CORP: Cooperative Opportunistic Resource Provisioning for Short-Lived Jobs in Cloud Systems

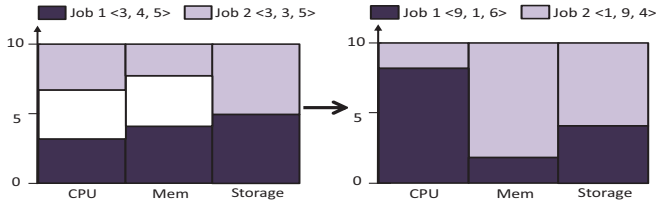
Jinwei Liu*, Haiying Shen† and Liuhua Chen*
*Department of Electrical and Computer Engineering, Clemson University, Clemson, SC 29634, USA
†Department of Computer Science, University of Virginia, Charlottesville, VA 22904, USA
{jinweil, liuhuac}@clemson.edu, hs6ms@virginia.edu

***Abstract*—In cloud systems, achieving high resource utilization and low Service Level Objective (SLO) violation rate are important to the cloud provider for high profit. For this purpose, recently, some methods have been proposed to predict allocated but unused resources and reallocate them to long-running service jobs. However, the accuracy of their prediction method relies on the existence of patterns in jobs' resource utilization. Therefore, these methods cannot be used for short-lived jobs, which usually do not have certain patterns but exhibit frequent fluctuations in resource requirements. Also, these methods may result in resource fragmentation and lead to low resource utilization because they neglect job resource intensity in multi-resource allocation and may allocate much more resources to jobs. To handle this problem, we propose a Cooperative Opportunistic Resource Provisioning scheme (CORP) for short-lived jobs. CORP uses the deep learning method to predict the amount of temporarily-unused resource of each short-lived job. It also predicts the fluctuations of the amount of unused resource using Hidden Markov Model, and adjusts the predicted amount for the peak and valley of unused resource, and dynamically allocates the corrected amount of resource to jobs. Further, CORP uses a job packing strategy by leveraging complementary jobs' requirements on different resource types and allocates such jobs to the same VM to fully utilize unused resources, which increases resource utilization. Extensive experimental results based on a real cluster and Amazon EC2 show that CORP achieves high resource utilization and low SLO violation rate compared to previous resource provisioning schemes.**

I. Introduction

Cloud computing, as a paradigm for the on-demand provision of virtualized resources, attracts many interests. Cloud providers typically offer resources for leasing with elastic infrastructure as a service (IaaS) paradigm. In order to maximize the profit, the cloud provider aims to achieving high resource utilization and low Service Level Objective (SLO) violation rate, which, however, are mutually contradictory to each other. Higher resource utilization (i.e., lower allocated resources to jobs) leads to lower SLO violation rate and vice versa. Although the elastic and on-demand nature of cloud computing enables cloud users to meet their dynamic and fluctuating demands (i.e., low SLO violation rate) with minimal management overhead, users usually are allocated more resources than their jobs' demand, resulting in resource wastage [1], [2].

Currently, the resource allocation in a cloud is either reservation-based or demand-based to achieve low SLO violation rate. In reservation-based resource allocation, the cloud reserves resources for each user [3]. In demand-based resource allocation, the cloud imposes a small resource ceiling for each user, allows users to scale on demand, and copes with workload variations [4]. Reserved resources ensure long-term availability (e.g., over a year), and on-demand resources ensure short-term availability (e.g., hours) [5]. A user usually does not fully utilize the reserved resources or the imposed ceiling. Even if its peak resource demand reaches the reserved resources or the imposed ceiling, its average resource requirement is much lower than the peak usually [6]. To further increase the resource utilization, another approach [4], [7], [8] reallocates temporarily-unused resources to new jobs in an opportunistic manner with no or weaker SLO guarantee. These resources do not guarantee availability.



(a) Without avoiding fragmentation (b) Avoiding fragmentation

Fig. 1: Allocate resource to jobs by leveraging complementary jobs' requirements on different resource types to increase resource utilization.

The method in [4] uses time series forecasting to predict the amount of resources that will remain unused during multi-month periods. However, such a prediction method assumes that resource utilization pattern exists in training data of resource utilization of the jobs. Though this assumption may be true for long-lived jobs, it usually does not hold true for short-lived jobs, in spite of the fact that short-lived jobs occupy most of the jobs in the cloud [9]. Short-lived jobs, such as short-lived queries in the applications of Internet-of-Things and online data processing, typically run for seconds or minutes with a maximum timeout of 5 minutes [10]–[13]. Short-lived jobs usually cannot tolerate long delays and must be processed quickly. To efficiently process the short-lived jobs, continuous provisioning of sufficient resources (e.g., computing resource) is required. Therefore, it is important to ensure efficient processing on short-live jobs with sufficient resource provision, while achieving high resource utilization and low SLO violation rate. This task is challenging because short-lived jobs usually do not exhibit certain resource utilization patterns [6] and exhibit fluctuations in resource use [14]. Since existing approaches cannot directly handle this challenge, we mainly deal with this challenge in this paper. Also, the methods in [4], [7] do not consider leveraging complementarity of jobs' requirements on different resource

types (CPU-high and MEM-low, CPU-low and MEM-high) for allocating multiple resource types to jobs. Therefore, they may allocate much more resources to different intensive jobs (e.g., CPU intensive and MEM intensive) that have different resource demands on different resource types [15], which can easily result in resource fragmentation and hence low resource utilization (see Figure 1).

In this paper, we aim to design a resource provisioning scheme for short-lived jobs in cloud with high resource utilization while achieving low SLO violation rate. The key challenges include: (1) how to accurately predict the amount of temporarily-unused resources of short-lived jobs with resource fluctuations? (2) how to more fully utilize the temporarily-unused resources by considering diverse resource intensities of jobs? and (3) how to allocate the resource to short-live jobs to satisfy their time constraints. We propose a Cooperative Opportunistic Resource Provisioning method (CORP) for short-lived jobs. This method can cooperate with other methods for long-lived jobs for resource allocation in cloud systems. Since the deep learning algorithm does not require the existence of patterns in training data for accurate prediction [16] as its multiple hidden layers are capable of modeling complex data with great efficiency and it has an advantage over shallow machine learning methods [17]–[19], we use it to predict the amount of temporarily-unused resource. To handle the fluctuations of the amount of unused resource to further increase the prediction accuracy, we use Hidden Markov Model (HMM) to adjust the prediction error. In resource allocation, CORP tries to consolidates complementary jobs whose demands on multiple resources are complementary to each other in order to more fully utilize the unused resources (see Figure 1).

We summarize the contributions of this work below:

- CORP uses the deep learning method to predict the amount of temporarily-unused resource of each short-lived job, and offers an opportunistic approach to reallocate predicted unused resources in order to increase the resource utilization.
- CORP also considers the fluctuations of the amount of the unused resource caused by the peak and valley of jobs' resource demands. It first predicts the fluctuations of the amount of the unused resource using HMM, then adjusts the predicted amount for the peak and valley of unused resource, and dynamically allocates the corrected amount of resource to jobs. CORP thus can adapt well to the requirement of time-varying user demand on resources.
- CORP uses a job packing strategy by leveraging complementary jobs' requirements on different resource types (e.g., CPU, MEM) and allocates such jobs to the same VM to fully utilize unused resource, which reduces the resource fragmentation and further increases the resource utilization.

The remainder of this paper is organized as follows. Section II describes the cooperative opportunistic resource provisioning problem. Section III presents the details of the system design. Section IV presents the performance evaluation for our method. Section V reviews the related work. Section VI concludes this paper with remarks on our future work.

