Machine Learning for the Load Balance in Cloud Computing

Zhiming Fan
zf7ja@virginia.edu

Wanyu Du
wd5jq@virginia.edu

Xingchen Liu
xl4hk@virginia.edu

Abstract

In the cloud data center, the workload of different PM varies with time and it may lead to SLO violation and degrades the VMs performance. In order to achieve long-term SLO, load balance is used to dynamically migrate VM from overload PM to PM with low workload. In this project, we propose different load balance methods and evaluate their performance on several metrics. We use the Fast Fourier Transform (FFT) method, improved FFT method and LSTM model to predict the workload of VM and PM in the future time. Then we apply reinforcement learning and heuristic domain-knowledge based method to implement the migration matching for PM and VM. Based on the google cluster trace dataset, we compared four methods including prediction with Domain Knowledge matching, non prediction with Domain Knowledge matching, prediction with Reinforcement learning and non prediction with Reinforcement learning. We evaluate the performance on the SLO violation rate, migration number and overhead of algorithms. The result shows that using prediction method could achieve obvious performance improvement compared with non prediction method both on SLO violation rate and immigration numbers. For the migration matching algorithms, reinforcement learning could achieve a bit improvement on the SLO violation rate compared with the domain-knowledge based algorithm. However, for the migration number and overhead metrics, the reinforcement learning represents worse performance than domain-knowledge based method.

1 Introduction

Cloud Computing is a hot topic for IT service and the concept of "pay as you go" makes it more convenient for customers to connect to the IT service flexibly and cost-efficiently. Generally, for the cloud computing providers, the hardware virtualization is implemented, which means the Physical Machines (PM) in the data center always run several Virtual Machines (VM) with different resource allocation and configurations for various demand of customers. Since the load of each VM on a PM varies over time, the resource usage in a PM varies with different load states, which will lead to the load imbalance between different PMs in the datacenter. For example, one PM may be lightly loaded, while another PM may be overloaded, which will affect the performance of VM and violate the Service Level Agreement (SLA) between customers and service providers. Thus, in order to guarantee the SLA, resource management strategy is essential for cloud infrastructure to solve the problem of resource imbalance by migrating the VMs on the overloaded PM to release the excess load. If we successfully manage the resource, it will not only have more resource utilization, but also get energy efficiency and long term SLA. These will eventually improve the incomes of cloud providers.

In the report, Proposed Methods part records all the methods we use in the project. 3.1 is our data preprocessing part, We use cpu usage and memory usage data from Google Cluster Trace as out dataset. 3.2 is the workload prediction part. Based on the preprocessed data, we use three algorithms(FFT, improved FFT and LSTM) to do the workload prediction that predicts the future work loads in the cloud based on previous VM workload data. Then, based on the prediction results, we design two methods to do the migration matching, which migrates the VMs when a PM is overloaded. 3.3 introduces a domain-knowledge based migration method and 3.4 introduces a reinforcement learning based migration method. In the experiment part, we try all the previous methods and do the experiment on cpu usage. Evaluation metrics part explains the three metrics we use to evaluate our experiment and then we show our results in the result part.

The results show that by doing workload prediction, SLO violation rate and migration number reduces, which improves the workload balance performance. Therefore, prediction is a good method to deal with workload imbalance problem. For the two migration matching method, reinforcement learning does not show an obvious advantage compared with the domain knowledge based matching algorithm.

2 Related Work

For this project, we aim to do some research on the resource management strategy in the datacenter for the problem of load imbalance. Many previous researches have been done in this field. Many load balancing algorithms have been proposed to reactively perform the VM migration after the occurrence of load imbalance. For example, the method Sandpiper carries out dynamic monitoring and hotspot probing on PMs. It defines a volume-to-size ratio (VSR), which is used to select migration VMs. It prefers to migrate the VM with maximum VSR to PM. VectorDot is a method that migrates VMs from an overloaded PM to a PM that has the lowest vector product of node path vector and item path vector. The drawbacks of these algorithms are that they only consider the current state of the system. Besides, fixing the load imbalance problem upon its occurrence not only generates high delay for load balance but also can not guarantee the subsequent long-term load balance state.

