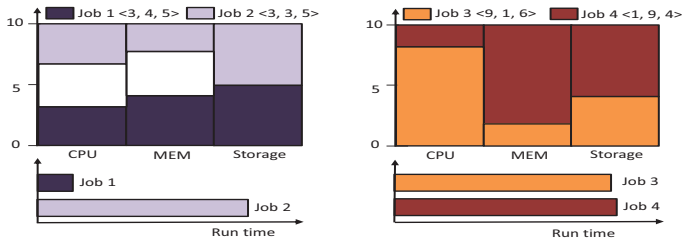
CCRP: Customized Cooperative Resource Provisioning for High Resource Utilization in Clouds

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

Abstract*—In cloud systems, efficient resource provisioning is needed to maximize the resource utilization while reducing the Service Level Objective (SLO) violation rate, which is important to cloud providers for high profit. Several methods have been proposed to provide efficient provisioning. However, the previous methods do not consider leveraging the complementary of jobs' requirements on different resource types and job size concurrently to increase the resource utilization. Also, by simply packing complementary jobs without considering job size in the job packing, it can decrease the resource utilization. Therefore, in this paper, we consider both jobs' demands on different resource types (in the spatial space) and jobs' execution time (in the temporal space); we pack the complementary jobs (whose demands on multiple resource types are complementary to each other) belonging to the same type and assign them to a Virtual Machine (VM) to increase the resource utilization. Moreover, the previous methods do not provide efficient resource allocation for heterogeneous jobs in current cloud systems and do not offer different SLO degrees for different job types to achieve higher resource utilization and lower SLO violation rate. Therefore, we propose a Customized Cooperative Resource Provisioning (CCRP) scheme for the heterogeneous jobs in clouds. CCRP uses the hybrid resource allocation and provides SLO availability customization for different job types. To test the performance of CCRP, we compared CCRP with existing methods under various scenarios. Extensive experimental results based on a real cluster and Amazon EC2 show that CCRP achieves 50% higher or more resource utilization and 50% lower or less SLO violation rate compared to the previous resource provisioning strategies.

I. Introduction

Infrastructure-as-a-Service (IaaS) clouds have emerged as appealing computing infrastructures, which allow customers to rent resources such as CPU, memory (MEM in short), storage, etc. in terms of Virtual Machines (VM) from cloud providers. Customers benefit IaaS from being able to adjust capacity to demand without practical limits (resource capacity) while shifting fixed infrastructure cost to providers. In practical scenarios, however, elasticity does not come true [1]. Most IaaS cloud providers (e.g., Amazon EC2) offer different VM types (e.g., small, medium, large and extra large) with a fixed amount of CPU cores, MEM, and disk. Moreover, in cloud-scale clusters, an increasing number of jobs are submitted for processing. The jobs can vary significantly in size and have diverse requirements regarding the job completion time, latency [2–5]. Short jobs (e.g., queries) are latency sensitive while long jobs (e.g., graph analytics) can tolerate long latencies. Therefore, efficiently processing heterogeneous workloads in data centers and determining the most appropriate resource provisioning strategy for the jobs with different sizes are challenging and become important problems [6–8].



(a) W/o avoiding fragmentation or considering job size.

(b) W/ avoiding fragmentation and considering job size.

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

Continuous resources (e.g., computing resource) are required to process jobs, and different jobs have different resource demands. For example, some jobs such as a scientific computing type are CPU intensive, the other jobs such as games are MEM intensive, and even some jobs like I/O bound jobs are storage intensive [9–11]. Therefore, allocating the resource to multiple jobs with complementary resource requirements can decrease resource fragmentation and thereby increasing resource utilization.

To reduce resource waste and improve the resource utilization, some works [12, 9] use packing strategies to pack the jobs (tasks) whose demands on multiple resource types are complementary to each other, and then allocate resources to the jobs (tasks). The work [12] presents Tetris, a multi-resource scheduler that packs tasks to machines based on their requirements of all resource types. Tetris [12] packs tasks to machines based on the alignment score (the weighted dot product value between the vector of a machine's available resources and the task's peak usage of resources). The task with the highest alignment score is scheduled and allocated its peak resource demands. The work [9] packs jobs to servers to improve resource utilization and throughput. However, the *limitation* of these works is that they neglect the diversities on the size (execution time) of jobs (tasks) when they pack jobs (tasks) together. Nevertheless, our workload analysis according to Facebook 2009 (FB09), Facebook 2010 (FB10), Google cluster trace data, Yahoo cluster trace, and Cloudera Hadoop workload (Cloudera-b) is shown in Figs. 2 and 3. Figs. 2 and 3 show the distribution of jobs and resource usage based on job size in various commercial systems. The sizes of jobs vary significantly as shown in Fig. 2, and long jobs consume a large portion of resource as presented in Fig. 3. Therefore, the diversities on job size can still result in resource wastage when the sizes of the packed jobs are different from each other.

TABLE I: Comparison of resource allocation strategies with respect to: utilization, cost, flexibility, SLO violation rate and performance unpredictability.

Strategy	Utilization	Cost	Flexibility	SLO violation rate	Perf. unpredictability
Demand-based	Low	High	Yes	Low	Yes
Opportunistic	High	Low	Yes	High	Yes
Hybrid	High	Median	Yes	Median	Yes

Fig. 2: Job distribution based on job size of various commercial workloads.

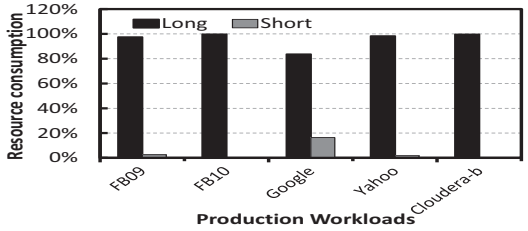


Fig. 3: The distribution of jobs' resource consumption based on job size of various commercial workloads.

To increase the resource utilization, the work [13] proposes Wrangler, a system that proactively avoids situations causing stragglers. Wrangler automatically learns to predict such a situations using a statistical learning technique based on cluster resource utilization counters. The work [14] presents Hopper to dynamically allocate the resource (slots) to jobs by considering the speculation requirements for predictable performance. The work [15] presents High Utilization with Guarantees (HUG) to achieve maximum attainable network utilization without sacrificing the optimal isolation guarantee, strategy-proofness. The work [16] presents CloudCache, an on-demand cache management solution to meet VM cache demands and minimize cache wear-out. To support on-demand cache allocation, the paper proposes a new cache demand model, Reuse Working Set (RWS), to capture only the data with good temporal locality. However, these methods do not consider the job (or workload) size for job packing and resource reallocation to improve the resource utilization, and thus cannot fully utilize the resource.