II. Cooperative Opportunistic Resource Provisioning Problem

The physical machines (PMs) are deployed in a cloud system, and their resources are allocated to virtual machines (VMs). The VM capacity comprises of multiple types of resource (e.g., CPU, MEM and storage) and their resources are allocated to jobs based on job workloads. In this paper, we consider the problem of allocating allocated but unused resources in VMs to jobs for achieving high resource utilization and low SLO violation rate. To increase the resource utilization, the allocated but unused resource can be reallocated to jobs with a certain probability. Also, for jobs with different resource intensities (e.g., a job with high demand on CPU and a job with a high demand on MEM), they can be allocated with the unused resources in a VM together to reduce resource fragmentation in order to further increase resource utilization.

TABLE I: Notations.

$\mathbf{J}$ A set of jobs	$r'_{ij,t}$ Unused type j resc. allocated to J_i at t
J_i The ith job in $\mathbf{J}$	$U_{j,t}$ System's utilization of type j resc. at t
l # of resc. types	$w_{j,t}$ Type j resc. wastage ratio at t
$U_{a,t}$ Utilization of all resc. at t	$w_{a,t}$ Overall resc. wastage ratio at t
N_p Total # of PMs	$r_{ij,t}$ Amount of type j resc. allocated to J_i at t
N_v Total # of VMs	$d_{ij,t}$ J_i's demand on type j resc. at t
η Confidence level	$\hat{Y}_i$ Predicted J_i's unused resc. using DNN
C_{ij} CAP of v_i's type j resc.	$\hat{\sigma}$ Estimated SD for prediction errors
n_t # of jobs submitted at t	P_{th} Prob. threshold for prediction error

Suppose there are N_v VMs, and l types of resources (e.g., CPU, memory, storage) in the system. We use v_i to denote the i-th VM and use C_{ij} to denote the capacity for the type j resource of VM v_i. Assume the time is split into slots, denoted by $\mathbf{T} = \{t_1, t_2, ...\}$. Let n_t be the number of jobs submitted at time slot t. Denote $r_{ij,t}$ ($i \in \{1,...,n_t\}, j \in \{1,...,l\}$) as the amount of the type j resource allocated to job J_i at time slot t, $r'_{ij,t}$ in $r_{ij,t}$, as the amount of unused type j resource allocated to job J_i at time slot t, $d_{ij,t}$ as job J_i's demand on type j resource at time slot t. Therefore, $r_{ij,t} = r'_{ij,t} + d_{ij,t}$. For easy reference, Table I lists the main notations used in this paper.

Hence, in the system, the resource utilization of type j resource at time slot t is

$$U_{j,t} = \frac{\sum_{i=1}^{n_t} d_{ij,t}}{\sum_{i=1}^{n_t} r_{ij,t}} \tag{1}$$

where n_t is total number of jobs submitted to the system at time slot t. The overall resource utilization for all resources at time slot t is

$$U_{a,t} = \frac{\sum_{j=1}^{l}(\omega_j \sum_{i=1}^{n_t} d_{ij,t})}{\sum_{j=1}^{l}(\omega_j \sum_{i=1}^{n_t} r_{ij,t})} \tag{2}$$

where l is the total number of resource types, ω_j is the weight for type j resource, and $\sum_j^l \omega_j = 1$. The reason for setting different weights for different resource types is that sometimes some resources are more important than other resources. For example, CPU and MEM are more important than storage because storage is not the bottleneck resource [4]. The type j resource wastage ratio at time slot t is

$$w_{j,t} = \frac{\sum_{i=1}^{n_t}(r_{ij,t} - d_{ij,t})}{\sum_{i=1}^{n_t} r_{ij,t}} \tag{3}$$

The overall resource wastage ratio for all resources at time slot t is

$$w_{a,t} = \frac{\sum_{j=1}^{l}(\omega_j \sum_{i=1}^{n_t}(r_{ij,t} - d_{ij,t}))}{\sum_{j=1}^{l}(\omega_j \sum_{i=1}^{n_t} r_{ij,t})} \tag{4}$$

CORP tries to pack jobs to allocated VMs as much as possible by minimizing the overall resource wastage ratio $w_{a,t}$ in Equ. (4), and if CORP cannot pack all jobs to allocated VMs, then CORP allocates the unallocated VMs to jobs.

Our objective is to minimize $w_{a,t}$, which will be shown in our formulated problem below.

A. Objective

Our problem of the VM resource allocation to jobs can be stated as follows.

Problem Statement: Given a certain amount of resources (e.g., CPU, MEM, etc.), resource demands of each job, resource capacity constraints of VMs, how to allocate the VM resources to jobs to achieve high resource utilization while avoiding SLO violations as much as possible?

The resource allocation problem in our work is an NP-hard problem and has high computational complexity [20], [21]. Therefore, we propose a heuristic method called CORP, which approximately achieves the same goal mentioned above. Specifically, CORP first accurately predicts the amount of temporally-unused allocated resource based on the historical data with the consideration of the non-existence of pattern and fluctuations of the amount of the unused resource. It then leverages complementarity of jobs' requirements on different resource types, and utilizes the packing strategy to allocate the unused resource to other jobs with a certain probability.

III. The Design of CORP

In the following, Section III-A presents the prediction of the amount of temporally-unused resource and Section III-B presents the unused resource allocation algorithm.

A. Prediction Process and Resource Preemption

In this paper, we use the deep learning together with HMM to accurately predict the temporally-unused resource with the consideration of the fluctuations of the amount of the unused resource, and then dynamically allocate the unused resource to users' jobs. We use L to denote the size of the window (the prediction horizon). After each time period L, we use the deep learning technique to make the predictions for the amount of the temporally-unused resource in a time window $\Delta W = (t, t+L]$, where t is the time when the prediction is made. After conducting the analysis of the Google trace from our system, we chose to make the predictions for a 1 minute window because short-lived jobs typically run minutes. We then use HMM to predict whether the amount of the temporally-unused resource will be in the peak or valley at $t+L$, based on which we adjust the predicted unused resource (e.g., CPU) by deep learning as the final predicted amount.

As shown in Figure 2, deep neural network (DNN) utilizes multiple hidden layer structure for hierarchical feature learning. The multiple hidden layers enable the composition of features from lower layers, giving the potential of modeling complex data with fewer units. Compared with other machine learning methods, deep learning has the following inherent advantages. First, deep learning only needs the raw data for training without requiring sufficient high quality and truly representative past data [22], [23]. Also, deep learning has

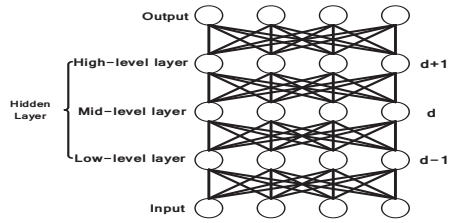


Fig. 2: Deep neural network.

better accuracy in many applications [24], [25], so it can more accurately predict the real resource demands with the same given past data. More importantly, deep learning does not require that the historical data must have patterns, which is required by the methods like fast Fourier transform [4], [26].

The prediction process of the amount of unused resource consists of three parts: 1) predicting the amount of unused resource using deep learning with HMM; 2) Prediction with confidence intervals; 3) Probabilistic-based resource preemption.

1) Predicting Unused Resource:

a) Predicting Unused Resource Using Deep Learning: We use CPU as an example to illustrate the prediction of the amount of unused resource using deep learning. Each input data contains CPU utilization of a job at each slot in last $\triangle$ slots. To build the DNN, for each input, there are three steps: feed-forward evaluation, back-propagation, weight update. Below, we introduce the details of each step.

