2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid) | 978-1-7281-9586-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CCGrid51090.2021.00028

Machine Learning for Load Balancing in Cloud Datacenters

Rakshita Kaulgud Ramesh, Haoyu Wang, Haiying Shen and Zhiming Fan
Department of Computer Science
University of Virginia
Charlottesville, USA
{rrk7pb, hw8c, hs6ms, zf7ja}@virginia.edu

***Abstract*—In the cloud datacenter, the resource utilization of different virtual machine (VM) and physical machine (PM) varies with time and it may lead to SLO violation and then degrade the application performance. In order to minimize the probability of SLO violation, load balancing is used to dynamically migrate VMs from overloaded PMs to underloaded PMs. Previous load balancing methods fail to achieve long term load balance. To address this problem, in this paper, we propose different load balancing methods and evaluate their performance on several metrics. We use the Fast Fourier Transform (FFT) method, an improved FFT method considering more frequencies in FFT and the long short term memory (LSTM) machine learning model to predict the resource utilization of VM and PM in the future. LSTM can always achieve the best prediction performance in the prediction. Taking advantage of the ML technique, we then propose a heuristic based method and a reinforcement learning (RL) based method relying on ML workload prediction to generate the VM migration plan in the datacenter. We conduct experiments in both trace-driven simulation (based on Google cluster trace, PlanetLab trace, Worldcup trace) and real implementation in terms of the SLO violation rate, the number of migrations and overhead. The experimental results show that the workload prediction helps reduce the SLO violation rate and/or the number of migrations, which improves the load balance performance in a datacenter. Also, the RL based VM migration method outperforms the heuristic based method in a heavily loaded system but does not show obvious advantages in a lightly loaded system.**

***Index Terms*—Load balancing, Cloud computing, Reinforcement learning, Virtual machine**

I. Introduction

Cloud computing provides various IT services and its "pay-as-you-go" pricing policy makes it convenient for customers to connect to the IT services flexibly and cost-efficiently. Generally, cloud providers use hardware virtualization, where the Physical Machines (PM) in a datacenter always run several Virtual Machines (VMs) with different resource demands and configurations for different customers [1], [2], [3]. Since the workload of each VM on a PM varies over time, the resource utilization of the PMs may not be stable, resulting in overloaded PMs and underloaded PMs. When a PM is overloaded, the competition among all the VMs on this overloaded PM may affect the performance of each of the VMs, resulting in Service Level Objective (SLO) violation between customers and cloud providers. Thus, in order to guarantee SLO, resource management strategy is essential, which assign and reassign different VMs to different PMs to maintain the SLOs while fully utilizing the PM resources. Specifically, to solve the above problem of PM workload imbalance, a strategy is to migrate VMs from the overloaded PMs to the underloaded PMs, thereby release some workload from the overloaded PMs [4], [5], [6]. A successful resource management strategy can ensure higher resource utilization and SLO guarantee in long term, thereby increasing user experience.

Previously, researchers have proposed many heuristic resource management methods, which identify overloaded PMs and migrate out some of their VMs to underloaded PMs to achieve load balance. One issue with these methods is that they still wait for a PM to get overloaded before taking actions, which possibly leads to SLO violations. Researchers then started exploring prediction based methods, which predict workload based on historical workload data to proactively identify overloaded PMs and migrate out VMs from these PM. A challenge here is to increase the accuracy of the workload prediction. Many previous methods use machine learning (ML) algorithms to perform workload prediction and proactively migrate VMs in resource management. We find that simply relying on an ML-based prediction algorithm for workload prediction and then migrating VMs out of the predicted overloaded PMs is not a sufficiently effective way to solve the load imbalance problem. It is mainly because different ML-based prediction methods have different prediction accuracies and low-accuracy predictions might be counterproductive and lead to more SLO violations. Additionally, since the workload of each VM and each PM keeps varying, the VM migration planning on which VMs in an overloaded PM need to migrate out and which PM that a selected VM needs to migration to needs to adapt to the varied workload patterns in order to maintain the load balance state in long term.

In this paper, we study the prediction accuracy performance of Fast Fourier Transform (FFT) method, Improved FFT, and long short term memory (LSTM) ML model for workload prediction using the Google cluster trace [7], PlanetLab trace [8], and Worldcup trace [9]. We find that LSTM has the best prediction accuracy performance while generating the least time overhead. We then study two methods for VM migration: heuristic VM migration method and Reinforcement Learning (RL) based VM migration method to generate the VM migration plan, including which VM to migrated out

978-1-7281-9586-5/21/$31.00 ©2021 IEEE
DOI 10.1109/CCGrid51090.2021.00028

from the overloaded PMs, and the destination PM for the migrated out VM. We use each method for the cases with and without workload prediction. Our results show that by using workload prediction, the SLO violation rate and/or the number of migrations are reduced, which improves the load balance performance in a datacenter. In addition, the RL based VM migration method performs well in a high workload environment. In other scenarios, the RL based VM migration method does not show obvious advantages compared to the heuristic VM migration method.

The rest of the paper is organized as follows. Section II presents the related work. In Section III, we introduce the comparison between three different workload prediction methods and two VM migration methods. Section IV presents the results from trace-driven and real experiments. Section V concludes our work with remarks on our future work.

II. Related Work

In recent years, many load balancing algorithms have been proposed to reactively perform VM migration from PMs that are overloaded in a cloud datacenter. For example, Sandpiper proposed by Wood *et al.* [10] carries out dynamic monitoring and hotspot probing on PMs. It defines a volume-to-size ratio (VSR), which is used to select VMs to be migrated out and migrates the VM with maximum VSR to the least-loaded PM. Singh *et al.* [11] proposed VectorDot, a load balancing method that takes into account the multidimensional nature of workloads in a datacenter, and provides a strategy to migrate out VMs from an overloaded PM to a PM that is underutilized. Xiao *et al.* [12] presented the concept of "skewness" as a measure of unevenness in the multidimensional resource utilization of a server and aimed to reduce skewness to improve the overall utilization of server resources. The authors also presented a set of heuristics to prevent system overloading. Alsadie *et al.* [13] presented a dynamic threshold-based fuzzy approach (DTFA) to detect overloaded and underutilized PMs and the Lowest Interdependence Factor Exponent Multiple Resources predictive (LIFE-MP) approach for placing VMs on PMs. The shortcoming of these methods is that they only consider the current state of the system and fix the problem only when an imbalanced state is detected or when a PM is overloaded, resulting in higher delay for load balancing and higher SLO violations in the long term.