To increase the resource utilization, we propose a customized cooperative resource provisioning (CCRP) scheme to process the heterogeneous jobs existing in current cloud systems. CCRP uses the hybrid resource allocation strategy for heterogeneous jobs. CCRP customizes SLO availability by offering different degrees of SLO availability for different job types so that CCRP can achieve higher resource utilization and lower SLO violation rate. Table I shows the comparison of resource allocation strategies with respect to utilization, cost, flexibility, and performance unpredictability. The hybrid resource allocation strategy has the potential to offer the best of both worlds by allowing users to leverage on-demand resource for the long term jobs and opportunistic resource for short term jobs [6]. The key *contributions* of this paper are summarized below.

- CCRP presents a customized cooperative resource provisioning scheme for high resource utilization and low SLO violation rate. Specifically, CCRP uses the hybrid resource allocation strategy for heterogeneous jobs by applying the opportunistic-based resource allocation for short jobs with lower SLO availability guarantee and the demand-based resource allocation for long jobs with higher SLO availability guarantee.
- CCRP accurately predicts the execution time of jobs and classifies jobs into two types: short jobs and long jobs for job packing to fully utilize the resource. Also, CCRP combines the deep learning algorithm and the Semi-Markov Model (SMM) model with considering the fluctuations of the amount of unused resource to accurately predict the amount of unused resource, and it increases the resource utilization and reduces the SLO violation rate by avoiding resource over-provisioning and under-provisioning.
- CCRP leverages the complementary of jobs' requirements on different resource types and jobs' heterogeneity in job size, and it packs complementary jobs whose demands on multiple resource types are complementary to each other and allocates them to a VM to further increase the resource utilization.
- Extensive experiments have been conducted based on a real cluster and Amazon EC2. The results show that CCRP achieves 50% higher or more resource utilization and 50% lower or less SLO violation rate compared to previous resource provisioning strategies.

The rest of this paper is organized as follows: Section II describes the system model in CCRP. Section III presents the system design of CCRP. Section IV presents the performance evaluation of CCRP. Section V reviews the related work. Section VI concludes this paper.

II. System Model

In this section, we first introduce some concepts and assumptions, and then we formulate our problem based on the concepts and assumptions. The physical machines (PMs) are deployed in a cloud system, and their resources are allocated to VMs (or containers). The VM capacity comprises multiple resource types (e.g., CPU, MEM, and storage). In the cloud system, we assume that each user can get a set of VMs [17, 15]. A user can execute as many jobs as she can fit in each VM she gets. For long jobs, CCRP uses a demand-based resource allocation to ensure higher availability SLO, and the VMs allocated to users' jobs cannot be reallocated to other jobs until users' jobs finish execution [18, 1]. For short jobs, CCRP uses opportunistic-based resource allocation, and the VMs allocated to users' jobs can be reallocated to other jobs so that the resource utilization can be increased. In this paper, we consider the problem of allocating multiple resource types to users' jobs for achieving high resource utilization and low SLO violation rate. To reduce resource fragmentation and increase resource utilization, jobs with different resource intensities (e.g., a job with high demand on CPU and a job with a high demand on MEM) can be allocated resources in a VM together.

A. The Overall Utilization

Denoted N_p as the number of PMs (servers), N_v as the number of VMs, and N_u as the number of users in the cloud system. Let n_t be the number of jobs $\mathbf{J} = \{J_1, ..., J_n\}$ submitted at time slot t in the cloud system. Denoted $s_1, ..., s_{n_t}$ as sizes of jobs $J_1, ..., J_{n_t}$. Suppose there are l resource types (e.g., CPU, MEM, storage). Different jobs have different resource demands on different resource types. Let d_{ij} be job J_i's demand on resource type j ($j \in \{1, ..., l\}$), r_{ij} be type j resource allocated to job J_i, and r_{ij}^u be the amount of unused type j resource allocated to job J_i. Denoted $U_{j,t}$ as the type j resource utilization at time slot t, and $U_{a,t}$ as the overall resource utilization at time slot t. Then, the type j resource utilization at time slot t can be expressed as follows:

$$U_{j,t} = \frac{\sum_{i=1}^{n_t}(r_{ij} - r_{ij}^u)}{\sum_{i=1}^{n_t} r_{ij}}. \quad (1)$$

The overall resource utilization at time slot t can be obtained as

$$U_{a,t} = \frac{\sum_{j=1}^{l}(w_j \sum_{i=1}^{n_t}(r_{ij} - r_{ij}^u))}{\sum_{j=1}^{l}(w_j \sum_{i=1}^{n_t} r_{ij})}, \quad (2)$$

where w_j is the weight for type j resource and $\sum_j^l w_j = 1$. The reason for setting different weights for distinct 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 [1]. Our goal is to maximize the overall resource utilization, which will be shown in the formulated problem in the following subsection.

B. The Optimization of the Overall Utilization

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

Let $x_{ik} = \{0, 1\}$ be a binary variable representing if job J_i is assigned to VM v_k. To maximize the resource utilization, we present an integer linear programming (ILP) model and formulate the problem as a linear optimization problem below:

$$Max\{U_{a,t}\}, \quad (3)$$

$$s.t.\ r_{ij} \geq d_{ij}\ (\forall i \in \{1, ..., n_t\}, j \in \{1, ..., l\}), \quad (4)$$

$$\sum_{i=1}^{n_t} x_{ik} \cdot r_{ij} \leq R_{kj}\ (\forall i \in \{1, ..., n_t\}, j \in \{1, ..., l\}, k \in \{1, ..., N_v\}), \quad (5)$$

$$x_{ik} \in \{0, 1\}\ (\forall i \in \{1, ..., n_t\}, k \in \{1, ..., N_v\}), \quad (6)$$

$$d_{ij} \geq 0\ (\forall i \in \{1, ..., n_t\}, j \in \{1, ..., l\}), \quad (7)$$

$$r_{ij}^u \geq 0\ (\forall i \in \{1, ..., n_t\}, j \in \{1, ..., l\}), \quad (8)$$

where R_{kj} is the remaining type j resource of VM v_k. Constraint (4) is to ensure that the allocated resource meets each job's demand on each resource type. Constraint (5) is to ensure that the amount of the allocated resource from each VM does not exceed its capacity of each resource type. Constraint (7) is to ensure that job J_i's demand on type j resource is non-negative. Constraint (8) is to ensure that job J_i's unused type j resource at time slot t is non-negative.