Feed-forward evaluation: The output of each neuron i in layer d (called activation, denoted by $g_i(d)$) is computed as a function of its c inputs from neurons in the lower layer $d-1$. Let $w_{ij}(d-1,d)$ be the weight associated with a connection between neuron j in layer $d-1$ and neuron i in layer d, we have

$$g_i(d) = F((\Sigma_{j=1}^{c} w_{ij}(d-1,d) \cdot g_j(d-1)) + e_i) \quad (5)$$

where e_i is a bias term for the neuron. Equ. (5) is a sigmoid function, which is a nonlinear function associated with all neurons in the network, and is more accurate [27].

Back-propagation: For each neuron i in the output layer, the error terms E are computed using the following equation:

$$E_i(d_h) = (t_i(d_h) - g_i(d_h)) \cdot F'(g_i(d_h)) \quad (6)$$

where $t(x)$ is the true value of the output and $F'(x)$ represents the derivative of $F(x)$. Next, these error terms are back-propagated for each neuron i in layer d connected to m neurons in layer $d+1$ summed as follows:

$$E_i(d) = (\sum_{j=1}^{m} E_j(d+1) \cdot w_{ji}(d,d+1)) \cdot F'(g_i(d)) \quad (7)$$

Weight updates: The error terms are used to update the weights by using the following equation:

$$\Delta w_{ij}(d-1,d) = \mu \cdot E_i(d) \cdot g_j(d-1), \ \forall j = 1,...,c \quad (8)$$

where μ represents the learning rate parameter, and c is the number of inputs from neurons in layer $d-1$.

The process of these three steps is repeated for each input until the entire training dataset has been processed, which constitutes a training epoch. At the end of a training epoch, the model prediction error is computed as a held-out validation set. Basically, the training continues for multiple training epochs, processing the training data set each time, until the validation set error converges to a low value. Finally, the DNN is built.

The deep learning algorithm for predicting the amount of unused resource is comprised of two parts: training and testing. For training, it first computes the hidden activation. Next, it computes the reconstructed output from the hidden activation. Then the algorithm computes the error gradient, and it back-propagates error gradient to update weight. For testing, the algorithm autoencodes the input and generates the output.

After the training, the DNN is built. To predict the unused resource of a job at time $t+L$, we input CPU utilization of a job at each slot in last $\triangle$ slots to the DNN, and the output is the amount of unused CPU resource of the job. The deep learning algorithm predicts the amount of unused resources of each job J_i in a time period, denoted by $\hat{Y}_i = (\hat{r}_{i1}, ..., \hat{r}_{il})$, where $\hat{r}_{ij}$ denotes the predicted amount of unused type j resource of job J_i. The resource usage of short-lived jobs sometimes fluctuates; it reaches a peak and a valley sometimes [26], which makes the actual amount of unused resource under fluctuations cannot be accurately predicted. To handle this problem, CORP then uses the HMM model to predict the peak and valley occurrences of the unused resource for prediction error correction. We present the details of the HMM model below.

b) Predicting Fluctuations of Unused Resource Using HMM: We use CPU as an example to illustrate the process, and the method can be directly applied to other resource types. Given a set of historical data, let max_{cpu}, m_{cpu} and min_{cpu} be the maximum amount, average amount and minimum amount of unused CPU resource in the historical data, respectively. We split the interval $[min_{cpu}, max_{cpu}]$ into 3 subintervals: $[min_{cpu}, min_{cpu} + \frac{1}{2}(m_{cpu} - min_{cpu})]$, $(min_{cpu} + \frac{1}{2}(m_{cpu} - min_{cpu}), m_{cpu} + \frac{1}{2}(max_{cpu} - m_{cpu}))$, $[m_{cpu} + \frac{1}{2}(max_{cpu} - m_{cpu}), max_{cpu}]$. We call these three parts as peak, center, valley, respectively, which are used to categorize the observation symbols of the HMM model. The corresponding (hidden) states that determine the observation symbols are over-provisioning (OP), normal-provisioning (NP), under-provisioning (UP), respectively (see Figure 3) [28].

Denote $S = \{S_1, ..., S_H\}$ $(H = 3)$ as the set of states, q_t as the state at t, and $Q = q_1 q_2 ... q_T$ as a state sequence. Let $V = \{1, ..., M\}$ $(M = 3)$ be the set of possible observation symbols per state, and $O = \{O_1, ..., O_T\}$ $(O_i \in V, \forall i = 1, ..., T)$ be the observation sequence, where M is the number of observation symbols[1] (1,2,3 represent "peak", "center" and "valley" regions, respectively) and T is the length of observation sequence. To determine the observation symbols, we consider the time interval between two consecutive observation time slots j and $j+1$ $(j = 1, ..., T-1)$ as a window, and we divide the window into $L-1$ subwindows. Let Δ_j be the difference between the maximum amount of unused resource and the minimum amount of unused resource in the window. If Δ_j falls in $[min_{cpu}, min_{cpu} + \frac{1}{2}(m_{cpu} - min_{cpu})]$, then we consider the observation symbol at $j+1$ is valley; if Δ_j falls in $(min_{cpu} + \frac{1}{2}(m_{cpu} - min_{cpu}), m_{cpu} + \frac{1}{2}(max_{cpu} - m_{cpu}))$, then we consider the observation symbol at $j+1$ is center; otherwise, we consider the observation symbol at $j+1$ is peak. Then, the state transition probability matrix is

$$A = \{a_{ij}\} \ (a_{ij} = P\{q_{t+1} = S_j | q_t = S_i\}, \ 1 \le i, j \le H) \quad (9)$$

where the state transition coefficients satisfy: $a_{ij} \ge 0$ and $\sum_{j=1}^{H} a_{ij} = 1$. The observation probability matrix B is

$$B = \{b_j(k)\} \ (b_j(k) = P\{O_t = k | q_t = S_j\}, 1 \le j \le H, 1 \le k \le M) \quad (10)$$

where $b_j(k)$ is the probability that the observation symbol is k given the sate at t is S_j. Thus, the initial state distribution is

$$\pi = \{\pi_i\} \ (\pi_i = P\{q_1 = S_i\}, \ 1 \le i \le H) \quad (11)$$

Given the model $\lambda = (A, B, \pi)$ and an observation sequence O, our goal is to find the most likely state sequence. Specifically, we aim to maximize the expected number of correct states for the HMM. We define $\gamma_t(i)$ as the probability of being in state S_i at time t, given the observation sequence O and the model λ:

$$\gamma_t(i) = P\{q_t = S_i | O, \lambda\} \quad (12)$$

Equ. (12) can be simplified with the forward-backward variables as follows:

$$\gamma_t(i) = \alpha_t(i)\beta_t(i)/P(O|\lambda) \quad (13)$$

where $\alpha_t(i)$ is the forward variable defined as

$$\alpha_t(i) = P(O_1 O_2 \cdots O_t, q_t = S_i | \lambda) \quad (14)$$

where $\beta_t(i)$ is the backward variable defined as

$$\beta_t(i) = P(O_{t+1} O_{t+2} \cdots O_T | q_t = S_i, \lambda) \quad (15)$$

Based on [29], $\alpha_t(i)$ and $\beta_t(i)$ can be solved inductively.

By using $\gamma_t(i)$, we can solve for the individually most likely state q_t at time t, as

$$q_t = argmax_{0 \le i \le M-1}[\gamma_t(i)], \ 1 \le t \le T \quad (16)$$

Equ. (16) chooses the most likely state for each t to maximize the expected number of correct states. In implementation, we use Viterbi algorithm to find the single best state sequence (path), denoted by $Q^* = q_1^* ... q_L^*$, i.e., maximizing $P(Q, O|\lambda)$ which is equivalent to maximizing $P(Q|O, \lambda)$ [29], and we use the method in [30] to re-estimate the parameters A, B, π.

