measured the CPU and memory utilizations every 5 minutes for all servers. Fig 7(d) and Fig 7(e) show the median, 5th and 95th percentile of the CPU and memory utilizations at all time points of all servers of the five algorithms with 40, 50, and 60 demands, respectively. In both Fig 7(d) and Fig 7(e), we observed that the median utilization follows: FFD≈FFD-w≈BFD≈BFD-w<PDA, which indicates that PDA can more fully utilize the resources of servers and hence save the total reservation cost. The reason is the same as in Fig 7(a).

*2) Trace-driven Real-world Experiments on Amazon EC2:* We carried out experiments in a cluster built from Amazon EC2 US East Region. We applied the Google Cluster trace to a synthetic load generator *lookbusy* [23] to emulate the real workload in the VMs. Our algorithm uses the utilization trace to predict the demands and then places each demand to an appropriate reserved server. After allocating the demands to the reserved servers, we measured the real resource consumptions of each server.

Fig. 8 shows the performance of the five algorithms implemented in Amazon EC2 using Google Cluster trace. Comparing Fig. 7 and Fig. 8, we have the following observations: (1) in both Fig. 7(a) and Fig. 8(a), the total reservation cost of servers follow: FFD≈BFD>FFD-w≈BFD-w>PDA, (2) in both Fig. 7(b)-(c) and Fig. 8(b)-(c), the CPU overload rate and the SLA violation rate follow:

FFD≈BFD≈FFD-w≈BFD-w>PDA, and (3) in all Fig. 7(d)-(e) and Fig. 8(d)-(e), both CPU utilization and memory utilization follow FFD≈BFD≈FFD≈BFD<PDA. The results in Fig. 8 confirm that PDA more fully utilizes resources in each server without overloading them and hence saves total reservation cost in Amazon EC2.

## VII. Related Work

There has been two groups of works studying the demand allocation problems. The first group of demand allocation works [8]–[10], [24] aim to find an allocation schedule that minimizes the number of allocated servers while satisfy all demands. For example, Wood *et al.* [10] reduced the number of servers by enabling live migration of VMs using FFD, by taking the product of CPU, memory, and network loads. Tang *et al.* [9] developed an application placement controller for data centers, which performs demand assignment by combining the CPU and memory into a scalar by taking the ratio of the CPU demand to the memory demand. Srikantaiah *et al.* [8] proposed to use Euclidean distance between resource demands and residual capacity as a metric for consolidation, a heuristic analogous to Norm-based Greedy. Michael and Zaourar [24] modified the traditional BFD algorithm by minimizing the bin's remaining capacity variance when selecting a bin to fit an item, which is similar to our strategy. However, all these works are based on the assumption that the demands are known, which is not practical in real world.

The second group of demand allocation works predict the demands before allocating the demands to servers [11], [17], [19]. For example, [19] predicts the seasonal period using FFT, which decomposes the demands into a number of periodic series and eliminates the periodic series with high frequencies. Niu *et al.* [17] predicted the demand using the auto-regression moving average (ARMA) model, which is a widely used prediction model for time series. However, all these works cannot accurately predict tenant demands since they neglect prediction errors. A recent work, CloudScale [11], takes into account the underestimation of demands in prediction and adds padding on demands to avoid underestimation. However, it omits the case that the demands can be also overestimated, which leads to under-utilization of server resources. Further, all these methods to estimate the seasonal period are not time-efficient [12] and they are not suitable for predicting demands that do not exhibit seasonal periods (e.g. some demands in Google cluster [13]).

## VIII. Conclusions

In this paper, we proposed a demand allocation system for CSB, called *CSB demand allocation (PDA)* system, which aims to determine how to reserve servers and allocate demands into the reserved servers, such that all the demands from tenants can be satisfied with a given probability and the total reservation cost is minimized. PDA is composed of two parts: 1) prediction and 2) demand allocation. In the prediction part, we proposed a predictive model that can dynamically predict different types of resources of demands based on historical data. In the demand allocation part, given the prediction results from prediction part, we formulated a probabilistic demand allocation problem. We showed that PDA can also be formulated as the SOCMIP problem, which can be solved by the *rounding cuts* method. Both simulation and real-world experimental results demonstrate the superior performance of our system in achieving the objective in comparison with some previous methods. In our future work, when predicting the demands, we will further consider the correlation among the demands from different tenants and different types of resources. In addition, we will consider the demand migration among different servers.

## Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, IBM Faculty Award 5501145 and Microsoft Research Faculty Fellowship 8300751.

## References

[1] Gartner, "http://www.gartner.com/."
[2] Z. C. Computing, "http://www.zimory.com/."
[3] 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.
[4] A. C. Compute, "http://aws.amazon.com/ec2/hpc- applications/," 2011.
[5] D. Niu, C. Feng, and B. Li, "A theory of cloud bandwidth pricing for video-on-demand providers.," in *Proc. of INFOCOM*, 2012.
[6] L. Chen and H. Shen, "Consolidating complementary vms with spatial/temporal-awareness in cloud datacenters.," in *Proc. of Infocom*, 2013.
[7] S. Lee, R. Panigrahy, V. Prabhakaran, V. Ramasubramanian, K. Talwar, L. Uyeda, and U. Wieder, "Validating heuristics for virtual machines consolidation," 2011.
[8] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy aware consolidation for cloud computing.," in *Proc. of HotPower*, 2008.
[9] C. Tang, M. Steinder, M. Spreitzer, and G. Pacifici, "A scalable application placement controller for enterprise data centers.," in *Proc. of WWW*, 2007.
[10] 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.
[11] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: Elastic resource scaling for multi-tenant cloud systems.," in *Proc. of SOCC*, 2011.
[12] R. N. Bracewell, *The Hartley Transform*. Oxford Univ. Press, New York, 1986.
[13] G. cluster data., "https://code.google.com/p/googleclusterdata/."
[14] S. M. Ross, *Introduction to Probability Models, 8th Edition*. Amsterdam: Academic Press, 2003.
[15] M. Bazaraa, H. Sherali, and C. Shetty, "Nonlinear programming: Theory and algorithms.," *Wiley Interscience*, 2006.
[16] "Amazon EC2." http://aws.amazon.com/ec2.
[17] D. Niu, Z. Liu, B. Li, and S. Zhao, "Demand forecast and performance prediction in peer-assisted on-demand streaming systems.," in *Proc. of Infocom*, 2011.
[18] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, *Time Series Analysis: Forecasting and Control*. WILEY, 2008.
[19] P. X. Gao, A. R. Curtis, B. Wong, and S. Keshav, "Its not easy being green.," in *Proc. of Sigcomm*, 2012.
[20] R. N. Bracewell, "The fast hartley transform.," in *Proc. of IEEE*, 1986.
[21] H. S. Hou, "The fast hartley transform algorithm algorithm.," 1987.
[22] F. TA and R. MG, "Greedy randomized adaptive search procedures.," *Journal of global optimization*, 1995.
[23] "lookbusy." http://devin.com/lookbusy/.
[24] M. G. S. Zaourar, "Variable size vector bin packing heuristics application to the machine reassignment problem.," *Distributed, Parallel, and Cluster Computing*.