The linear optimization problem can be solved by using the CPLEX linear program solver [19]. The ILP optimization problem is an NP-hard problem and has a high computational complexity [20]. Therefore, we propose a heuristic method called CCRP. CCRP first accurately estimates the run time of jobs [4], and it then classifies the jobs into long and short jobs based on the estimated run time of jobs. Next, CCRP leverages the complementarity of jobs' requirements on different resource types by using a packing strategy to pack the complementary jobs together and then allocates resource to the packed jobs. For long jobs, CCRP uses the demand-based resource allocation to obtain higher SLO availability, and for short jobs, CCRP uses the opportunistic-based resource allocation.

III. System Design

In this section, we introduce the design of our proposed method CCRP. CCRP uses opportunistic-based resource allocation strategy for short jobs with lower SLO availability guarantee to achieve higher resource utilization, and uses demand-based resource allocation for long jobs with higher SLO availability guarantee to achieve higher resource utilization and lower SLO violation simultaneously. The reason behind this approach is that short jobs usually can tolerate some SLO unavailability [1], and long jobs do not.

Fig. 4 shows the architecture of the hybrid resource allocation scheme of CCRP. In the left part, CCRP first predicts if the job is short job based on the selected features shown in Table II. The right part of Fig. 4 shows that CCRP first packs the same type jobs. Then, CCRP uses demand-based resource allocation for long jobs and opportunistic-based resource allocation for short jobs. To have such a hybrid scheme, jobs need to be classified. Below, we describe the job classification based on jobs' execution time.

A. Job Classification

The reasons that we prefer to use a classification model are as follows: (i) For job classification, we care about the approximate execution time rather than the exact execution time. From the machine learning point of view, a classification model usually performs better than a two-step approach (i.e., step 1: use the regression model to estimate the execution time by minimizing the mean square error; step 2: compare the estimated value with a threshold of execution time) [21]. (ii) Regression models only give the single point estimation, and the single point estimation does not include enough information for CCRP to make profit-aware decisions [21]. However, the classification model gives the probabilities that a job falls in the category of short job or long job.

We use a non-linear classification model to predict jobs' execution time. The reason is that the execution of a job may depend on many factors in a non-linear fashion [21]. We choose Support Vector Machine (SVM) as the classification model. SVM [22] constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which is used for classification, regression, or other tasks. Support Vector Machines are widely used classifier because of its robustness in the presence of noise, and high reported accuracy. We use the WEKA implementation of SVM of SMO [23, 24].

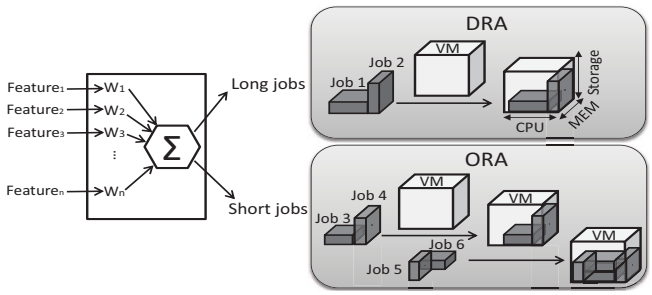


Fig. 4: Architecture of the hybrid resource allocation scheme CCRP: DRA (demand-based resource allocation for long jobs) and ORA (opportunistic-based resource allocation for short jobs)

1) Predicting Execution Time of Jobs: To more accurately predict the execution time of jobs, we extract two types of features: job-related features and system-related features [21, 13, 25]. We use the historical data from the Google trace [26] to estimate the run time of jobs. We extract the numerical values of the features from the historical data for predicting the jobs' execution time (see Table II). In the historical data, we consider part of them as training data, and use part of the data as testing data. To improve the accuracy, we use the cross-validation to perform classification.

We classify jobs into two types: short jobs and long jobs. We consider jobs with execution time no more than 10 minutes as short jobs [2], and we consider jobs with execution time more than 10 minutes as long jobs. To achieve high resource utilization, CCRP packs the complementary jobs belonging to the same type together and allocates the resource to the packed job. Below, we introduce the resource allocation algorithm.

B. Resource Allocation Algorithm

After classifying jobs based on jobs' execution time, CCRP uses opportunistic-based resource allocation for short jobs, and uses demand-based resource allocation for long jobs. In opportunistic-based resource allocation, the resource allocated to short jobs can be reallocated to other short jobs with a certain probability. Below, we introduce the opportunistic-based resource allocation for short jobs and demand-based resource allocation for long jobs, respectively.

1) Opportunistic-based Resource Allocation: For short jobs, the allocated resource (e.g., VM) can be reallocated to other jobs [1]. To determine the amount of unused resource, CCRP uses the deep neural network (DNN) to predict the amount of unused resource of VMs. This is because the resource usage of short jobs usually does not have certain patterns [27], and the prediction accuracy of DNN does not rely on the existence of resource utilization patterns [28]. However, the prediction accuracy of other methods such as time series forecasting and fast Fourier transform [29, 1, 30, 31], rely on the existence of resource utilization patterns of short jobs. Also, DNN is more accurate as its multiple hidden layers are capable of modeling complex data with great efficiency and it has an advantage over shallow machine learning methods [32, 33]. Below, we present the details of the method for predicting the amount of unused resource of VMs.

a) Predicting Unused Resource Using DNN: We use CPU as an example to illustrate the prediction of the amount of unused resource. Each input data contains CPU utilization 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. In the feed-forward evaluation, the neurons take input data and perform simple operations on the data, and pass the results on to the up-layer neurons. In the back-propagation, the neurons calculate errors at output units and the errors are then back-propagated for each neuron in the lower layer of the output layer. In the weight update step, the errors are used to update the weights. Below, we introduce the details of each step.

TABLE II: Features for predicting jobs' execution time.

Feature	Description
Job-related features	
Required CPU	Amount of CPU resource required by the job
Required MEM	Amount of MEM resource required by the job
Required storage	Amount of storage resource required by the job
# of tasks	Number of tasks which the job contains
Priority of job	The priority of the job
System-related features	
CPU utilization	VM's CPU utilization
MEM utilization	VM's MEM utilization
Storage utilization	VM's storage utilization

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

$$\phi_i(d) = F((\Sigma_{j=1}^{c}\omega_{ij}(d-1,d)\cdot\phi_j(d-1)) + b_i) \quad (9)$$

where b_i is a bias term for the neuron. Equ. (9) is a sigmoid function, which is a nonlinear function associated with all neurons in the network, and is more accurate [34].