Based on the work [31], the probability distribution of the next fluctuation observation of the amount of unused resource can be estimated as

$$E_{P_{T+1}(k)} = \sum_{j=1}^{H} P(q_{T+1} = S_j | q_T = q_L^*) \cdot b_j(k) \ (k \in \{1, ..., M\}) \quad (17)$$

We consider the observation symbol which has the highest value of $E_{P_{T+1}(k)}$ as the observation symbol of the next time $T+1$, that is, $k|_{E_{P_{T+1}(k)} = max_{u=1}^{M}(E_{P_{T+1}(u)})}$.

Given the resource utilization of jobs, CORP uses HMM to predict the fluctuations of the amount of unused resource (i.e., peak, center, valley symbols) for the next time period. Recall that we use deep learning to perform predictions for the amount of temporally-unused resource at the end of each window L, denoted by $\hat{Y}_j$. Then, we use HMM to

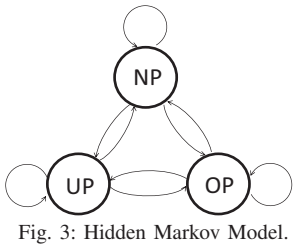


Fig. 3: Hidden Markov Model.

[1]The number of states H does not necessary equal the possible observation symbols per state M, and the HMM model in our work is a special case.

predict the fluctuations of the amount of unused resource and adjusts the predicted amount accordingly. We use $\hat{u}_{t+L}$ to represent the predicted unused resource with prediction error correction at time t for a future time $t+L$. Specifically, if the predicted observation symbol of unused CPU resource falls in the valley, CORP reduces the predicted amount by $\hat{u}_{t+L} = \hat{r}_{j1} - min(h_{cpu} - m_{cpu}, m_{cpu} - l_{cpu})$ (suppose the first resource type in $\hat{Y}_j$ is CPU), where m_{cpu} is the average value of unused CPU resource in the historical data, h_{cpu} is the highest amount of unused resource within a period, and l_{cpu} is the lowest amount of unused resource within a period. If the predicted unused CPU resource falls in the peak, CORP makes the adjustment by $\hat{u}_{t+L} = \hat{r}_{j1} + min(h_{cpu} - m_{cpu}, m_{cpu} - l_{cpu})$. The reasons for using $min(h_{cpu} - m_{cpu}, m_{cpu} - l_{cpu})$ to correct overestimation (or underestimation) errors are as follows. First, $h_{cpu} - m_{cpu}$ and $m_{cpu} - l_{cpu}$ indicate the deviation between the amount of unused resource in peak and the average of the unused resource, and the amount of unused resource in valley and the average of unused resource. The predicted amount may be close to m_{cpu}. Therefore, such adjustment can make the predicted unused resource closer to the actual amount of unused resource if it is in the peak or valley. Second, we use min because it is more conservative for ensuring sufficient resource being able to allocated to jobs.

2) Prediction with Confidence Intervals: To ensure the accuracy of the prediction, we use a confidence interval for the probability that the resource will be available. The confidence interval is an estimate of the range of values within which the true value should lie with a certain confidence level (in the form of probability denoted by η). The higher the confidence level, the wider the confidence interval, and the more conservative the predictions. The confidence interval calculation depends on the variance of the prediction errors and the confidence level η. Let $\theta = 1 - \eta$ be the significance level. The confidence interval is

$$[\hat{u}_{t+L} - \hat{\sigma} \cdot z_{\frac{\theta}{2}}, \hat{u}_{t+L} + \hat{\sigma} \cdot z_{\frac{\theta}{2}}] \tag{18}$$

where $\hat{u}_{t+L}$ is the forecast for unused resource at time t for a future time $t+L$, $\hat{\sigma}$ is the estimated standard deviation (SD) for the prediction errors, and $z_{\frac{\theta}{2}}$ is the value for the $100 \cdot \frac{\theta}{2}$ percentile in the standard normal distribution.

Based on a given confidence interval, the predicted amount of unused resource for time $t+L$ is adjusted as follows

$$\hat{u}_{t+L} = \hat{u}_{t+L} - \hat{\sigma} \cdot z_{\frac{\theta}{2}} \tag{19}$$

We use the lower bound of the confidence interval in Equ. (19) because the underestimation of the unused resource makes it conservative in reallocating allocated resources, thus avoiding SLO violations.

Based on the historical data with prediction error samples, we calculate the prediction error in a time window as follows

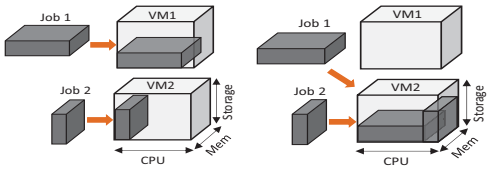
$$\delta_{t+\tau} = u_{t+\tau} - \hat{u}_{t+L}, \forall \tau \in [1, L] \tag{20}$$

That is, we calculate the prediction error for each time slot in the window $\tau \in [1, L]$ by subtracting the predicted unused resource at time t from the actual amount of unused resource at each time slot.

3) Probabilistic-based Resource Preemption: Let ε denote pre-specified prediction error tolerance and P_{th} denote a pre-defined probability threshold. For a predicted temporarily-unused resource with prediction error δ_{t+L}, if δ_{t+L} satisfies [32]

$$Pr(0 \leq \delta_{t+L} < \varepsilon) \geq P_{th} \tag{21}$$

then it can be allocated to a new arriving job, and we call it as unlocked predicted unused resource.



(a) Without packing (b) With packing

Fig. 4: Allocate the resource of VMs to the jobs W/o and W/ packing strategy.

B. Recourse Allocation Algorithm

CORP periodically predicts the allocated and unused resources in each VM. For newly arriving jobs, CORP conducts packing to pack complementary jobs, and then allocates unallocated resources to the packed jobs based on their resource demands. The job packing is used to avoid resource fragmentation and achieve high resource utilization. In the following, we first present an example to show the complementary job packing, and then explain its algorithm. Then, we present the resource allocation algorithm that reallocates unlocked predicted unused resources to newly arriving jobs. Finally, we present an example to show this algorithm.

Figure 4 shows an example illustrating how packing strategy decreases the resource fragmentation and increases resource utilization. In Figure 4(a), job 1 (CPU intensive) and job 2 (storage intensive) are assigned to VM1 and VM2, respectively, which increases VMs' resource fragmentation. However, in Figure 4(b), job 1 and job 2 are packed first and then assigned to VM2, which releases VM1, and thus decreases the resource fragmentation of VMs and increases the resource utilization.

Each job has a dominant resource, defined as the one that requires the most amount of resource. CORP first packs the jobs with complementary dominant resources such that the summation of the deviation of the two jobs' resource demands on each resource type is the largest. Given a list of jobs, CORP fetches each job J_i, and tries to find its complementary job from the list to pack with J_i. Note that it is possible that job J_i's complementary job cannot be found from the list. In this case, the job J_i solely constitutes an entity to be allocated with resources in a VM. To find J_i's complementary job, CORP calculates its deviation with every other job J_j if J_j has different dominant resource from J_i. The deviation is calculated by $DV(j,i) = \sum_{k=1}^{l}((d_{jk} - \frac{d_{jk}+d_{ik}}{2})^2 + (d_{ik} - \frac{d_{jk}+d_{ik}}{2})^2)$. Finally, the job with the highest deviation value is the complementary job of J_i.