Recently, some algorithms turn to predict VM resource demand and perform VM migration in a proactive way. In proactive load balancing algorithms, a PM could know whether it will be overloaded in the short future according to the prediction results of VM resource demand in the PM. People have tried various algorithms to build this demand prediction model: Sharma et al. [4] employ linear programming; Bobroff et al. [5] use a sliding window of data to make dynamic prediction; Beloglazov et al. [6] introduce a Markov model. However, these algorithms still cannot maintain a long-term load balance state. Besides, it is difficult for PM to decide which VM to migrate and it will generally introduce additional high delay and overhead for the VM migrations. To alleviate this problem, Shen and Chen [1] propose to apply MDP algorithm to teach PM how to select the optimal action for maintaining long-term load balance. This work sheds light on our project, and we hope to apply more advanced learning algorithms (i.e. the reinforcement learning algorithm) which can reduce the overhead and delay for proactive load balance algorithms, and also achieve long-term SLA.

3 Methods

In this project, we would like to apply some machine learning algorithms to implement the proactive resource management algorithms for load balance of the data center. We implement three approaches for the workload prediction part. The first approach is FFT, the second one is improved FFT method. The last approach is using the prediction methods based on deep neural network LSTM to predict the workload of PM and VM. For the time-series data of PM and VM, the deep neural network could achieve high performance for prediction. Then according to the prediction workload of PM and VM, we implement domain knowledge based alogrithm for the VM migration between PMs to minimize the SLO violations and achieve load balance. For another approach, we use reinforcement learning method (policy gradient method) to implement the VM migrations between PMs. For the reinforcement learning , we define different states, actions, rewards for the PM and VM in the whole system, then establish models for the agents to implement load balance and VM migrations. We can define {high, medium, low} workload level for different resources types in PM and VM to describe the status. Then, the action is defined as different workload level state of VM migrations from one PM to another PM. We give different rewards such as positive or negative reward for different actions and state transitions. By interactive with the environments, the optimal policies could be achieved. For this method, the policy is used for proactively migrating the VM to reduce the excess load for some overloaded PMs. By dynamically learning and modify the optimal policies, the long-term SLA is achieved.

3.1 Data Preprocessing

In the project, we use Google Cluster trace as our trace and extract data from it. The Google Cluster trace records resource usage on a cluster of about 11000 machines from May 2011 for 29 days. As there are a large number of data with various resource types, we focus on the CPU resource and memory resource in the experiments. We extract nearly 1000 different task traces from the google cluster trace and we record the CPU usage and memory usage workload for at least 1000 timestamp. We use these data to form the dataset for the simulation in the experiment and we regarded the workload trace of each task as a VM workload trace in the following experiment.

3.2 Workload Prediction

After getting the processed VM workload data, we build three workload prediction models in order to use the previous VM workload data to predict the future workloads in the cloud.

Fast Fourier Transform (FFT) Prediction Model: We follow the previous work [8] to implement the FFT method for predicting the workload usage in the cloud computing environment. The FFT is used to decompose history workload usage data into a number of different frequency components. Those frequency components represent for the repeated workload usage patterns in the history data. To predict the workload usage at future time t_{N+i}, FFT sum up the low-frequency components at time t_{N+i} and convert 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.

Algorithm 1 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 $k = 0 \to L - 1$ do:
    $f_k = \sum_{n=0}^{L-1} u_{t_{N-L+i+n}} \cdot e^{-j2\pi \frac{k}{L} n}$;
end for
for $k = 0 \to L/2$ do
    $\hat{p}_{t_{N+i}} + = \frac{1}{L} \sum_{k=0}^{L/2} f_k \cdot e^{j2\pi \frac{k}{L}(N+i)}$;
end for
return $\hat{p}_{t_{N+i}}$.
```

Improved FFT Prediction Model: Since the FFT method only uses the low-frequency components to make the prediction, it will miss out some useful high-frequency components which also affects the prediction accuracy. Therefore, we follow another previous work which takes the high-frequency components into account for workload prediction, and implement the improved FFT prediction model. The only difference between FFT and improved FFT is whether we compute 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.

LSTM Prediction Model: We also apply the time series analysis method to build the prediction model. We build a single 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 of the model to predict the workload at the next timestamp. For the LSTM model, the input time lag is 8 and we train our model on 739,050 samples and validation them on 82,117 samples. We further get the prediction of each PM workload by summing up the prediction workload of each VM in the PM based on the PM and VM matching record at each timestamp.

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 $k = 0 \to L - 1$ do:
    $f_k = \sum_{n=0}^{L-1} u_{t_{N-L+i+n}} \cdot e^{-j2\pi \frac{k}{L} n}$;
end for
for $k = 0 \to L/2$ do
    $\tilde{p}_{t_{N+i}} += \frac{1}{L} \sum_{k=0}^{L/2} f_k \cdot e^{j2\pi \frac{k}{L}(N+i)}$;
end for
Calculate the residual component $r_{t_{N+i}}$ based on AR(2);
return $\hat{p}_{t_{N+i}} = \tilde{p}_{t_{N+i}} + r_{t_{N+i}}$.
```