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

$$E_i(d_h) = (t_i(d_h) - \phi_i(d_h))\cdot F'(\phi_i(d_h)) \quad (10)$$

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

$$E_i(d) = (\sum_{j=1}^{v} E_j(d+1)\cdot\omega_{ji}(d,d+1))\cdot F'(\phi_i(d)) \quad (11)$$

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

$$\Delta\omega_{ij}(d-1,d) = \beta\cdot E_i(d)\cdot\phi_j(d-1),\ \forall j = 1,...,c \quad (12)$$

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

The process of these three steps repeats 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 on a held-out validation set. Normally, the training continues for multiple epochs, reprocessing the training data set each time, until the validation set error converges to a low value. The deep learning algorithm for predicting the amount of unused resource comprises two parts: training and testing. In 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. In testing, the algorithm autoencodes the input and generates the output.

b) Predicting Fluctuations of Unused Resouce using SMM: Short jobs usually exhibit fluctuations in resource usage [35], and the amount of unused resource of VMs running short jobs may fluctuate from time to time. Thus, we use a SMM to predict the fluctuations of the amount of unused resource of VMs.

Consider a continuous time chain $\{\mathcal{C}(t)\}_{t\in[0,\infty)}$. Denote T_n $(n \geq 1)$ as time of the occurrence of the nth transition occurs after $t = 0$. Define the duration between two consecutive transitional epochs (say T_{n-1}, T_n) as the time chain's nth holding time h_n. In the semi-Markov process, the chain's development after each transition is independent of the chain's behavior before that time, that is, the distribution of the chain's the holding time $h_n = T_n - T_{n-1}$ is independent of the chain's behavior before T_{n-1} but may be a function of $\mathcal{C}_{n-1} = \mathcal{C}(T_{n-1})$ and $\mathcal{C}_n = \mathcal{C}(T_n)$. For convenience, we denote $h_n = T_n - T_{n-1}$ by h_{ij} if $\mathcal{C}_{n-1} = i$ and $\mathcal{C}_n = j$.

Denote $S = \{S_1, ..., S_M\}$ $(M = 3)$ as the set of states in SMM (S_1, S_2, S_3 represent "peak", "center", "valley", respectively). Let A be the transition probability matrix, and we have

$$A = \{a_{ij}\}\ (a_{ij} \geq 0, \sum_{j=1}^{M} a_{ij} = 1,\ i, j \in \{1, ..., M\}) \tag{13}$$

where M is the total number of states in the model [36].

Before state S_i transits to state S_j, the process remains in S_i for a time h_{ij} (i.e., holding time). In SMM, the holding times are positive, integer-valued, random variables. All holding times are finite, and each holding time is no less than 1 time unite. The probability mass function H_{ij} in h_{ij} is called holding time mass function for the transition from state S_i to state S_j. Thus we have,

$$H_{ij}(m) = Pr\{h_{ij} = m\},\ m \in \{1, ..., n\} \tag{14}$$

where n is the number of time intervals. To describe a discrete-time semi-Markov process completely, the holding time mass functions and transition probabilities need to be specified. Denote $B(m)$ as the core matrix, and $B_{ij}(m)$ is the probability of the joint event that the system entered state S_i at time 0 and transits to S_j after a holding time m. Thus we have,

$$\begin{aligned} B_{ij}(m) &= Pr((\mathcal{C}_n = j | \mathcal{C}_{n-1} = i), (h_{ij} = m)) \\ &= Pr(\mathcal{C}_n = j | \mathcal{C}_{n-1} = i) \cdot Pr(h_{ij} = m) \\ &= A_{ij} \cdot H_{ij}(m),\ i, j \in \{1, ..., M\}, m \in \{1, ..., n\} \end{aligned} \tag{15}$$

For simplicity, we denote Equ. (15) in congruent matrix multiplication form by

$$B(m) = A \otimes H(m) \tag{16}$$

where the operator $\otimes$ means multiplication of corresponding elements. The waiting time mass function $w_i(m)$ for the ith state (i.e., The probability that the waiting time for the ith state equals m) is the summation of the elements of $B(m)$ across the ith row

$$w_i(m) = \sum_{j=1}^{M} B_{ij}(m) = \sum_{j=1}^{M} A_{ij} H_{ij}(m) \tag{17}$$

Thus, the probability that the waiting time for the ith state is less than or equal to n, denoted by $cw_i(n)$, is

$$cw_i(n) = \sum_{m=1}^{n} w_i(m) \tag{18}$$

Hence the cumulative probability distribution of the waiting time can be obtained from Equ. (18). We can get the probability of the waiting time for the ith state being greater than n (denoted by $\overline{cw}_i(n)$) by complementing $cw_i(n)$.

$$\overline{cw}_i(n) = \sum_{m=n+1}^{\infty} w_i(m) \tag{19}$$

To know the transition in an interval, we need to compute the interval transition probability matrix, in which the probability of a transition from state S_i to state S_j in the interval $(0, n)$ requires that the process transits at least once during that interval. The process may transit from S_i to some other state at time m $(0 \leq m \leq n)$ first, and then makes its way to S_j at time n by the sum of a succession of transitions. Hence,

$$\begin{aligned} \mathcal{F}(n) &= \overline{CW}(n) + \sum_{m=0}^{n} A \otimes H(m) \mathcal{F}(n-m) \\ &= \overline{CW}(n) + \sum_{m=0}^{n} B(m) \mathcal{F}(n-m), n \in \{0, 1, 2...\} \end{aligned} \tag{20}$$

where $\overline{CW}(n)$ is a diagonal matrix with its ith element equaling $\overline{cw}_i(n)$. The interval transition probability $\mathcal{F}(n)$ can be obtained using recursive process, and $\mathcal{F}(n)$ is obtained for the interval $1 \leq m \leq n$ because $H(0) = 0$. In the case $n = 0$, $\mathcal{F}(n)$ equals the following Kronecker Delta or identity matrix.

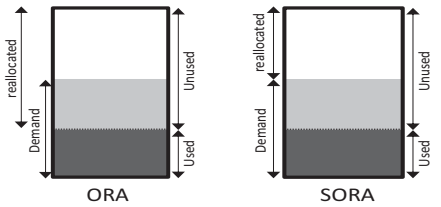
$$\mathcal{F}_{ij}(0) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \tag{21}$$

Denote $\hat{\xi}$ as the predicted amount of unused resource CPU using the deep learning algorithm. Let $\hat{u}_{t+L}$ represent the predicted amount of unused resource CPU corrected by the SMM model at time t for a future time $t + L$. If the predicted state of the amount of unused resource falls in the peak, CCRP increases the amount by $\hat{u}_{t+L} = \hat{\xi} + min(h_{cpu} - m_{cpu}, m_{cpu} - l_{cpu})$, 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 state falls in the valley, CCRP makes the adjustment by $\hat{u}_{t+L} = \hat{\xi} - min(h_{cpu} - m_{cpu}, m_{cpu} - l_{cpu})$.

c) Prediction with Confidence Intervals: To improve the prediction accuracy, we generate 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 calculation of confidence interval depends on the variance of the prediction errors and the confidence level η. Let $\alpha = 1 - \eta$ be the significance level. Hence, the confidence interval is

$$\hat{u}_{t+L} \pm \hat{\sigma} \cdot z_{\frac{\alpha}{2}} \tag{22}$$

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



(a) ORA (b) SORA

Fig. 5: Comparison of opportunistic-based resource allocation and semi-opportunistic-based resource allocation.