After the job packing, CORP needs to assign each job entity (packed jobs or a job) to a VM with unlocked predicted unused resources. Among the VMs with unlocked predicted unused resources that can satisfy the resource demand of the job entity, we will choose the VM that has the least remaining resources (called most matched VM) in order to more fully utilize resources. If predicted unused resources cannot satisfy the resource demand of the job entity, unallocated resources in a VM will be used for the job entity using the same

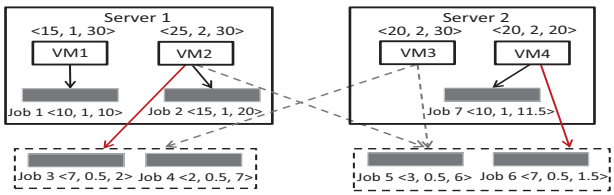


Fig. 5: Allocate unused resource to (packed) jobs with low resource wastage.

method. To find the most matched VM, we introduce a concept called unused resource volume. Suppose the vector of the maximum capacity of each resource type among all VMs is $\mathbf{C}' =< C'_1, C'_2, \ldots, C'_l >$. Assume that the amount of predicted unused resource of VM j is $\hat{R}_j = (\hat{r}_{j1}, ..., \hat{r}_{jl})$. Then, the unused resource volume of VM j is calculated by

$$volume_j = \sum_{k=1}^{l} \hat{r}_{jk}/C'_k \tag{22}$$

The VM satisfying the resource demand and has the smallest unallocated volume is the most matched VM.

Figure 5 shows an example illustrating the process of job packing and how CORP allocates the predicted unused resource to a job entity. For VMs, the numerical values indicate the capacities of different resource types. For jobs, the numerical values indicate the resource demands of jobs. Job 3, job 4, job 5 and job 6 are new arriving jobs. The dominant resource of jobs 3 and 6 is CPU, and the dominant resource of jobs 4 and 5 is storage. CORP first conducts job packing. The resource demand deviation of job 3 and job 4 is 25, and that of job 3 and job 5 is 16. Since $25 > 16$, job 3 and job 4 are packed together. Similarly, job 5 and job 6 are packed together. We denote the job entities as (job 3, job 4) and (job 5 and job 6). The maximum CPU, MEM and storage of all VMs among both servers are $\mathbf{C}' =< 25, 2, 30 >$. If the amount of unlocked predicted unused resource of VMs 1-4 are as follows: $< 5, 0, 20 >$, $< 10, 1, 10 >$, $< 20, 2, 30 >$ and $< 10, 1, 8.5 >$, respectively, based on Equ. (22), their unused resource volumes are 0.867, 1.233, 2.8, 1.183, respectively. To allocate resources to entity (job 3, job 4), CORP first checks if the VMs' predicted unused resources can satisfy the demands on each type of resource of the entity. Then, CORP chooses the VM that has the smallest unused resource volume to be allocated to the job entity. In this example, VM1 and VM4 cannot satisfy its resource requirements of the packed job (jobs 3, 4). By comparing the unused resource volumes of VM2 and VM3, because $1.233 < 2.8$, then CORP chooses VM2 rather than VM3 and allocates its temporarily-unused resource to the packed job (job 3 and job 4). Similarly, the predicted unused resource of VM1 cannot satisfy the resource requirements of the packed job (job 5, job 6). By comparing the unused resource volumes of VM2, VM3 and VM4, because $1.183 < 1.233 < 2.8$, then CORP chooses VM4 and allocates its temporarily-unused resource to the packed job (job 5 and job 6). The above process of allocating unused resource to jobs also applies to the single job case.

IV. Performance Evaluation

In this section, we present our trace-driven experimental results on a large-scale real cluster, Clemson University's high-performance computing (HPC) resource [34], and Amazon EC2 [35], respectively. To show the performance of CORP, we compared CORP with RCCR [4], CloudScale [26], DRA [36] in various scenarios since all these methods share the same objective of maximizing the resource utilization while avoiding SLO violation.

TABLE II: Parameter settings.

Parameter	Meaning	Setting	Parameter	Meaning	Setting
N_p	# of servers	30-50	h	# of layers in DNN	4 [33]
N_v	# of VMs	100-400	N_n	# of units per layer	50
$\|\mathbf{J}\|$	# of jobs	50-300	H	# of states in HMM	3
l	# of resc. types	3	θ	Significance level	5%-30%
P_{th}	Prob. threshold	0.95	η	Confidence level	50%-90%

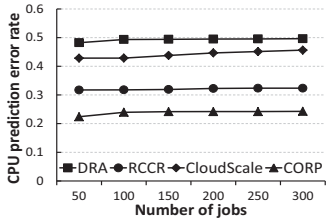
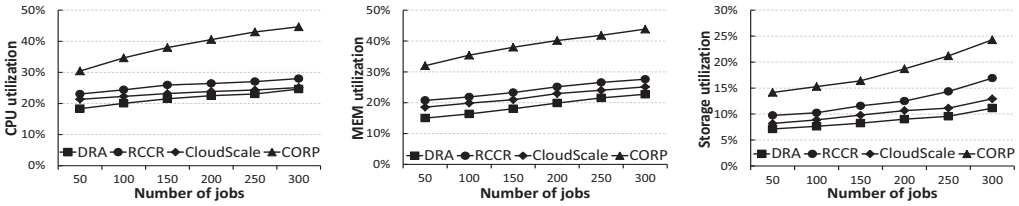


Fig. 6: Prediction error rate of different methods on a real cluster.

RCCR uses time series forecasting to predict the fraction of unused resources that will almost certainly not be required in the future based on historical resource usage patterns and allocates the unused resource to long-term service jobs in an opportunistic manner. CloudScale employs online resource demand prediction and prediction error handling to adaptively allocate the resources on PMs to VMs to achieve high resource utilization. DRA provides the cloud customer with the abstraction of buying bulk capacity (rather than pre-defined VM configurations based on the peak demands of the applications). DRA first purchases capacity for the customers, and then re-distributes the purchased capacity among customer's VMs based on their demand. Specifically, DRA considers the share value and the demand value of VMs and allocates the aggregate amount of capacity purchased by the customers among the VMs in an equitable manner taking into account shares and not giving the VMs more than what they demand.

In the implementation on the real cluster, we applied for 50 nodes, and we simulated a node as a PM, and we simulated a logic disk as a VM; in the implementation on Amazon EC2, we applied for 30 nodes, each node is simulated as a VM (It does not compromise the result much, though the setting is a little different from that in the cluster). For CORP, we first used the deep learning algorithm to predict the amount of unused resource of jobs running on the VMs based on the historical resource usage data from the Google trace. Next, we used the HMM model to predict fluctuations of the amount of unused resource of jobs, and we adjusted the predicted amount for the peak and valley of the unused resource. Then, we packed two jobs with complementary dominant resources such that the summation of the deviation of the two jobs' resource demands on each resource type is the largest (see Section III-B). Finally, we chose the VM that has the least remaining resources that can satisfy the resource demands of job(s) and allocated it to the job(s) (see Section III-B) (We know the capacity for each type of resource of a VM, and we can know the amount of unused resources of each VM after we get the amount of unused resource of jobs and the amount of resource allocated to jobs). For RCCR, we first used a



(a) CPU (b) Memory (c) Storage

Fig. 7: Utilizations of different resource types vs. number of jobs of different methods on a real cluster.

time series forecasting technique, i.e., Exponential Smoothing (ETS), to predict the amount of unused resource of VMs. Then we calculated confidence intervals and chose the lower bound of the confidence interval as the predicted value for a time window ΔW. Finally, we randomly chose a VM that can satisfy the resource demands of a job and allocated resource to the job without considering job packing. For CloudScale, we first used the prediction model developed in [37] and a discrete-time Markov chain to predict the amount of unused resource of VMs based on historical resource usage data. Then we extracted the burst pattern to get the padding value and calculated the prediction errors by subtracting the predicted amount of unused resource from the actual amount of unused resource. Next, we used the adaptive padding that is based on the recent burstiness of resource usage and recent prediction errors to correct the prediction errors. Finally, we also randomly chose a VM that can satisfy the resource demands of the job and allocated the unallocated resource to the job without considering job packing. For DRA, we simulated the purchased capacity as the total amount of resource of all VMs that are used to be allocated to jobs. For each VM, we defined two properties: share and demand. We statically set the share value at the time of VM creation so that the VMs had a mix of high, medium and low shares that correspond to a ratio of 4:2:1, respectively. We used the run-time software to periodically estimate the amount of unused resource of VMs based on the historical resource usage data. Then, we redistributed the purchased capacity among different VMs based on their shares and demands. Finally, we randomly chose a VM that can satisfy the resource demands of the job and allocated the unallocated resource to the job.