Recently, some algorithms have been proposed to predict VM resource demand and perform VM migrations proactively. In the proactive load balancing algorithms, when a PM is predicted to be overloaded in the immediate future based on the predicted resource demands of its hosted VMs, it moves out some VMs to prevent the host PM from being overloaded. Researchers have tried various algorithms to build the demand prediction models. Sharma *et al.* [14] employed linear programming; Bobroff *et al.* [15] used a sliding window of data to make dynamic prediction; Beloglazov *et al.* [16] introduced a Markov model. Cortez *et al.* [17] characterized Azure's VM workload to demonstrate how the VM characteristics can be utilized for better resource management. They also introduced a system called Resource Central that relies on prediction results from algorithms such as FFT, which make use of the detected VM characteristics. Guo *et al.* [18] and Hua *et al.* [19] used various forms of LSTM for time series forecasting. Liuhua *et al.* [20] presented a Resource-efficient Predictive Resource Provisioning system in clouds (RPRP) that relies on FFT to predict resource demand in a periodic pattern in order to avoid resource over-provisioning. Chen *et al.* [21] proposed a deep Learning based Prediction Algorithm (L-PAW), that integrates top-sparse auto-encoder and gated recurrent unit (GRU) block into RNN to achieve adaptive and accurate prediction of workloads that are highly variable, thereby resulting in lower resource wastage and lower SLO violations. Kumar *et al.* [22] used LSTM based workload prediction model to explore its efficacy in workload prediction for datacenter workloads along with exploring the benefits of employing such workload prediction both in terms of efficient resource scaling and energy consumption, thereby helping to achieve green computing. Janardhanan *et al.* [23] studied the performance of LSTM Network and evaluated the results with the performance of a more traditional forecasting model - AutoregRessive Integrated Moving Average (ARIMA). This study shows that the LSTM models generally perform more consistently than ARIMA due to their ability to learn non-linear data better than ARIMA models. Ding *et al.* [24] used predicted resource utilization and Performance-to-power Ratio (PPR) of heterogeneous hosts in order to ensure dynamic VM consolidation and balance of workload and energy. Waschneck *et al.* [25] introduced Industrie 4.0, a decentralized, self-organizing and self-learning system for production control, based on the application of Google DeepMind's Deep Q Network(DQN) for production scheduling. Zhou *et al.* [26] introduced and showed the effectiveness of an RL algorithm for adaptive resource management of differentiated services in geo-distributed datacenters, for balancing Quality of Service (QoS) and power consumption. Telenyk *et al.* [27] proposed an RL based method that determines an optimal policy for deciding whether a PM should be active or asleep and VM placement to minimize energy consumption and achieve load balance. However, these methods cannot effectively balance the workload in a datacenter in long term. It is a challenge for an algorithm to choose which VMs to migrate out from an overloaded PM, and to choose a destination PM to host a migration VM. It will generally introduce additional delay and overhead if an inappropriate VM is chosen for VM migration. To alleviate this problem, Shen and Chen [28] proposed a Markov decision process (MDP) algorithm to teach a PM how to select VMs to migrate out for maintaining long-term load balance. Our work further improves this method by adding the workload prediction component and using deep RL algorithm.

III. Load Balancing Methods

In this section, we introduce several strategies for proactive resource management. We explore three prediction methods, FFT (that considers low frequency components), Improved-FFT (that considers both low and high frequency components),

and LSTM (machine learning based prediction method). We find that LSTM achieves higher accuracy than the two FFT-based methods. We then study two methods for VM migration decision making based on the predicted workloads: a heuristic based VM migration method and an RL based migration method.

A. Data Preparation

In this paper, we conduct both trace driven simulation based experiments as well as real cluster setup based experiments. For trace-driven experiments, we used three traces: Google Cluster [29], PlanetLab [8] trace and WorldCup trace [9].

The Google cluster trace [29] records resource usage on a cluster for about 11000 machines from May 1st 2011 for 29 days. We extract 1000 different tasks' traces with 1000 timestamps from the Google cluster trace, and get the CPU utilization for every 5 minutes interval. We use the processed data in the experiment as workload generation and we considered the workload trace of each task as a VM workload trace in the experiment. The PlanetLab trace is from the CloudSim simulation tool [8]. The PlanetLab trace contains CPU utilization measured with every 5 minutes interval for 24 hours on 10 random days between March and April 2011. It records CPU utilization as a percentage value for a total of 3000 VMs. The Worldcup trace [9] records the request information for the 1998 World Cup website. The trace logs on this site were collected from April 30, 1998 to July 26, 1998, for a period of 88 days comprising a total of 1,352,804,107 requests. These requests were handled by 33 servers placed in four locations: one in France, and other three in USA. The trace is in binary format, and each request in the trace contains the request size and the timestamp for this request. We processed this data to compute the number of requests made to each server every five minutes.

B. Workload Prediction

After getting the preprocessed VM workload data, we build three workload prediction models that use the historical VM workload data to predict the future workloads of the VM in the cloud. The VM workload includes multiple types of resources such as CPU, memory, and network bandwidth. In this paper, we use the CPU resource utilization percentage as an example to show the prediction accuracy performance.

Fast Fourier Transform (FFT) Prediction Model: We follow the previous work [20] to implement the FFT prediction method for the VM workload prediction in the cloud computing environment. FFT is used to decompose historical workload usage data into a number of different frequency components, f_n. These frequency components represent the repeated workload usage patterns in the historical data. To predict the workload usage at a future time t_{N+i}, FFT sums up the low-frequency components at time t_{N+i} and converts them from the frequency domain back to the time domain in order to make the prediction. The pseudocode of the FFT prediction model is shown in Algorithm 1. The algorithm takes historical workload data as an input (line 1). It first determines the length of dominant repeating pattern (line 3), L, which in our case is the time lag that we are considering for prediction. It is equal to 8. The algorithm loops over L times (line 4) and executes the equation in line 6 for each data entry to convert each workload CPU utilization value from time domain to frequency domain. The results in a list, f_n, where each entry corresponds to the list of frequencies representing corresponding workload CPU utilization of a VM at that timestamp. Finally, the algorithm takes the sum of average inverse FFT (line 9), iterating over the first half of the length of the dominant repeating pattern, L. The sum, $\hat{p}_{t_{N+i}}$, is the predicted value and is the output of the algorithm (line 10).

Algorithm 1: FFT workload prediction pseudocode

1 **Input**: Historical workload usage data $\mathcal{U}$.
2 **Output**: Estimated workload usage $\hat{p}_{t_{N+i}}$ at t_{N+i}.
3 Calculate autocorrelation of $\mathcal{U}$; Determine length of dominant repeating pattern L;
4 **for** $k = 0 \rightarrow L-1$
5 **do**
6 $\quad f_n = \sum_{n=0}^{L-1} u_{t_{N-L+i+n}} \cdot e^{-j2\pi \frac{k}{L} n}$;
7 **for** $k = 0 \rightarrow L/2$
8 **do**
9 $\quad \hat{p}_{t_{N+i}} += \frac{1}{L} \sum_{k=0}^{L/2} f_n \cdot e^{j2\pi \frac{k}{L}(N+i)}$;
10 **return** $\hat{p}_{t_{N+i}}$.