Based on the confidence interval, we adjust the predicted amount of unused resource for time $t+L$ as follows

$$\hat{u}_{t+L} = \hat{u}_{t+L} - \hat{\sigma} \cdot z_{\frac{\alpha}{2}} \tag{23}$$

We use the lower bound of the confidence interval in Equ. (23) because the underestimation of the amount of unused resource makes it conservative in reallocating allocated resources, thus lowering the risk of SLO violations.

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

$$\delta_{t+\tau} = u_{t+\tau} - \hat{u}_{t+L}, \forall \tau \in [1, L] \tag{24}$$

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

Let ϵ be pre-specified prediction error tolerance and Pr^{th} be a pre-defined probability threshold. For the predicted unused resource with prediction error δ_{t+L}, if δ_{t+L} satisfies the following inequality [37]

$$Pr(0 \leq \delta_{t+L} < \epsilon) \geq Pr^{th} \tag{25}$$

then it can be allocated to a new arriving job.

2) Semi-opportunistic-based Resource Allocation: To improve the SLO availability for short jobs, CCRP introduces a concept called semi-opportunistic-based resource allocation (SORA), that is, the cloud provider reallocates only the amount of resources beyond the job's demand to other jobs. Current public cloud providers offer resources to users' jobs in the form of pre-defined VMs [38]. As users usually do not know how much resource their jobs need, and they are usually allocated resources up to their ceiling [1, 38]. The actual amount of unused resource of jobs may be more than the amount of resource allocated to users' jobs beyond jobs' demands. Denote r_{ij} as the allocated amount of type j resource to job i. Then the amount of type j resource beyond job j's demand is $r_{ij} - d_{ij}$. The amount of type j resource that CCRP will allocate to other jobs is $r_{ij} - d_{ij}$. In this case, it is more conservative in reallocating the allocated resource as it can better ensure that the reallocation of the resource will not affect job j's execution and the availability of the unused resource, and thus it reduces the chance of the occurrence of SLO violation due to the unavailability of the resource [1, 39]. Also, the SORA improves the resource utilization. Fig. 5 shows the comparison of ORA and SORA. Fig. 5(a) shows that ORA reallocates the actual amount of unused resource of the job to other jobs, and Fig. 5(b) shows that SORA reallocates the amount of resource beyond the job's demand to other jobs. From Fig. 5, we see that SORA reallocates part of the actual unused resource to other jobs, which has more chance of ensuring higher SLO availability. In practical scenarios, the selection of ORA and SORA can be based on users' SLO requirements.

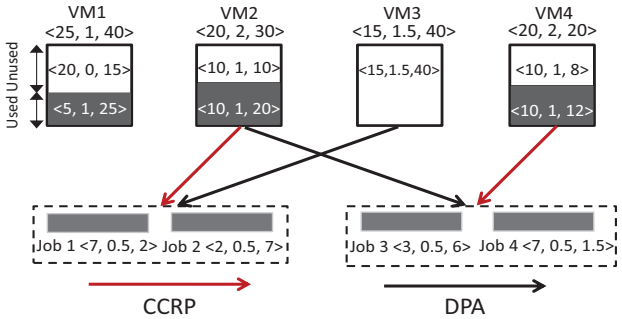


Fig. 6: Comparison of packing-based resource allocation. CCRP: allocate resource to jobs based on unused resource volume and demand volume of jobs; DPA: allocate resource to jobs based on the dot product of the unused resource of VMs and jobs' resource demands.

3) Demand-based Resource Allocation: For long jobs, C-CRP uses demand-based resource allocation to provide high SLO availability. In demand-based resource allocation, the resources (e.g., VMs) allocated to jobs cannot be reallocated to other jobs. For demand-based resource allocation, CCRP first classifies jobs, and then it packs the complementary jobs with the same job type whose demands on multiple resource types are complementary to each other together, and then finds the VM that can best fit the resource demands of the packed jobs.

After classifying jobs, CCRP packs the complementary jobs together to increase the resource utilization. Each job has a dominant resource, defined as the one that requires the most amount of resource. CCRP 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. 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, CCRP calculates its deviation with every other job J_j if J_j has different dominate 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. We use an example shown in Fig. 6 to illustrate the process of job packing in CCRP. Consider jobs 1-4 with resource requirements $< 7, 0.5, 2 >$, $< 2, 0.5, 7 >$, $< 3, 0.5, 6 >$ and $< 7, 0.5, 1.5 >$, respectively. The dominant resource of job 1 and job 4 is CPU, and the dominant resource of job 2 and job 3 is storage. The resource demand deviation of job 1 and job 2 is 25, and the resource demand deviation of job 1 and job 3 is 16. Since 25>16, job 1 and job 2 are packed together. Similarly, job 3 and job 4 are packed together.

After packing jobs, CCRP needs to assign jobs to VMs. To increase the resource utilization, CCRP assigns the job entity (packed jobs or job) to a VM that on the one hand can satisfy the job entity's resource demand and on the other hand can best fit its requirements. Below we introduce the method that CCRP finds the VM for a job entity. To more fully utilize the resource, CCRP chooses the VM that has the

least remaining unused resource (referred to as best fit VM). To find the best fit VM, we introduce two concepts called unused resource volume and demand volume, respectively. Denote $C^* =< C_1^*, ..., C_l^* >$ as the vector of the maximum capacity of each resource type among all VMs. Let $\hat{\mathbf{R}}_j = (\hat{R}_{j1}, ..., \hat{R}_{jl})$ be the amount of predicted unused resource of VM j. The unused resource volume of VM j is obtained as follows

$$VOL_j = \sum_{i=1}^{l} \frac{\hat{R}_{ji}}{C_i^*} \tag{26}$$

Suppose the resource demands of job entity i (job 1, job 2) is $(r_{i1}, ..., r_{il})$. Then the demand volume of job entity i is $vol_i = \sum_{k=1}^{l} \frac{r_{ik}}{C_k^*}$. CCRP utilizes the logical operation "$\wedge$" to check if a VM (say VM j) satisfies the resource demands of a job entity (say job entity i). Specifically, it uses a l-bit number to represent the result of the comparison for each resource type, i.e, $\hat{R}_{jk} \geq r_{ik}$ $(k \in \{1, ..., l\})$, where "1" means true. If $\hat{R}_{j1} \geq r_{i1} \wedge ... \wedge \hat{R}_{jl} \geq r_{il} = 1$, then VM j satisfies the resource demands of job entity i. Then CCRP chooses the VM that satisfies the resource demand of job entity i and has the smallest unused resource volume as the best fit VM.