3.3 Domain-Knowledge Based Migration Matching

For the load balance procedure, the VM migration policy need to be implemented to avoid the overload of PM. At each timestamp, the current workload of each VM could be obtained from the trace dataset. The current workload of PM could be calculated according to the PM-VM matching list of the last timestamp. The migration matching was regarded as a PM-VM re-allocated procedure based on the workload information and update the PM-VM matching list. Firslty We implemented Domain-Knowledge based migration policy.

During the policy, each VM predicted the workload of the next timestamp using the pre-trained prediction model mentioned above. In this case, the workload of the PM could be obtained using the current PM-VM matching list. According to the prediction workload, we obtained the high resource utilization PM list for the next time. Then, we implemented the VM migration procedure for the PM in the list. During the VM migration procedure, each PM decided to migrate which VM and the corresponding PM destination. For each overloaded PM, we define a target workload after migration, such as 65% resource usage. In this way, the migrating workload could be calculated and we choose the migration VM according to the migrating workload. For the PM destination, we directly choose the PM with the least workload in the next time. The algorithm could be described as follows:

Algorithm 3 Domain Knowledge Based Migration Matching Algorithm

```
for each Timestamp t do:
    $PM\_workload_{t+1} = Predict(PM\_workload_{t,t-1,...})$
    $OverloadPM\_list \leftarrow Select(PM\_workload_{t+1})$
    for each PM in Overload PM_list do
        $Migration\_workload = PM\_workload[i]_{t+1} - target\_workload$
        choose VM_index_Migration← Migration_workload
        choose PM destination with low workload
        update PM_VM Matching List
    end for
end for
```

3.4 Reinforcement Learning Based Migration Matching

Based on the resource utilizaton and resouce capacities of VM and PM, we design a Reinforcement Learning-based model to address the resource allocation and VM migration in the data center. We regrads the VM scheduler as an RL agent and the VM migration as an action. At each timestamp, for earch PM, the scheduler choose an action to interact with the environment. The overall procedure for Reinforcement learning is shown in the figures as follows.

For reinforcement learning, the complete environment is described by several parameters such as states, action, rewards and current time, which is described as a tuple (S,A,R,T). At each timestamp, the agent choose the action based on the current environment state. Then the environment update the state and feedback the corresponding reward according to the input of action. For our work, we firstly

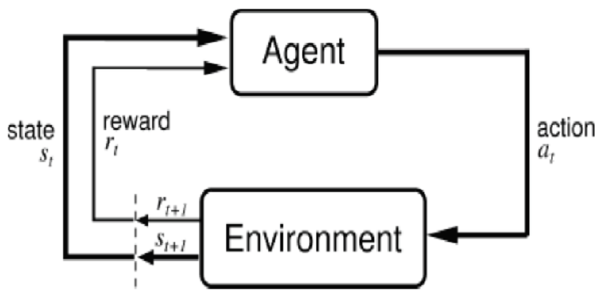


Figure 1: Reinforcement Learning Environment Interaction

classify the resource utilization as three level {High,Medium,Low}. We defined two threshold T_1 and T_2 to distinguish the different levels. In this way, the workload of PM and VM are allocated to three levels. The reinforcement learning environment tuple <S,A,T,R> for our work is described in the following:

States (S) Actions(A): A set of PM states are used to represent to the current resource utilization of all PMs in the datacenter. Assume PM_Num represent the total number of PMs in the datacenter. Thus, the dimension of state set is PM_Num. Since each PM has three possible workload level, the total states number for the state set is $3 * PM\_Num$. For the action space, each PM needs to decide to choose which VM level to migration or no action. Besides, agent also needs to decide which PM to accept the migrated VM. Thus, the total number of the action space is $3 * (PM\_Num - 1) + 1$.

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

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

Agent Policy Network: The agent picks actions based on a policy. We define a probability distribution over action $\pi(s, a)$ to represent the probability that action a is taken in state s. Since there many possible {state, action} pairs, we use deep neural network to represent the policy in our design. We denote θ as the parameters for training in the network and the policy is represented as $\pi_\theta(s, a)$. The figure1 shows the RL agent. At each timestamp, the current PM states is fed to the policy network to get the probability of different actions. The agent chooses the action to output based on the action probability distribution.