Improved FFT Prediction Model: Since the FFT method explores only the low-frequency components to make predictions, it misses out some useful high-frequency components, which affects its prediction accuracy. To address this issue, in the improved FFT model, along with the predicted value from FFT, $\tilde{p}_{t_{N+i}}$, we make use of a second-order auto regressive predictor (AR(2)), which better captures high-frequency residuals $r_{t_{N+i}}$. The final prediction results equal to $\hat{p}_{t_{N+i}} = \tilde{p}_{t_{N+i}} + r_{t_{N+i}}$. Therefore, we adopt the improved FFT prediction model in [30], which takes the high-frequency components into account for workload prediction. The only difference between FFT and improved FFT is whether the prediction considers the high-frequency components using the second-order auto-regressive process (AR(2)). The pseudocode of the improved FFT prediction model is shown in Algorithm 2. This algorithm takes historical data as an input (line 1). Lines 1-10 of the algorithm follow the same steps as in Algorithm 1 and calculate the predicted value as per FFT. In addition to these steps, in line 11, for the second half of the length of dominant repeating pattern L, the inverse FFT value is computed and the sum of these inverse FFT values is stored in the variable $\tilde{r}_{t_{N+i}}$. This variable, $\tilde{r}_{t_{N+i}}$, is passed to the second-order Autoregressive AR(2) model (line 12), which returns the value $r_{t_{N+i}}$ as the result. Finally, in line 13, the algorithm sums the result given by the FFT model, $\tilde{p}_{t_{N+i}}$ and the result from Autoregressive model, $r_{t_{N+i}}$, to get the final result of the improved FFT model and returns this value.

LSTM Prediction Model: We also use time series analysis

Algorithm 2: Improved FFT workload prediction pseudocode

Input: Historical workload usage data $\mathcal{U}$.
Output: Estimated workload usage $\hat{p}_{t_{N+i}}$ at t_{N+i}.
Calculate autocorrelation of $\mathcal{U}$;
Determine length of dominant repeating pattern L;
for $k = 0 \rightarrow L-1$
do
  $f_n = \sum_{n=0}^{L-1} u_{t_{N-L+i+n}} \cdot e^{-j2\pi \frac{k}{L} n}$;
for $k = 0 \rightarrow L/2$
do
  $\tilde{p}_{t_{N+i}} += \frac{1}{L} \sum_{k=0}^{L/2} f_n \cdot e^{j2\pi \frac{k}{L}(N+i)}$;
  $\tilde{r}_{t_{N+i}} += \frac{1}{L} \sum_{k=L/2}^{L} f_n \cdot e^{j2\pi \frac{k}{L}(N+i)}$;
Calculate the residual component $r_{t_{N+i}}$ based on AR(2) by passing $\tilde{r}_{t_{N+i}}$ as input;
return $\hat{p}_{t_{N+i}} = \tilde{p}_{t_{N+i}} + r_{t_{N+i}}$.

method to build the prediction model. We build an LSTM model to predict the future workload of each VM. For the time-series data of VM workload, we use the previous workload trace data as the input to the model, and the model predicts the workload at the next timestamp. We train and evaluate our LSTM model on historical trace data, with the input time interval set to 5 minutes. We further get the workload prediction of each PM by summing up the predicted workload of each VM hosted on the corresponding PM at each timestamp.

C. Heuristic Based VM Migration Method

A VM migration policy is needed as a part of a load balancing algorithm to avoid PM overload. The current workload of a PM is calculated by summing up the workload values of all the VMs hosted on the PM. A PM-VM matching list records the VMs that are hosted in each PM. During migration matching, a VM is reallocated/migrated from the PM on which it is currently hosted to a more suitable PM, and the PM-VM matching list is updated accordingly.

First, we introduce a heuristic based migration policy. For the load balancing and workload migration, if a PM is overloaded on at least one resource type and at least one time period T, we consider this PM is overloaded. Then, the workload on this PM needs to be migrated to other underloaded PMs in order to make this PM not overloaded. Since there are many types of resources for one VM and for one PM, a PM needs to evaluate the workload of each type of resource in itself and also in the VMs on it. At the beginning of one time period, the prediction model predicts the resource utilization of each VM and PM in the next time period and the load balancer makes the load balance decisions according to the predicted results. We use the resource consumption ratio over a PM's resource capacity to represent the resource utilization of a PM and of a VM. The capacity of the r^{th} type of resource of the k^{th} PM is denoted by $C_r^{P_k} = 1$. The workload of VM v_j (hosted in PM P_k) on the r^{th} type of resources in the time period is denoted by $a_r^{v_j}$. We use V_{P_k} to denote the set of VMs running in PM P_k. Then, the workload of PM P_k in this time period is the sum of all the workloads of VMs on this PM:

$$A_r^{P_k} = \sum_{v_j \in V_{P_k}} a_r^{v_j} \tag{1}$$

where V_{P_k} represents the VM list on PM P_k. We use τ (e.g., 95%) to denote the overload threshold. For PM P_k with r^{th} resource, if $A_r^{P_k} > \tau$ in this time period, PM P_k is overloaded in this time period for the r^{th} type of resource. For all the PMs in the cluster, each overloaded PM needs to select its excess workload, that is, to migrate out some VMs to avoid being overloaded.

In this heuristic based method, each PM periodically checks if it will become overloaded in the next time period based on the VM workload prediction using the above prediction method. If a PM predicts that it will be overloaded, the PM migrates out the VMs contributing more workloads in the predicted overloaded time period. For the destination PM of the selected VMs to migrate out, our method selects the PM that is more underloaded in the predicted time period. After the load balance execution, each PM is less likely to be overloaded or underloaded in each epoch during the predicted time period. Next, we introduce how to select the VMs from the overloaded PMs. It is possible that a PM is overloaded on certain types of resources in some time periods while underloaded on other types of resources. Therefore, it is necessary to select the VMs that contribute more workload on the certain overloaded resource types when the PM is overloaded. In this way, the overloaded PM can relieve its workload to avoid PM overload and also use more resources as much as possible. We next introduce how to calculate the priority for each VM on an overloaded PM to be migrated out. For the r^{th} type of resource on the overloaded PM P_k in this time period, we use the difference between $A_r^{P_k}$ and $C_r^{P_k}$ to represent the overload level: $U_r^{P_k} = A_r^{P_k} - C_r^{P_k}$. A higher $U_r^{P_k}$ means a higher overload level on the r^{th} type resource of a PM. When the r^{th} type of resource is overloaded, a VM that has a higher $a_r^{v_j}$ should have a higher priority to migrate out. Thus, for a predicted overloaded PM that will be overloaded in the r^{th} type of resource, if a VM has a higher $A_r^{P_k} \cdot a_r^{v_j}$ value, it should have a higher priority to be migrated out from a predicted overloaded PM.