Fig. 6 illastrutes an example which compares the packing-based resource allocation in CCRP with dot product-based approach (DPA) [12]. In Fig. 6, for VMs, the numerical values from top to bottom indicate VMs' capacities, unused resource, and used resource of different resource types (VM3's used resource is not shown in Fig. 6 because VM3 is an idle VM). The dominant resource of job 1 and job 4 is CPU, and the dominant resource of job 2 and job 3 is storage. The maximum capacities of CPU, MEM and storage among all VMs are $C^* =< 25, 2, 40 >$. According to Formula (26), the unused resource volumes of VMs 1-4 are 1.175, 1.15, 2.35 and 1.1, respectively. To allocate resource to job entity (job 1, job 2), CCRP finds only VM2 and VM3 can satisfy the resource demands of job entity (job 1, job 2). Since the $VOL_2 = 1.15 < VOL_3 = 2.35$, job entity (job 1, job 2) will be allocated to VM2. Similarly, only VMs 2-4 can satisfy the resource demands on each resource type of job entity (job 3, job 4). Since $VOL_4 = 1.1 < VOL_2 = 1.15 < VOL_3 = 2.35$, job entity (job 3, job 4) will be allocated to VM4. In DPA, to allocate resource to job entity (job 1, job 2), DPA calculates the dot product of the job entity's resource demands and the unused resource of VM2 and VM3, and they are 181 and 496.5, respectively. Since 496.5>181, DPA chooses VM3 as the best fit VM for job entity (job 1, job 2). Similarly, $451.5 > 176 > 161$ (VM3>VM2>VM4 for unused resources), but VM3 has been allocated to job entity (job 1, job 2), DPA chooses VM2 as the best fit VM for job entity (job 3, job 4), and allocates job entity (job 3, job 4) to VM2. Therefore, in this scenario, CCRP can reduce more resource fragmentation than DPA, and CCRP thus has higher utilization than DPA.

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 [41], and Amazon EC2 [42], respectively. To show CCRP's performance, we compared CCRP with R-CCR [1], CloudScale [29], D-DA [38] and Tetris [12] in various scenarios since all these methods share the same objective of maximizing the resource utilization while avoiding SLO violation.

TABLE III: Parameter settings.

Parameter	Meaning	Setting	Parameter	Meaning	Setting
N_p	# of servers	30-50	h	# of layers in DNN	4 [40]
N_v	# of VMs	100-400	N_n	# of units per layer	50
$\|\mathbf{J}\|$	# of jobs	300-1500	M	# of states in SMM	3
l	# of resc. types	3	α	Significance level	5%-30%
Pr^{th}	Prob. threshold	0.95	η	Confidence level	50%-90%

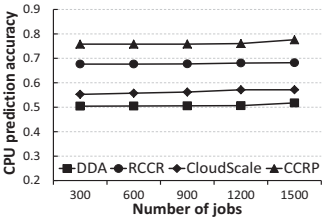
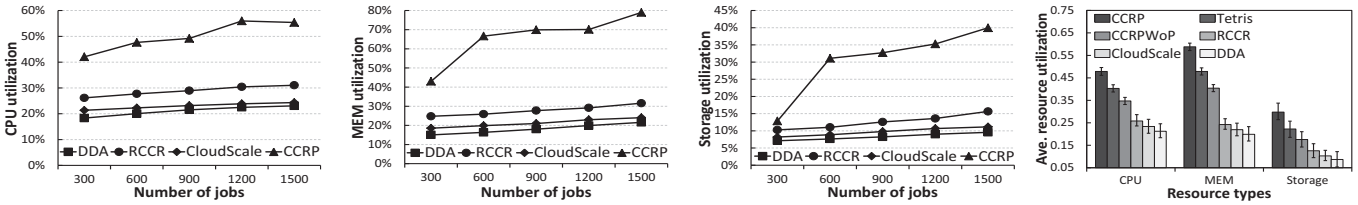


Fig. 7: Prediction accuracy of different methods on a real cluster.

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. DDA purchases capacity for the customers, and then re-distributes the purchased capacity among customer's VMs based on their demand. Specifically, DDA considers the share value and the demand value of customer's VMs and allocates the aggregate amount of capacity among the VMs. Tetris adopts DPA approach to pack tasks to machines. Specifically, when resources on a machine become available, Tetris first selects the set of tasks (jobs) whose peak usage of each resource can be accommodated on that machine. Then, for each task (job) in this set, Tetris computes an alignment score to the machine. The task with the highest alignment score is scheduled and allocated its peak resource demands. However, Tetris does not have the prediction method. To have fair comparison with Tetris, we have integrated our prediction method to Tetris and compare the average utilization performance.

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 MEM) [41]. The servers in Amazon EC2 are from commercial product HP ProLiant ML110 G5 servers (2660 MIPS CPU, 4GB MEM) [7]. In both experiments, each server is set to have 1GB/s bandwidth and 720GB disk storage capacity. In both experiments, we used the Google trace [26] which records the resource requirements and usage of tasks every 5 minutes, and we transformed the 5-minute trace into 20-second trace and we set the CPU, MEM and storage consumption for each job based on the Google trace [26]. 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 job in the trace. To fully verify the performances of our method and the other four methods, we varied the number of jobs from 300



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

Fig. 8: Utilizations of different resource types vs. number of jobs of different methods on a real cluster.

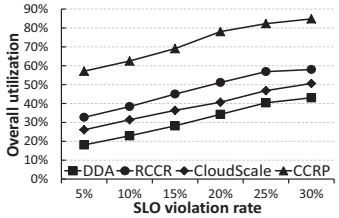


Fig. 9: Resource utilization vs. SLO violation rate on a real cluster.

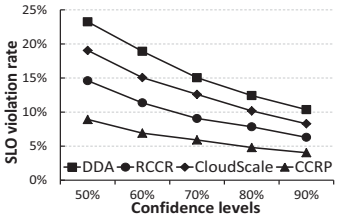


Fig. 10: SLO violation rate vs. confidence levels on a real cluster.

to 1500 with step size of 300. Table III shows the parameter settings in our experiment unless otherwise specified.