We first deployed our testbed on the real cluster using 50 servers and then conducted experiments on the real-world Amazon EC2 using 30 servers. The servers in the real cluster are from HP SL230 servers (E5-2665 CPU, 64GB memory) [34]. The servers in Amazon EC2 are from commercial product HP ProLiant ML110 G5 servers (2660 MIPS CPU, 4GB memory). In both experiments, each server is set to have 1GB/s bandwidth and 720GB disk storage capacity. In both experiments, we used the trace from Google [38] which records the resource requirements and usage of tasks every 5 minutes. Most of the jobs in the Google trace are short jobs [6]. The resource usage of long-lived jobs has some patterns, and by using the original Google trace, the approaches without considering the fluctuations of the amount of unused resource may also handle the prediction of jobs' amount of unused resource. Therefore, we removed the long-lived jobs from the Google trace because it

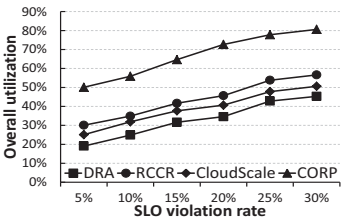


Fig. 8: Resource utilization vs. SLO violation rate on a real cluster.

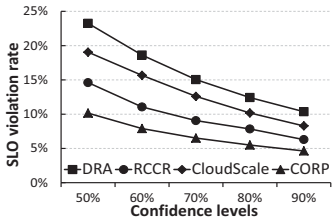


Fig. 9: SLO violation rate vs. confidence levels on a real cluster.

can fully verify if CORP can really overcome the limitations of the other approaches for handling the prediction of the amount of unused resource of short-lived jobs, and it thus makes the evaluation of CORP's performance more convincible (CORP can also achieve good results using the original Google trace because it can handle both long-lived and short-lived jobs with deep learning and HMM model). We transformed the remaining of the 5-minute trace into 10-second trace, and we set the CPU, memory and storage consumption for each job based on the Google trace [38]. In the trace, we considered the tasks of jobs in the trace as short-lived jobs, the bandwidth consumption for each short-lived job is set as 0.02 MB/s [39]. SLO is specified by using a threshold on the response time of a job, and the threshold is set based on the execution time of a task in the trace. To fully verify the performances of our method and the other three methods, we varied the number of jobs from 50 to 300 with step size of 50. Table II shows the parameter settings in our experiment unless otherwise specified.

A. Experimental Results on the Real Cluster

We first calculated the prediction error of CPU by subtracting the predicted amount of unused resource from the actual amount of unused resource for each job. Then we calculated the ratio of the correctly predicted jobs (the jobs whose prediction errors are within $[0,\varepsilon)$) to the number of jobs as the prediction error rate which ranges from 0 to 1. Figure 6 shows the relationship between the prediction error rate and the number of jobs. We see that the prediction error rate follows CORP<RCCR<CloudScale<DRA. The prediction error rate in RCCR is higher than that in CORP because CORP takes advantage of deep learning which can detect complex interactions among features and can learn low-level features from minimally processed raw data. Also, the prediction accuracy of the deep learning algorithm does not rely on the assumption that the historical data for prediction has patterns, which can decrease the prediction error rate generated by the data pattern assumption, and it is suitable for short-lived jobs. Moreover, CORP adequately considers the fluctuations of the unused resource caused by the bursts

of jobs' resource demands and utilizes HMM model to correct the prediction errors, which reduces the prediction error rate. However, RCCR uses a time series forecasting method of which the accuracy relies on the existence of patterns in resource usage [40], [41], to predict the unused resource for long-running service jobs, which can increase the error rate when the resource usage does not have patterns. Also, RCCR does not adequately consider the fluctuations of the unused resource of short-lived jobs caused by their bursts of resource demands, which can increase the error rate, and thus is not suitable for short-lived jobs. CloudScale generates higher prediction error rate than CORP and RCCR because CloudScale's prediction accuracy relies on the assumption of the existence of data patterns in the historical data, which can increase the error rate caused by the data pattern assumption. Although CloudScale uses a multi-step Markov prediction to dealing with the prediction when pattern is not found, it has limited prediction accuracy since the correlation between the resource prediction model and the actual resource demand becomes weaker. Also, CloudScale does not utilize confidence levels to make appropriately-conservative predictions and thus reduce the error rate. The prediction error rate in DRA is higher than all the other methods because DRA does not consider the fluctuations of unused resource, which can increase the prediction error rate. Also, DRA uses the run-time software to estimate the resource periodically, the accuracy of which also relies on the existence of patterns in the training data. In addition, DRA does not utilize confidence levels to make appropriately-conservative predictions and reduce the error rate.

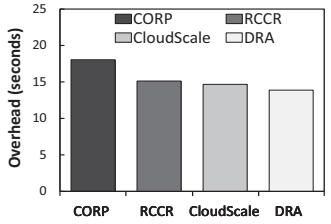
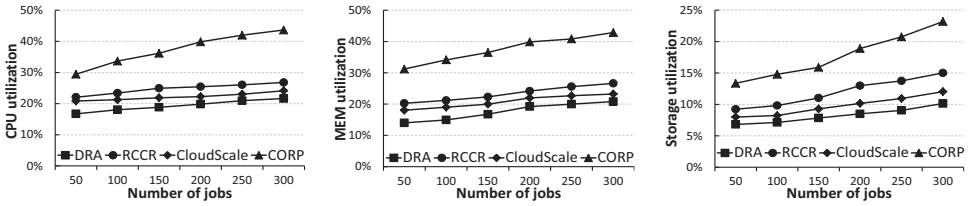


Fig. 10: Overhead of different methods on a real cluster.

We used Equ. (1) to calculate the resource utilization of type j resource. Figure 7 shows the relationship between the resource utilization and the number of jobs. We observe that the resource utilization follows CORP>RCCR>CloudScale>DRA. The resource utilization in CORP is higher than that in RCCR because CORP leverages complementarity of jobs' demands on different resource types and uses a job packing strategy to reduce the resource fragmentation. Also, CORP uses deep learning to predict the unused resource, and adequately considers the fluctuations of short-lived jobs' unused resource, and uses the HMM model to correct the prediction error, and then dynamically allocates the resource to jobs to well meet the requirement of time-varying resource demands and decreases the probability of resource over-provisioning, which is suitable for short-lived jobs. However, RCCR uses a time series forecasting to predict the unused resource for long-term service jobs which is not suitable for short-lived jobs, and the prediction accuracy relies on the existence of patterns in the training data, which can increase the prediction error rate and thus increase the chance of over-provisioning, decreasing the resource utilization. Also, RCCR does not adequately consider the fluctuations of the unused resource in short-lived jobs, which can increase the error rate and thereby increase the probability of over-provisioning. The resource utilizations in CORP and RCCR are higher than that in CloudScale and DRA. This is because CORP and RCCR allocate the resource to jobs in an opportunistic approach in which the allocated unused resource can be reallocated to other new arriving jobs with a certain probability, which can increase the resource utilization. DRA has the lowest resource utilization among all the methods because DRA neglects the fluctuations of the resource which can result in inaccurate prediction of the resource and thus may lead to over-provisioning. Also it is a demand-based resource allocation and does not utilize the allocated but unused resource and reallocate it to other jobs to increase the resource utilization.