Since we consider multiple types of resource together, we assume there are R types of resource in the system. We use a R dimensional vector $\vec{a}^{v_j} = < l_1^{v_j}, ...l_r^{v_j}, ...l_R^{v_j} >$ to represent the workload of VM v_j in the predicted time period for all types of resources, and use $\vec{U}^{P_k} = < U_1^{P_k}, ...U_r^{P_k}, ...U_R^{P_k} >$ to represent the overload level of PM P_k. Thus, we define the VM migration priority score of VM v_j on PM P_k in the predicted time period as the dot product of the above two vectors:

$$S_{P_k}^{v_j} = \vec{U}^{P_k} \bullet \vec{a}^{v_j}. \tag{2}$$

According to the equation above, in the system, the VM

with a larger $S_{P_k}^{v_j}$ has a higher priority to be migrated out in order to more quickly relieve the excess workload in a predicted overloaded PM. That is, in a predicted overloaded PM, the VMs that contribute more workload for the overloaded resource types on the overloaded PM when it is overloaded have higher priorities to be selected to be migrated out. Each predicted overloaded PM sorts the VMs on it in the descending order of the priority scores. The PM then selects the VMs from the top of the sorted list one by one and removes this VM from the list until it is not overloaded. The PM then reports its selected migration VMs to the load balancer to be reallocated to underloaded PMs. In the following, we present how the load balancer selects the destination PM for each migration VM.

To select the destination for the migration VMs, in order to achieve the load balance faster, the heuristic VM migration method tends to migrate out the VMs with higher workload first. Additionally, considering the multiple types of resources, we should migrate the VMs with the highest workload on some types of resources to the PMs with the lowest resource utilization on these types of resources. Therefore, to measure the workload of migration VMs, we first define the workload level as below:

$$A^{v_j} = |\vec{a}^{v_j}| \tag{3}$$

which considers all types of resources. The centralized load balancer sorts all the migration VMs in descending order of their workload levels, selects the VMs from the top of the sorted VM list one by one, and then finds a destination PM for each VM. For each migration VM, its destination PM is the PM that has the highest matching score $S'^{v_j}_{P_k}$ among all the predicted underloaded PMs in the next time period, which is calculated by:

$$S'^{v_j}_{P_k} = -\vec{U}^{P_k} \bullet \vec{a}^{v_j}. \tag{4}$$

If a PM has lower resource utilization on the types of overloaded resources when a VM is overloaded in these resources, the PM has a higher matching score $S'^{v_j}_{P_k}$ with the VM and thus is a better option as the destination of the VM. After the heuristic VM migration method conducts VM migration from the predicted overloaded PMs to the predicted underloaded PMs, the load balance is achieved.

D. Reinforcement Learning Based Migration Method

We design an RL based method to handle the resource management and VM migration in the datacenter. We use an RL agent to generate the VM migration decisions. At each timestamp, for each PM, based on the current environment, the RL agent hosted in the PM chooses an action (which VMs to migrate out and which VMs to accept to host) to interact with the environment. An RL consists of states, actions, rewards, and current time, which is described as a tuple (S,A,R,T). At each timestamp, the agent chooses the action based on the current state measured from the environment. We explain the RL's <S, A, T, R> tuple in the following.

States (S): In our work, we first classify the resource utilization to three levels $L = \{High, Medium, Low\}$ and define the heavily loaded threshold H_1 (e.g., 90%) and lightly loaded threshold H_2 (e.g., 40%) to distinguish the different levels. A PM or a VM has a resource utilization level for each type of resource. If the utilization of at least one resource in a PM reaches the heavily loaded threshold, this PM's state is high; only when the utilization of all resources in a PM do not reach the lightly loaded threshold, this PM's state is low, and otherwise, the PM's state is Medium. The same applies to a VM. Here, the utilization of the r^{th} resource of a VM is the ratio between its load and assigned resource of the r^{th} resource. A PM's state or a VM's state includes its own resource utilization levels for all types of resources such as <CPU-high, Mem-Medium> and <CPU-median, Mem-low>. Therefore, if there are n types of resources $\mathcal{R} = \{r_1, r_2, ..., r_n\}$ in the system, the total number of states of VMs or PMs equals to $|L|^R$. An RL agent is hosted in each PM, and it checks the PM's state and then takes actions accordingly.

Actions(A): In order to reduce the number of actions in the system, as in [28], we define an action as migrating out a VM in a certain state (VM-state), accepting a migration VM in a certain state or no action, rather than migration out or accepting a particular VM. When a PM detects that it is overloaded or underloaded, it activates its RL agent. Based on the PM's state, the RL agent selects an action from the action set A. That is, if the PM is overloaded, the RL agent will determine the state of a VM that needs to be migrated out, and if the PM is underloaded, the RL agent will determine the state of a migration VM that the PM can accept to host. This process is repeated until the PM will not be overloaded or underloaded. Then, the overloaded PM sends the information of its selected migration VMs to the load balancer, and the underloaded PM sends the information of the migration VMs that it can accept to the load balancer. Finally, the load balancer matches the migration VMs to the destination PMs that will host them.

Rewards: The reward is used to give feedback according to the action. Since the goal is to minimize the SLO violations in the datacenter, the reward is designed based on the difference of SLO violations before and after taking the action. The reward function is described as:

$$Reward = SLO\_violation\_before - SLO\_violation\_after$$

Agent & Policy Network: The policy determines action $a \in A$ given state $s \in S$ to maximize the expected reward. An RL agent selects the actions based on the policy. The policy or the probability distribution over action $\pi(s, a)$ represents the probability that action a is taken in state s. The traditional Q-learning based RL [31] cannot store all the {state, action} pairs when the number of PMs and VMs is large since the number of the {state, action} pairs is large. We then use deep neural network to represent the policy. θ is denoted as the parameters for training in the network and then the probability distribution is represented by $\pi_\theta(s, a)$. Figure 1 shows the relations between the RL agent and the environment. At each timestamp, the current PM state is fed to the policy network

to get the probability of different actions. The agent selects the action based on the probability distribution.

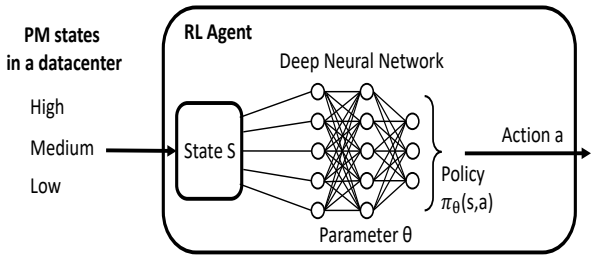


Fig. 1: Reinforcement learning agent.