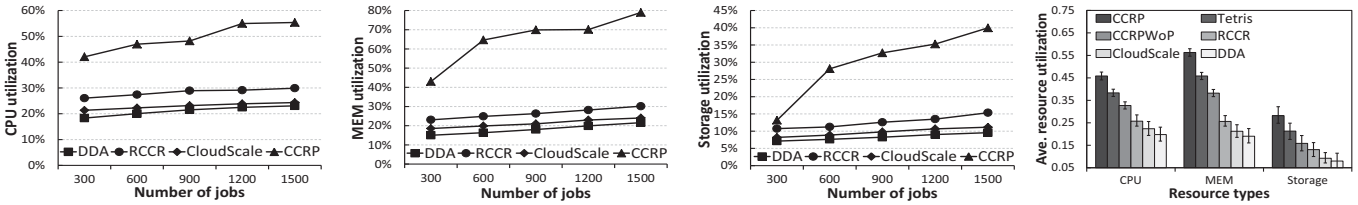
A. Experimental Results on the Real Cluster

Fig. 7 shows the prediction accuracy of different methods. We see that the prediction accuracy follows CCRP>RCCR>CloudScale>DDA. The prediction accuracy in RCCR is lower than that in CCRP because CCRP takes advantage of deep learning which can detect complex interactions among features and 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. Moreover, CCRP adequately considers the fluctuations of the amount of unused resource caused by the bursts of jobs' resource demands and utilizes SMM to correct the prediction errors, which increases the prediction accuracy. RCCA uses a time-series based forecasting of which the accuracy relies on the existence of resource usage patterns, to predict the unused resource, which can decrease prediction accuracy. CloudScale has lower prediction accuracy than CCRP and RCCR because CloudScale's prediction accuracy relies on the existence resource usage patterns in short jobs. Also, CloudScale does not utilize confidence levels to increase the prediction accuracy by increasing the probability of making accurate prediction. DDA has the lowest prediction accuracy because DDA's prediction accuracy also relies on the resource usage patterns in short jobs, and it does not have a strategy to handle prediction errors. Also it does not utilize confidence levels to improve the prediction accuracy.

Fig. 8 shows the relationship between the resource utilization and the number of jobs. We observe that the resource utilization follows CCRP>RCCR>CloudScale>DDA. The resource utilization in CCRP is higher than that in RCCR because CCRP leverages complementarity of jobs' demands on different resource types and uses a job packing strategy to reduce the resource fragmentation. Also, CCRP uses deep learning to predict the amount of unused resource, and adequately considers the fluctuations of the amount of unused resource, and CCRP uses SMM 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. However, RCCR uses a time series forecasting to predict the unused resource for long-term service jobs which is not suitable for short 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 jobs, which can increase the error rate and thereby increase the probability of over-provisioning. The resource utilizations in CCRP and RCCR are higher than that in CloudScale and DDA. This is because CCRP 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. DDA has the lowest resource utilization among all the methods because DDA 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.

To test the effects of job packing in increasing resource utilization, we also evaluated the performance of CCRPW/oP, a variant of CCRP in which job packing is not used, and Tetris, a DPA job packing strategy. Specifically, we used Equ. (2) to calculate the average resource utilization of CPU, MEM and storage with the number of jobs varying from 300 to 1500, respectively. Fig. 8(d) shows the average resource utilization of different resource types in different methods. We see that the average resource utilization follows CCRP>Tetris>CCRPW/oP>RCCR>CloudScale>DDA. The average resource utilization of CCRPW/oP is lower than CCRP, Tetris and higher than RCCR. This is because both CCRPW/oP, Tetris and CCRP allocate the resource to jobs in an opportunistic approach, which can increase the resource utilization. However, CCRPW/oP does not utilize the job packing strategy to reduce the resource fragmentation, which is considered in CCRP and Tetris, and Tetris adopts a DPA for job packing which is less efficient than that of CCRP for reducing resource fragmentation.

Fig. 9 shows the relationship between the overall resource utilization and the SLO violation rate on the real cluster. We



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

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

find the overall resource utilization increases as the SLO violation rate increases 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 given an SLO violation rate, the overall resource utilization follows CCRP>RCCR>CloudScale>DDA due to the same reasons in Fig. 8.

Fig. 10 shows the relationship between the SLO violation rate and the confidence levels in the real cluster. From Fig. 10, 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 CCRP<RCCA<CloudScale<DDA. This is because CCRP utilizes deep learning to accurately predict the amount of unused resource. Also, CCRP adequately considers the fluctuations of the amount of unused resource which can result in prediction errors and further lead to SLO violation, and uses SMM 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, DDA does not have a strategy to handle prediction errors.

B. Experimental Results on Amazon EC2

Fig. 11 shows the relationship between the resource utilization and the number of jobs on Amazon EC2. Similarly, we see the resource utilization increases as the number of jobs increases, and the resource utilization follows CCRP>RCCR>CloudScale>DDA due to the same reasons explained in Fig. 8. By examining Figs. 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. Fig. 11(d) shows the average resource utilization of different resource types. In Fig. 11(d), we also see that the average resource utilization follows: CCRP>Tetris>CCRPW/oP>RCCR>CloudScale>DDA due to the same reasons explained in Fig. 8(d).

Fig. 12 shows the relationship between the overall resource utilization and the SLO violation rate on Amazon EC2. Fig. 12 mirrors Fig. 9 due to the same reasons. Fig. 13 shows the relationship between the SLO violation rate and the confidence level on Amazon EC2. We also find that the SLO violation rate decreases as the confidence level increases and the SLO violation rate follows CCRP<RCCA<CloudScale<DDA due to the same reasons explained in Fig. 10.

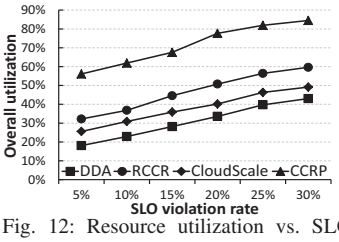


Fig. 12: Resource utilization vs. SLO violation rate on Amazon EC2.

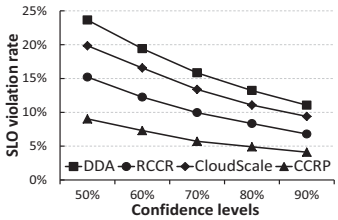


Fig. 13: SLO violation rate vs. confidence levels on on Amazon EC2.