We used Equ. (2) to calculate the overall resource utilization (the weighted average of the utilizations of CPU, MEM and storage). Compared to CPU and MEM, storage is not the bottleneck resource, hence we set the weights for CPU, MEM and storage as 0.4, 0.4 and 0.2, respectively. We varied the SLO violation rate by varying the probability threshold P_{th} and thereby varying the percentage of jobs that have SLO violation. Specifically, we considered the SLO violation occurs when a job's response time exceeds the threshold on its response time (We assume jobs' response time is affected by the unavailability of resource for job processing [42], [43]). We recorded the overall resource utilization when the SLO violation rate (approximately) equals 5%, 10%, 15%, 20%, 25% and 30%. Figure 8 shows the relationship between the overall resource utilization and the SLO violation rate. We find the overall resource utilization increases as the SLO violation rate increases. This is because the larger the SLO violation rate, the lower the probability that the resource over-provisioning occurs and thus the higher the overall resource utilization. Also, we see that given an SLO violation rate, the overall resource utilization follows CORP>RCCR>CloudScale>DRA due to the same reasons in Figure 7.

Figure 9 shows the relationship between the SLO violation rate and the confidence level on a real cluster. From Figure 9, we find the SLO violation rate decreases as the confidence level increases. This is because the higher the confidence level, the more conservative the prediction, and the less the amount of resource that will be allocated to jobs in the risk of SLO violations. Also, we find that the SLO violation rate follows CORP<RCCA<CloudScale<DRA. The reason is that CORP utilizes deep learning to accurately predict the amount of unused resource. Also, CORP adequately considers the fluctuations of the amount of unused resource which can result in prediction errors and further lead to SLO violation, and uses HMM model to correct the prediction errors. RCCA uses a time-series based forecasting to predict the unused resource with confidence interval prediction and error correction, which can decrease SLO violation probability. CloudScale uses a prediction error handling to correct prediction errors and perform online adaptive padding to avoid overestimation errors. However, DRA does not have a strategy to handle prediction errors.



(a) CPU (b) Memory (c) Storage

Fig. 11: Utilizations of different resource types vs. number of jobs of different methods on Amazon EC2.

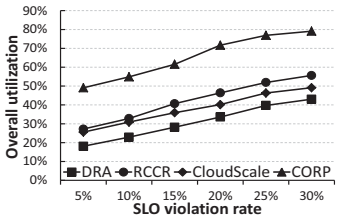


Fig. 12: Resource utilization vs. SLO violation rate on Amazon EC2.

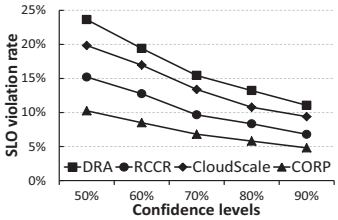


Fig. 13: SLO violation rate vs. confidence levels on Amazon EC2.

We evaluated the overhead of different methods by measuring the latency for allocating resource to 300 jobs in each method. Figure 10 shows the latency of different methods on a real cluster. In Figure 10, we see that the latency of CORP is slightly higher than the other methods. This is because CORP uses the DNN to predict the amount of unused resource of jobs. The DNN has complex structure with multiple layers, which obtains accuracy at the expense of computation overhead and thus increases the latency a little [44]. However, the other methods do not have such complex structure for prediction, thus they have relatively lower latency.

B. Experimental Results on Amazon EC2

To further verify the performance of CORP, we also compared CORP with other methods on Amazon EC2. The servers are from commercial product HP ProLiant ML110 G5 servers (2660 MIPS CPU, 4GB memory). Each server is set to have 1GB/s bandwidth and 720GB disk storage capacity. Figure 11 shows the relationship between the resource utilization and the number of jobs on Amazon EC2. We also see the resource utilization increases as the number of jobs increases, and the resource utilization follows CORP>RCCR>CloudScale>DRA due to the same reasons explained in Figure 7. By examining Figures 11(a)-11(c), we see that the utilizations of CPU and MEM are higher than storage. This is because the storage is not the bottleneck resource and has more wastage in allocation compared to CPU and MEM, thereby has lower resource utilization.

Figure 12 shows the relationship between the overall resource utilization and the SLO violation rate on Amazon EC2. Figure 12 mirrors Figure 8 due to the same reasons. Figure 13 shows the relationship between the SLO violation rate and the confidence level on Amazon EC2. We also find that given a confidence level, the SLO violation rate decreases as the confidence level increases and the SLO violation rate follows CORP<RCCA<CloudScale<DRA due to the same reasons explained in Figure 9.

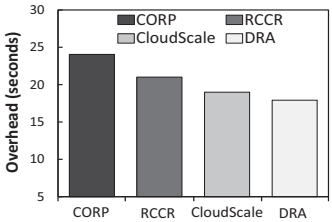


Fig. 14: Overhead of different methods on Amazon EC2.

Figure 14 shows the overhead of different methods measured by the latency for allocating resource to 300 jobs on Amazon EC2. Figure 14 mirrors Figure 10 due to the same reasons. Comparing Figure 14 and Figure 10, we see that the latency in Figure 14 is relatively higher than that in Figure 10. This is because the communication overhead in Amazon EC2 is relatively higher than that in the cluster.

Our experimental results based on the real cluster and Amazon EC2 show that CORP has the best overall performance. This is because CORP explores the potential tradeoff between the efficiency and computation overhead by using deep learning and HMM to get high prediction accuracy of unused resource and utilizing job packing together to obtain high resource utilization.

V. Related Work

To increase resource utilization in a cloud system, some works [4], [7] and product [8] provide methods of reallocating allocated unused resources to new jobs opportunistically. Marshall *et al.* [7] presented reusing unused cloud resources by offering leases in an opportunistic and preemptible way with no SLO guarantees. Amazon EC2 Spot Instances [8] offers opportunistic resources with no SLO guarantees. Users can use spot instances only if their bids exceed the spot price, which is updated every five minutes. Recently, Carvalha *et al.* [4] presented a method to provide a portion of the unused resources with long-term availability SLOs. The method in uses time series forecasting with the assumption that the resource usage patterns exist in training data to predict the unused used resource for long-term service jobs. However, this method is not suitable for processing short-lived jobs because such jobs usually do not exhibit certain resource utilization patterns. Also, it fails to consider fluctuations of unused resource caused by time-varying resource demands of short-lived jobs. In addition, these methods may result in resource fragmentation and lead to low resource utilization because they neglect jobs' resource intensity in multi-resource allocation and may allocate much more resources to the jobs.

Many other works on resource provisioning also have been proposed to improve the resource utilization. The works [26], [36], [45], [46] try to improve the resource utilization by predicting the resource demands and allocating the resources based on the predicted demands. However, the above works do not focus on reallocating the allocated unused resources to increase the resource utilization.

Unlike previous works, CORP first predicts the amount of allocated but unused resource using the deep learning technique, in which the accuracy does not rely on the existence of resource utilization patterns of short-lived jobs. CORP additionally considers the fluctuations of unused resource and uses HMM model to correct prediction errors. Also, CORP packs jobs with complementary resource requirements to VMs to reduce the resource fragmentation and further increase the resource utilization. Thus our proposed method CORP can fully utilize the resource while reducing SLO violation rate.

VI. Conclusions

In this paper, in order to increase the resource utilization and reduce SLO violation rate, we proposed CORP for short-lived jobs, which offers the temporarily-unused resource in an opportunistic manner. CORP is different from previous works in three aspects. First, using the deep learning technique, it can more accurately predict the amount of allocated and unused resources of short-lived jobs, which do not have resource usage patterns. Second, it additionally considers the fluctuations of unused resource caused by time-varying resource demands of jobs to correct the prediction. Third, it leverages complementarity of jobs' requirements on different resource types and packs jobs with complementary requirements on resources to the same VM to further increase the resource utilization. Our extensive experimental results based on a real cluster and Amazon EC2 show our method achieves high resource utilization and provides high SLO guarantee. In the future, we will further consider designing a distributed deep learning training system to reduce the computation overhead caused by DNN, and we will consider both short-lived and long-lived jobs and design an efficient resource allocation strategy which can more increase the resource utilization while reducing the SLO violation rate. Also, we will consider the fluctuation of the workloads, and we will use different real workloads to fully verify the performance of our method.