Policy gradients methods: We use policy gradient strategy to learn the policy parameters in the neural network. The objective is to maximize the expected rewards. The idea of the policy gradient methods is to estimate the gradient by observing the obtained trajectories of executions following the policy. Then it updates the policy parameters via gradient descent. The update rule is described as follows:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta log \pi_\theta(s_t, a_t) \nu_t, \tag{5}$$

where α is the step size. $\nabla_\theta log \pi_\theta(s_t, a_t)$ gives the gradient descent to change the parameters. ν_t decides the direction of the updates for θ and how large for the step to update.

IV. Performance Evaluation

We conducted trace-driven simulation based experiments with different traces and real implementation experiments to evaluate the performance of the proposed methods for load balancing in a datacenter.

A. Experiment Setting

1) Simulation Setting: We simulated a heavy workload cloud datacenter with 100 PMs hosting 1000 VMs (i.e., VM/PM ratio is 10) to evaluate the performance of different methods. We set the overhead threshold τ to 0.95. The VMs are randomly chosen from the task traces of Google cluster trace. At the beginning of the experiments, the VMs are randomly allocated to the PMs so that some of the PMs may be overloaded. Following which we ran different load balancing methods on the setup to evaluate their performance.

We simulated a light workload cloud datacenter with 1000 PMs and 2000 VMs as 2000 (VM/PM ratio is 2). The overhead threshold τ was set to 0.9. At the beginning of the simulation, we randomly allocated 2000 VMs to the 1000 PMs, which made a portion of the PMs overloaded. For the Worldcup trace with request information, we multiplied the request count with a scaling factor of 10 for heavy workload setup experiment (100 PMs and 1000 VMs) and 2 for light workload setup experiment (1000 PMs and 2000 VMs).

In the simulation, the resource utilization information of all PMs and VMs is updated according to the trace in consideration. We record the number of overloaded PMs. If the resource utilization of one type of resource of a PM exceeds the overload threshold, the PM is considered overloaded and leads to SLO violations. Each PM conducts the load balancing methods for VM migration at each timestamp, and the number of VM migrations were recorded. For each simulation, we simulated 500 timestamps (nearly 2500 minutes) and got the cumulative numbers of overload PMs and VM migrations to evaluate the performance of different methods. We repeatedly ran each experiment 10 times and reported the median, the 10th percentile, and the 90th percentile of the 10 results.

2) Real Implementation Setting : In the real implementation experiments, we used 7 PMs with capacities of 28 CPUs, and these 7 PMs hosted 30 VMs with capacities of 4 CPUs each. We used the KVM [32] virtualization software with Qemu [33] emulator to turn the hosts into hypervisors, and used the libvirt [34] toolkit for creating and managing the KVM VMs on hypervisor hosts. The workload on the VMs was emulated by using the 'stress' tool [35]. For the CPU 'stress' process on each VM, we randomly selected a number from 25% to 100% as the CPU utilization of each VM every 2 minutes. At the beginning of the simulation, we randomly allocated 30 VMs to the 7 PMs, which potentially overloaded some of the PMs. The overhead threshold of a PM is set to 0.9.

3) Evaluation Metrics: We use three metrics to evaluate the performance of the load balancing methods. The first metric is the SLO violation rate. SLO violation rate is determined by the percentage of time during which the CPU utilization of a PM is above 95%. We define T as the total number of time periods for the whole experiment time, and t is the number of time periods. One time period equals 2 minutes. Therefore, the calculation of SLO violation rate follows:

$$SLO\_violation\_rate = \frac{\sum_{t=1}^{T} PM\_overloaded(t)}{T * PM\_num}$$

The second metric is the number of migrations that take place during the time period of the experiment. Since VM migrations may cause latency and network bandwidth cost, load balancing algorithms should minimize the number of VM migrations while avoiding SLO violations. The number of VM migrations is calculated as follows:

$$VM\_migration\_num = \sum_{t=1}^{T} VM\_migration\_num(t)$$

The third metric is the time overhead of different load balancing algorithms. The time overhead is defined as the execution time for each algorithm. In the experiment, we calculate the average execution time over the number of time periods in the whole experiment time. The time overhead is calculated as follows:

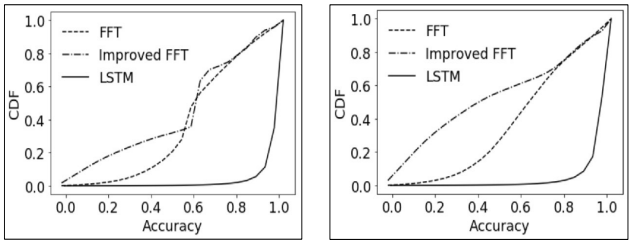
$$overhead = \frac{\sum_{t=1}^{T} execution\_time(t)}{T}$$

4) Compared Methods: We ran and compared four methods of load balancing in our experiments. The four load balancing

methods are the heuristic based method with workload prediction (Predict-Heuristic), the heuristic based method without workload prediction (Heuristic), the RL based migration method with workload prediction (Predict-RL), and the RL based migration method without workload prediction (RL). With workload prediction, the VM migration method uses the predicted workload in the next time period. Without workload prediction, the VM migration method uses the currently measured workload to achieve load balance. We also include the performance results of the system without a load balancing method (No-LB) as a reference.

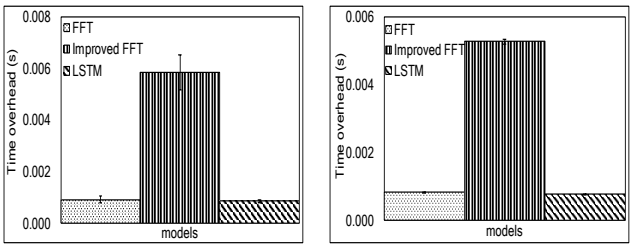
B. Experimental Results

1) Prediction Accuracy: Based on the prediction models described in Section III-B, we predict the resource utilization of each VM according to the trace data and get the accuracy of the predicted results. We train one model for all the VMs in the system. For the three different prediction models, we use half of the trace data to train the model and the remaining half of the trace data as the testing data. We use the latest 40 minutes workload as input of the model to predict the value after 5 minutes. We use the trained model to predict the resource utilization of all the VMs.



(a) CPU (b) Memory

Fig. 2: Prediction accuracy.

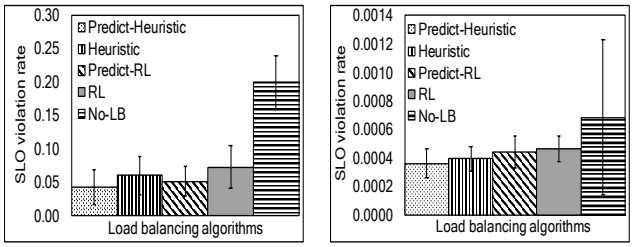


(a) CPU (b) Memory

Fig. 3: Prediction overhead.