V. Related Work

The current resource allocation approaches used in clouds are typically reservation-based [43] or demand-based [38]. To increase resource utilization in a cloud system, previous works [44, 1] and product [45] provide methods of reallocating allocated unused resources to new jobs opportunistically. Marshall *et al.* [44] proposed reusing unused resources by offering leases in an opportunistic and preemptible way with no SLO guarantees. Amazon EC2 Spot Instances [45] offers opportunistic resources with no SLO guarantees. Although the above two approaches can increase the resource utilization, they do not ensure SLOs, which affects the QoS because the allocated resource can be preempted at any time. Recently, Carvalha *et al.* [1] proposed an approach to provide a portion of the unused resources with long-term availability SLOs. This approach uses time series forecasting with the assumption of the existence of resource usage patterns in short jobs. However, this approach cannot effectively handling resource provisioning of short jobs because short jobs usually do not have certain resource usage patterns. Also, it neglects the fluctuations of the amount of unused resource caused by time-varying resources demands of short jobs. Moreover, this method may lead to resource wastage because it neglects jobs' diverse resource intensities and may cause resource fragmentation.

Many other works on resource provisioning also have been proposed to improve the resource utilization. The works [29, 43, 38] 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, CCRP not only considers the resource intensities of jobs' resource demands but also takes into account the diversities of jobs' sizes for job packing; CCRP packs the same type jobs with complementary resource

requirements to VMs to reduce the resource fragmentation and increase the resource utilization. Moreover, CCRP 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 jobs. CCRP additionally considers the fluctuations of the amount of unused resource and uses SMM model to correct prediction errors. Also, CCRP uses the hybrid resource allocation for heterogeneous jobs and customizes SLO availability for different job types so that CCRP can achieve high resource utilization and low SLO violation rate. Thus, our proposed method CCRP can fully utilize the resource while reducing SLO violation rate.

VI. Conclusions

In this paper, we propose customized cooperative resource provisioning scheme (CCRP) in clouds to increase the resource utilization and reduce SLO violation rate by customizing SLO availability and offering different degrees of SLO availability for different jobs types. CCRP uses the hybrid resource allocation strategy for the heterogeneous jobs in clouds. Also, CCRP additionally considers the fluctuations of the amount of unused resource and it combines the deep learning algorithm with the SMM model to accurately predict the amount of unused resource, and it increases the resource utilization and reduces the SLO violation rate by avoiding resource over-provisioning and under-provisioning. To further increase the resource utilization, CCRP leverages the complementarity of jobs' requirements on different resource types and jobs' heterogeneity in job size, and packs complementary jobs with the same job type to the same VM. Our extensive experimental results based on a real cluster and Amazon EC2 show that our method achieves 50% higher resource utilization and 50% lower SLO violation. Our proposed method can assist cloud service providers in efficiently utilizing their resource.

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

[2] Y. Chen, S. Alspaugh, and R. Katz. Interactive analytical processing in big data systems: across-industry study of mapreduce workloads. In *Proc. of VLDB*, 2012.

[3] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. The case for evaluating mapreduce performance using workload suites. In *MASCOTS*, 2011.

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

[5] J. Liu and H. Shen. A popularity-aware cost-effective replication scheme for high data durability in cloud storage. In *IEEE Big Data*, 2016.

[6] C. Delimitrou and C. Kozyrakis. HCloud: Resource-efficient provisioning in shared cloud systems. In *Proc. of ASPLOS*, 2016.

[7] J. Liu, H. Shen, and L. Chen. CORP: Cooperative opportunistic resource provisioning for short-lived jobs in cloud systems. In *Proc. of IEEE Cluster*, 2016.

[8] J. Mace, P. Bodik, R. Fonseca, and M. Musuvathi. Retro: Targeted resource management in multi-tenant distributed systems. In *Proc. of NSDI*, 2015.

[9] A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica. Hierarchical scheduling for diverse datacenter workloads. In *Proc. of SoCC*, Santa Clara, 2013.

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

[11] Z. Li, H. Shen, W. Ligon, and J. Denton. An exploration of designing a hybrid scale-up/out hadoop architecture based on performance measurements. *TPDS*, PP(99):1–1, 2016.

[12] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. Multi-resource packing for cluster schedulers. In *SIGCOMM*, 2014.

[13] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz. Wrangler: Predictable and faster jobs using fewer resources. In *SoCC*, 2014.

[14] X. Ren, G. Ananthanarayanan, A. Wierman, and M. Yu. Hopper: Decentralized speculation-aware cluster scheduling at scale. In *Proc. of ACM SIGCOMM*, London, 2015.

[15] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica. HUG: Multi-resource fairness for correlated and elastic demands. In *Proc. of NSDI*, 2016.

[16] D. Arteaga, J. Cabrera, J. Xu, S. Sundararaman, and M. Zhao. Cloudcache: On-demand flash cache management for cloud computing. In *Proc. of FAST*, 2016.

[17] E. J. Friedman, A. Ghodsi, and C.-A. Psomas. Strategyproof allocation of discrete jobs on multiple machines. In *Proc. of ACM EC*, 2014.

[18] I. Pietri, G. Juve, E. Deelman, and R. Sakellariou. A performance model to estimate execution time of scientific workflows on the cloud. In *WORKS'14*, pages 11–19, 2014.

[19] CPLEX linear program solver. http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/ [accessed in Aug. 2016].

[20] T. Patki, D. K. Lowenthal, A. Sasidharan, M. Maiterth, B. L. Rountree, M. Schulz, and B. R. de Supinski. Practical resource management in power-constrained, high performance computing. In *HPDC*, 2015.

[21] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. Hacigümüs. Activesla: A profit-oriented admission control framework for database-as-a-service providers. In *Proc. of SoCC*, Cascais, October 2011.

[22] V. N. Vapn. *The nature of statistical learning theory*. Springer-Verlag, 1995.

[23] Y. Liu, J. Bian, and E. Agichtein. Predicting information seeker satisfaction in community question answering. In *Proc. of SIGIR*, 2008.

[24] 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.

[25] A. Ganapathi, Y. Chen, A. Fox, R. H. Katz, and D. A. Patterson. Statistics-driven workload modeling for the cloud. In *SMDB*, 2010.

[26] Google trace. https://code.google.com/p/googleclusterdata/.

[27] 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.

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

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

[30] 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.

[31] B. Wu, H. Shen, and K. Chen. Exploiting active sub-areas for multi-copy routing in vdtns. In *Proc. of ICCCN*, 2015.

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

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

[34] 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.

[35] 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.

[36] J. Liu, H. Shen, and H. Hu. Load-aware and congestion-free state management in network function virtualization. In *Proc. of ICNC*, 2017.

[37] 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.

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

[39] 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.

[40] 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.

[41] Palmetto cluster. http://citi.clemson.edu/palmetto/.

[42] Amazon EC2. http://aws.amazon.com/ec2 [accessed in Aug. 2016].

[43] C. Curino, D. E. Difallah, C. Douglas, S. Krishnan, R. Ramakrishnan, and S. Rao. Reservation-based scheduling: If you're late don't blame us! In *Proc. of SoCC*, Seattle, November 2014.

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

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