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] W. Shi, L. Zhang, C. Wu, Z. Li, and F. Lau. An online auction framework for dynamic resource provisioning in cloud computing. In *Proc. of SIGMETRICS*, Austin, June 2014.

[2] S. Dutta and A. Verma. Service deactivation aware placement and defragmentation in enterprise clouds. In *CNSM*, pages 1–9, 2011.

[3] Y. Fu, J. S. Chase, B. N. Chun, S. Schwab, and A. Vahdat. Sharp: an architecture for secure resource peering. In *Proc. of ACM SOSP*, 2003.

[4] M. Carvalho, W. Cirne, F. Brasileiro, and J. Wilkes. Long-term slos for reclaimed cloud computing resources. In *Proc. of SoCC*, Seattle, 2014.

[5] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang. How to bid the cloud. In *Proc. of SIGCOMM*, London, 2015.

[6] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proc. of SoCC*, San Jose, October 2012.

[7] P. Marshall, K. Keahey, and T. Freeman. Improving utilization of infrastructure clouds. In *IEEE/ACM CCGrid*, pages 205–214, 2011.

[8] Amazon EC2 instance purchasing options. https://aws.amazon.com /ec2/purchasing-options [accessed in Mar. 2016].

[9] P. Delgado, F. Dinu, A.-M. Kermarrec, and W. Zwaenepoel. Hawk: Hybrid datacenter scheduling. In *Proc. of ATC*, 2015.

[10] X. Wang, E. Perlman, R. Burns, T. Malik, T. Budavari, C. Meneveau, and A. Szalay. Jaws: Job-aware workload scheduling for the exploration of turbulence simulations. In *Proc. of SC*, 2010.

[11] Y. Chen, S. Alspaugh, and R. Katz. Interactive analytical processing in big data systems: across-industry study of mapreduce workloads. *Proc. of VLDB Endowment*, 5(12):1802–1813, 2012.

[12] J. Liu, H. Shen, and X. Zhang. A survey of mobile crowdsensing techniques: A critical component for the internet of things. In *Proc. of 6th International Workshop on Context-aware Performance Engineering for the Internet of Things (ContextQoS) in conjunction with ICCCN'16, Waikoloa*, 2016.

[13] H. Shen, J. Liu, K. Chen, J. Liu, and S. Moyer. SCPS: A social-aware distributed cyber-physical human-centric search engine. *IEEE Transactions on Computers (TC)*, 64:518–532, 2015.

[14] V. Nae, A. Iosup, S. Podlipnig, R. Prodan, D. Epema, and T. Fahringer. Efficient management of data center resources for massively multiplayer online games. In *Proc. of SC*, 2008.

[15] J. Liu and H. Shen. Dependency-aware and resource-efficient scheduling for heterogeneous jobs in clouds. In *Proc. of CloudCom*, 2016.

[16] A. van den Oord, S. Dieleman, and B. Schrauwen. Deep content-based music recommendation. In *Proc. of NIPS*, pages 2643–2651, 2013.

[17] D. Chicco, P. Sadowski, and P. Baldi. Deep autoencoder neural networks for gene ontology annotation predictions. In *ACM BCB*, 2014.

[18] Z. Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521:452–459, 2015.

[19] H. Shen, Z. Li, J. Liu, and J. E. Grant. Knowledge sharing in the online social network of yahoo! answers and its implications. *IEEE Transactions on Computers (TC)*, 64(6):1715C1728, June 2015.

[20] A. Rai, R. Bhagwan, and S. Guha. Generalized resource allocation for the cloud. In *Proc. of SoCC*, San Jose, October 2012.

[21] H. Shen, A. Sarker, L. Yu, and F. Deng. Probabilistic network-aware task placement for mapreduce scheduling. In *Proc. of IEEE Cluster*, 2016.

[22] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 2015.

[23] J. Liu, H. Shen, and L. Yu. Question quality analysis and prediction in community question answering services with coupled mutual reinforcement. *TSC*, PP(99):1–14, 2015.

[24] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue. Droid-sec: Deep learning in android malware detection. In *SIGCOMM*, pages 371–372, 2014.

[25] J. Zhang, G. Tian, Y. Mu, and W. Fan. Supervised deep learning with auxiliary networks. In *Proc. of KDD*, New York, 2014.

[26] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: Elastic resource scaling for multi-tenant cloud systems. In *Proc. of SoCC*, Oct. 2011.

[27] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng. On optimization methods for deep learning. In *Proc. of ICML*, 2011.

[28] Q. Zhu and T. Tung. A performance interference model for managing consolidated workloads in QoS-aware clouds. In *IEEE CLOUD*, 2012.

[29] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc. of IEEE*, 77(2):257–286, 1989.

[30] M. Stamp. A revealing introduction to hidden markov models. January 18, 2004, http://www.cs.sjsu.edu/faculty/stamp/RUA/HMM.pdf.

[31] W. Gao and G. Cao. Fine-grained mobility characterization: Steady and transient state behaviors. In *Proc.of MOBIHOC*, Chicago, 2010.

[32] J. Liu, L. Yu, H. Shen, Y. He, and J. Hallstrom. Characterizing data deliverability of greedy routing in wireless sensor networks. In *Proc. of SECON*, Seattle, June 2015.

[33] Y. Lv, Y. Duan, W. Kang, Z. Li, and F. Wang. Traffic flow prediction with big data: A deep learning approach. *ITS*, 16(2):865–873, 2015.

[34] Palmetto cluster. http://citi.clemson.edu/palmetto/ [accessed in Mar. 2016].

[35] Amazon EC2. http://aws.amazon.com/ec2 [accessed in Mar. 2016].

[36] G. Shanmuganathan, A. Gulati, and P. Varman. Defragmenting the cloud using demand-based resource allocation. In *SIGMETRICS*, 2013.

[37] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Proc. of CNSM*, 2010.

[38] Google trace. https://code.google.com/p/googleclusterdata/ [accessed in Mar. 2016].

[39] A. L. Shimpi. The SSD anthology: Understanding SSDs and new drives from OCZ. Feb. 2014. http://dx.doi.org/10.1007/s11276-005-6612-9.

[40] C. Chatfield. *The analysis of time series*. Texts in Statistical Science, Chapman & Hall, sixth edition, 2004.

[41] T. Taskaya-Temizel and M.C. Casey. Configuration of neural networks for the analysis of seasonal time series. In *Proc. of ICAPR*, 2005.

[42] U. Sharma, P. Shenoy, and S. Sahu. A flexible elastic control plane for private clouds. In *Proc. of CAC*, Miami, August 2013.

[43] J. Liu and H. Shen. A low-cost multi-failure resilient replication scheme for high data availability in cloud storage. In *Proc. of HiPC*, 2016.

[44] N. D. Lane and P. Georgiev. Can deep learning revolutionize mobile sensing? In *Proc. of HotMobile*, Santa Fe, 2015.

[45] C. Curinom, D. E. Difallahu, C. Douglasm, S. Krishnanm, R. Ramakrishnanm, and S. Raom. Reservation-based scheduling: If you're late don't blame us! In *Proc. of SoCC*, Seattle, Nov. 2014.

[46] C. Delimitrou and C. Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proc. of ASPLOS*, pages 127–143, 2014.