We compare the accuracy of the workload prediction both in CPU and memory resource for all the three prediction methods. Figure 2 show the prediction accuracy CDF in CPU and memory, respectively. We can see that LSTM has the highest prediction accuracy, where a majority of its predictions could achieve around 90% accuracy. Figure 3 show the time overhead for CPU and memory prediction for all the three prediction methods. We can see that LSTM model generates 85% less time overhead compared to the improved FFT method, and 7% less overhead compared to the FFT method. Taking advantage of the machine learning technique, LSTM outperforms both FFT and improved FFT in terms of both accuracy and time overhead. In the following experiments, we always select LSTM as our workload prediction model in our experiments, where the load balancing methods use prediction.

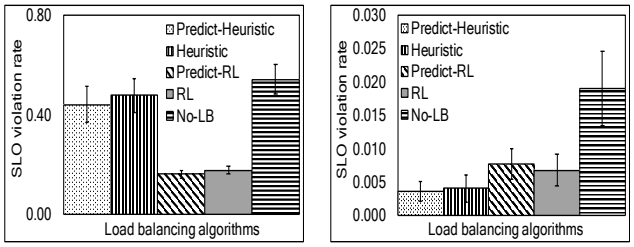
2) SLO violation rate: Figure 4(a), Figure 5(a) and Figure 6(a) show the SLO violation rate for different load balancing methods on Google cluster trace, PlanetLab trace and Worldcup trace in a heavy workload setting. In Figure 4(a), the SLO violation rate follows No-LB>Heuristic>Predict-Heuristic≈RL<Predict-RL. Here, the heuristic based and RL based migration methods can achieve similar SLO violation rate, and the prediction method can always improve the load balance performance. The common trend seen for SLO violation rate in Figure 5(a) and Figure 6(a) is No-LB>Heuristic>Predict-Heuristic>RL≈Predict-RL. Load balancing techniques that include the Heuristic based migration method perform worse than those that include the RL based migration method, because the latter can make better VM migration decisions for long term as they learn from historical data. On average, load balancing techniques that include RL based migration methods have 60% and 30% lower SLO violation rates compared to the heuristic based method in PlanetLab and Worldcup trace, respectively. From Figure 5(a) and Figure 6(a), we can conclude that the difference between the SLO violation rates on using load balancing techniques with and without prediction for the RL method is small, using workload prediction improves the heuristic based migration method, and the RL based migration method performs better than the heuristic based method. We conjecture that the RL does not exhibit higher performance over the heuristic method for the Google cluster trace because each VM's load is high, so even though the system is lightly loaded regarding the VM/PM ratio, the system load is not low considering the PM resource load. In all figures, No-LB has the worst performance since it does not have load balancing algorithm. In a conclusion, generally, the RL based method outperforms the heuristic based method, and the prediction helps improve the load balance performance. However, the particular system workload and the workload features (i.e., different traces) will affect whether the RL based method outperforms the heuristic based method and vice versa, and also whether the prediction helps improve the performance.



(a) 100PM, 1000VM (b) 1000PM, 2000VM

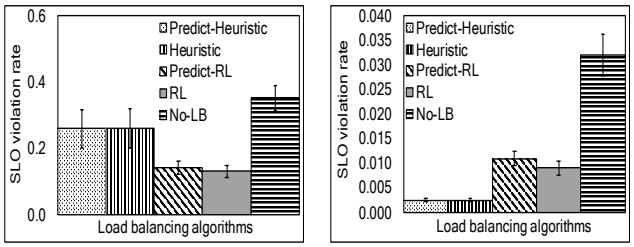
Fig. 4: SLO violation rate for Google cluster trace.

Figure 4(b), Figure 5(b) and Figure 6(b) show the SLO violation rate for different load balancing methods on Google cluster trace, PlanetLab trace and Worldcup trace in a light



(a) 100PM, 1000VM (b) 1000PM, 2000VM

Fig. 5: SLO violation rate for PlanetLab trace.



(a) 100PM, 1000VM (b) 1000PM, 2000VM

Fig. 6: SLO violation rate for Worldcup trace.

workload setting. The common trend seen for the SLO violation rate is No-LB>RL≈Predict-RL>Heuristic>Predict-Heuristic. Heuristic based methods reduce the SLO violation rate of the RL based method by 5% for Google cluster trace, 14% for PlanetLab trace and 20% for Worldcup trace. As expected, all the four load balancing methods perform better than having no load balancing (No-LB). Workload prediction shows marginal improvement in SLO violation rate when heuristic based migration method is used, and no improvement or even degraded performance when the RL based migration method is used. Because of the light workload, heuristic based migration methods can easily make better VM migration decisions with abundant underloaded PMs to host migration VMs while the RL based migration methods may conduct unnecessary VM migrations in advance in order to avoid SLO violations in long term. We conclude that heuristic based migration methods can achieve better SLO violation rate performance compared to RL based migration methods in a lightly loaded system.

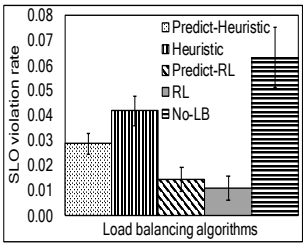
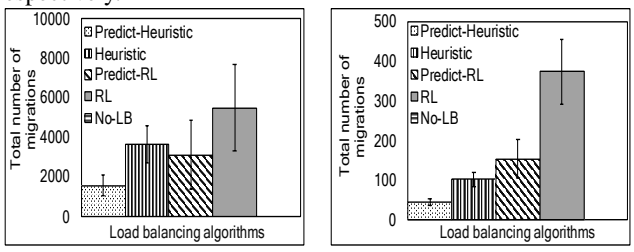


Fig. 7: SLO violation rate for real cluster experiment.

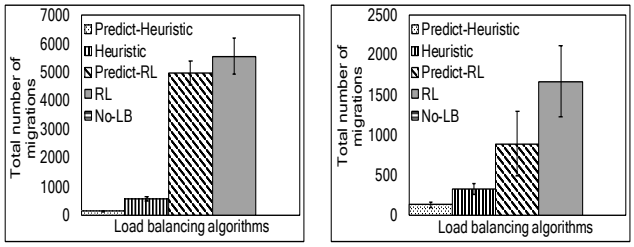
Figure 7 shows the SLO violation rate for different load balancing method on a real cluster experiment. We can see that the results follow the trend seen in Figure 5(a) and Figure 6(a) due to the same reasons.

3) The Number of Migrations: Figure 8(a), Figure 9(a), and Figure 10(a) show the number of migrations for different load balancing methods on the three traces in a heavy workload setting. The trend follows RL>Predict-RL>Heuristic>Predict-Heuristic. For Google cluster trace, PlanetLab trace, and the Worldcup trace, Predict-Heuristic method reduces migrations by 59%, 76%, 8% over Heuristic method, respectively. Predict-RL method reduces migrations by 40%, 10%, 1% over RL method, respectively. Predict-Heuristic reduces the number of migrations by 58%, 97%, 98% compared to Predict-RL method, respectively. Heuristic method reduces the number of migrations by 39%, 89%, 98% compared to RL method, respectively.



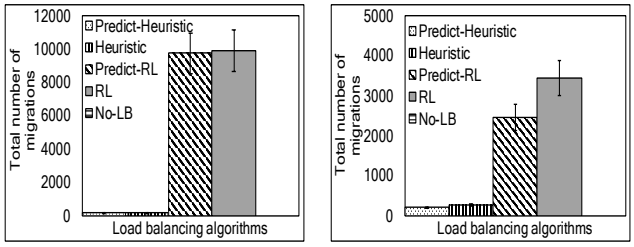
(a) 100PM, 1000VM (b) 1000PM, 2000VM

Fig. 8: The number of migrations for Google cluster trace.



(a) 100PM, 1000VM (b) 1000PM, 2000VM

Fig. 9: The number of migrations for PlanetLab trace.



(a) 100PM, 1000VM (b) 1000PM, 2000VM

Fig. 10: The number of migrations for Worldcup trace.

Figure 8(b), Figure 9(b), and Figure 10(b) show the number of migrations for different load balancing method on the three traces, in a light workload setting. The trend follows RL>Predict-RL>Heuristic>Predict-Heuristic. For Google Cluster trace, PlanetLab trace, and the Worldcup trace, Predict-Heuristic method reduces migrations by 58%, 61%, 28% over Heuristic method, respectively. Predict-RL method reduces migrations by 61%, 46%, 24% over RL method, respectively. Predict-Heuristic reduces the number of migrations by 72%, 85%, 91% compared to Predict-RL method, respectively. Heuristic method reduces the number of migrations by 69%, 80%, 92% compared to RL method, respectively.

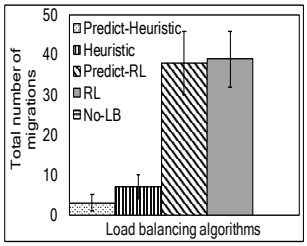
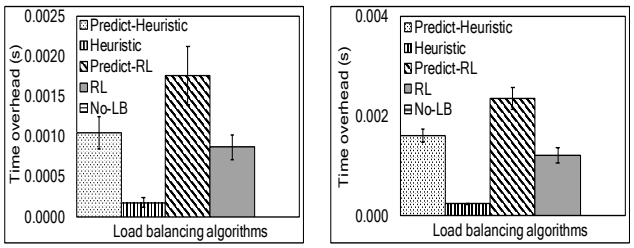


Fig. 11: The number of migrations for real cluster experiment.

Figure 11 shows the number of migrations for different load balancing methods used in experiments on a real cluster setup. We can see that the number of migrations follows RL>Predict-RL>Heuristic>Predict-Heuristic. Predict-Heuristic reduces the number of migrations by 57% compared to Heuristic method. However, in this case, Predict-RL reduces the number of migrations by 2% compared to RL. Predict-RL method increased the number of migrations by 92% compared to Predict-Heuristic method, and RL method increased the number of migrations by 82% compared to Heuristic method.

The reason for higher number of migrations for load balancing methods that include RL based migration methods is that we did not include the number of migrations in the reward function. Then, with historical data training, RL based methods migrate some VMs in advance (which may not be necessary) to avoid PM overload and then generate more VM migrations compared to the heuristic based migration methods. Additionally, both heuristic and RL based migration methods can get benefit from workload prediction in reducing the number of migrations.

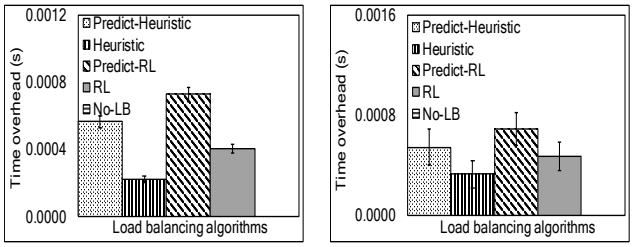
4) Time Overhead: Figure 12(a), Figure 13(a) and Figure 14(a) show the time overhead of different load balancing methods for the three traces in the heavy workload setting. These figures show that the time overhead follows Predict-RL>Predict-Heuristic>RL>Heuristic. For Google cluster trace, PlanetLab trace, and the Worldcup trace, Predict-Heuristic method adds 77%, 66%, 70% overhead over Heuristic method, respectively. Predict-RL method adds 46%, 42%, 50% overhead over RL method, respectively. Predict-Heuristic reduces the overhead by 66%, 14%, 16% compared to Predict-RL method, respectively. Heuristic method reduces the overhead by 75%, 50%, 50% compared to RL method, respectively.



(a) 100PM, 1000VM (b) 1000PM, 2000VM

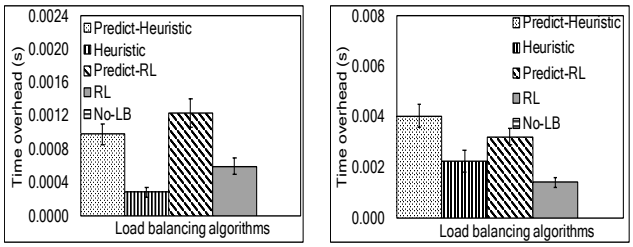
Fig. 12: Time overhead for Google cluster trace.

Figure 12(b), Figure 13(b), and Figure 14(b) show the time overhead of different load balancing algorithms on the three traces in the light workload setting. The figures show that the overhead generally follows Predict-RL>Predict-Heuristic>RL>Heuristic. For Google cluster trace, PlanetLab trace, and the Worldcup trace, Predict-Heuristic method adds 86%, 40%, 50% overhead over Heuristic method, respectively. Predict-RL method adds 45%, 28%, 66% overhead over RL method, respectively. Predict-Heuristic method reduces the overhead by 25%, 28%, 25% compared to Predict-RL respectively. Heuristic method reduces the overhead by 81%, 40%, 92% compared to RL method, respectively.



(a) 100PM, 1000VM (b) 1000PM, 2000VM

Fig. 13: Time overhead for PlanetLab trace.



(a) 100PM, 1000VM (b) 1000PM, 2000VM

Fig. 14: Time overhead for Worldcup trace.

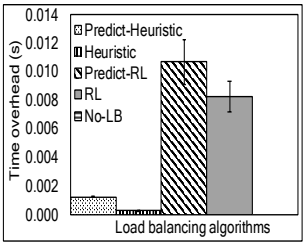


Fig. 15: Time overhead for real cluster experiment.

Figure 15 shows the time overhead of the load balancing algorithms in real cluster experiments. We can see that the time overhead follows Predict-RL>RL>Predict-Heuristic>Heuristic. Predict-Heuristic method adds 75% more overhead than Heuristic method, and Predict-RL adds 22% more overhead than RL method. Predict-Heuristic reduced the overhead by 88% compared to Predict-RL, and Heuristic method reduces the overhead by 96% compared to RL method.

As expected, the use of ML prediction model for load balancing decision making adds the time overhead of the overall load balancing methods used. This is because we use LSTM deep learning model for prediction, which generally takes long to give the outputs. Also, RL is built based on deep learning, which needs a certain time for decision making. Therefore, the load balancing method that includes both prediction and RL based migration methods has the highest time overhead

whereas the method that uses only a heuristic based migration method with no prediction has the least time overhead.

V. Conclusion

In this paper, we propose different load balancing methods to improve the performance of existing load balancing methods in a cloud datacenter, to quickly achieve long term load balance while lowering load balancing overhead. We proposed to predict resource demands of VMs and PM overload/underload, and conduct VM migration based on the prediction rather than the currently measured PM overload/underload state. We then proposed a heuristic based VM migration method and an RL based VM migration method that based on the prediction to determine the VMs to be migrated out from the predicted overloaded PMs and the destination PMs for the migration VMs. Using simulation and real implementation experiments, we evaluated the methods' performance with and without the prediction, based on different evaluation metrics, such as SLO violation rate, number of migrations, and time overhead. From the observed results, we conclude that using workload prediction improves the load balance performance regarding the SLO violations and/or the number of VM migrations. The RL based migration method performs better than the heuristic based method in highly loaded system but does not demonstrate an obvious advantage compared to the heuristic based VM migration method in a lightly loaded system. In the future, we plan to further optimize the RL based method by enhancing the reward function (e.g., minimizing the number of migrations) and using more features to improve the load balance performance.

Acknowledgment

This research was supported in part by U.S. NSF grants NSF-1827674, CCF-1822965 and Microsoft Research Faculty Fellowship 8300751, and AWS Machine Learning Research Awards.

References

[1] H. Wang, H. Shen, and G. Liu. Swarm-based incast congestion control in datacenters serving web applications. In *Proc. of SPAA*, 2017.

[2] H. Wang, H. Shen, Q. Liu, K. Zheng, and J. Xu. A reinforcement learning based system for minimizing cloud storage service cost. In *Proc. of ICPP*, 2020.

[3] H. Wang, Z. Liu, and H. Shen. Job scheduling for large-scale machine learning clusters. In *Proc. of CoNEXT*, 2020.

[4] J. Gao, H. Wang, and H. Shen. Machine learning based workload prediction in cloud computing. In *Proc. of ICCCN*, 2020.

[5] H. Wang, H. Shen, C. Reiss, A. Jain, and Y. Zhang. Improved intermediate data management for mapreduce frameworks. In *Proc. of IPDPS*, 2020.

[6] H. Wang and H. Shen. Proactive incast congestion control in a datacenter serving web applications. In *Proc. of INFOCOM*, 2018.

[7] C. Reiss, J. Wilkes, and J. Hellerstein. Google cluster-usage traces: format+ schema. *Google Inc., White Paper*, 2011.

[8] Cloudsim: A framework for modeling and simulation of cloud computing infrastructures and services. http://www.cloudbus.org/cloudsim/.

[9] 1998 world cup web site access logs. ftp://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[10] W. Timothy, S. Prashant, and Y. Mazin. Black-box and gray-box strategies for virtual machine migration. In *Proc. of NSDI*, 2007.

[11] A. Singh, M. Korupolu, and D. Mohapatra. Server-storage virtualization: Integration and load balancing in data centers. In *Proc. of SC*, 2008.

[12] Z. Xiao, W. Song, and Q. Chen. Dynamic resource allocation using virtual machines for cloud computing environment. *Trans. on TPDS*, 2013.

[13] D. Alsadie, Z. Tari, E. J. Alzahrani, and A. Y. Zomaya. Life: A predictive approach for vm placement in cloud environments. In *Proc. of NCA*, 2017.

[14] U. Sharma, P. Shenoy, S. Sahu, and A. Shaikh. A cost-aware elasticity provisioning system for the cloud. In *Proc. of ICDCS*, 2011.

[15] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Proc. of ISINM*, 2007.

[16] A. Beloglazov and R. Buyya. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. *Trans. on TPDS*, 2012.

[17] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proc. of SOSP*, 2017.

[18] T. Guo, Z. Xu, X. Yao, H. Chen, K. Aberer, and K. Funaya. Robust online time series prediction with recurrent neural networks. In *Proc. of DSAA*, 2016.

[19] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang. Deep learning with long short-term memory for time series prediction. *IEEE Communications Magazine*, 2019.

[20] L. Chen and H. Shen. Towards resource-efficient cloud systems: Avoiding over-provisioning in demand-prediction based resource provisioning. In *Proc. of BigData*, 2016.

[21] Z. Chen, J. Hu, G. Min, A. Y. Zomaya, and T. El-Ghazawi. Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning. *Trans on TPDS*, 2020.

[22] J. Kumar, R. Goomer, and A. Singh. Long short term memory recurrent neural network (lstm-rnn) based workload forecasting model for cloud datacenters. *Procedia Computer Science*, 2018.

[23] D. Janardhanan and E. Barrett. Cpu workload forecasting of machines in data centers using lstm recurrent neural networks and arima models. In *Proc. of ICIST*, 2017.

[24] W. Ding, F. Luo, C. Gu, H. Lu, and Q. Zhou. Performance-to-power ratio aware resource consolidation framework based on reinforcement learning in cloud data centers. *IEEE Access*, 2020.

[25] B. Waschneck, A. Reichstaller, L. Belzner, T. Altenmüller, T. Bauernhansl, A. Knapp, and A. Kyek. Optimization of global production scheduling with deep reinforcement learning. *Procedia CIRP*, 2018.

[26] X. Zhou, K. Wang, W. Jia, and M. Guo. Reinforcement learning-based adaptive resource management of differentiated services in geo-distributed data centers. In *Proc. of IWQoS*, 2017.

[27] S. Telenyk, E. Zharikov, and O. Rolik. Modeling of the data center resource management using reinforcement learning. In *Proc. of PICST*, 2018.

[28] H. Shen and L. Chen. Distributed autonomous virtual resource management in datacenters using finite-markov decision process. *Trans. on TON*, 2017.

[29] Google cluster data. https://code.google.com/p/googleclusterdata/.

[30] K. Qazi, Y. Li, and A. Sohn. Workload prediction of virtual machines for harnessing data center resources. In *Proc. of ICCC*, 2014.

[31] H. Van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

[32] Kvm:full virtualization solution for linux. https://www.linux-kvm.org/.

[33] Qemu: Qemu is a generic and open source machine emulator and virtualizer. https://www.qemu.org/.

[34] libvirt:is a toolkit to manage virtualization platforms. https://www.qemu.org/.

[35] stress-tool to impose load on and stress test systems. https://linux.die.net/man/1/stress.