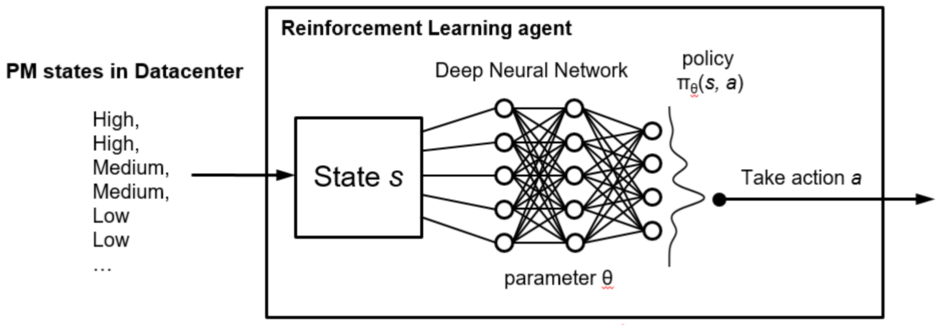


Figure 2: Reinforcement Learning Agent

Policy gradients methods: We use policy gradient methods 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:

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

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. In our design, ν_t is represented by the reward feedback.

According to the environment space and policy model described, the RL-based VM migration algorithm is given.

Algorithm 4 Reinforcement Learning based Migration Matching Algorithm

```
for each iteration do:
    $agent\_train\_data \leftarrow 0$
    for each PM in PM_Nums do
        $Network\_State = Cyclic(State)$
        $Action, Prob = agent.act(Network\_State)$
        $State, reward = environment.step(Action)$
        $agent.train\_data = agent.remember(State, action, prob, reward)$
    end for
    $agent.train\_batch\_size(agent.train\_data)$
end for
```

4 Experiments

In this Section, we conducted trace-driven experiments to evaluate the performance of the proposed methods for load balance.

4.1 Capturing Multi-Dimensional Loads

Since a PM or VM has multiple resource type such as CPU, memory, network bandwidth etc, the workload status for PM and VM should consider each type of the resource utilization. Each VM or PM can be overloaded along one or more of different resource dimensions, we define a new metric volume-to-size ratio (VSR) that captures the combined resource type load of a virtual and physical sever.Each PM and VM has a VSR, where the size is its memory footprint and the volume is the product of different types of resource utilization. Let $U_i (i = 1, 2, \cdots, k)$ denote the resource utilization for different resource type,the volume could be represented as:

$$volume = \frac{1}{(1 - U_1) * (1 - U_2) * \cdots * (1 - U_k)}$$

In this way, the higher the utilization of a resource, the greater the volume; if multiple resources are heavily utilized, the above product results in a correspondingly higher volume. The volume captures the degree of overload along multiple dimensions in a unied fashion and can be used by the mitigation algorithms to handle all resource hotspots in an identical manner. In this experiment, we mainly use the resource type of CPU and memory usage. For the high, medium, low workload states for PM and VM during the migration policy, we use the load states of VSR for each PM and VM to describe the workload states.

4.2 Experiment setting up-heavy Workload (100 PM hosting 1000 VM)

We simulated the cloud data center with 100 PMs hosting 1000 VMs to evaluate the performance of different algorithnms with heavy workload (per PM hosting average 10 Vms). The VMs are randomly chosen from the task traces of google cluster datasets. The Google Cluster trace records the resource usage (CPU, memory, etc) for each task every 5 minutes and the resource utilization trace from Google Cluster VMs are used to dirve the VM resource utilizations in the simulation. At the beginning of the experiments, the VMs are randomly allocated to the PMs which will cause the load imbalances between different PMs. Then we used the different load balanced methods for PM-VMs mapping and VM migrations in each experiment to evaluate the performance of different algorithms.

When the simulation started, at each timestamp with 5 minutes, the resource utilization of all PMs and VMs are updated according to the data of Google Cluster trace and the current matching status of PM-VMs. We recorded the number of overloaded PMs (the occurrence of SLO violations for PM). We define the SLO violations threshold is 0.95. If the resource utilization of PM exceed the threshold, it would cause the SLO violations. Each PM conducted the load balance algorithms for VM migrations at each timestamp and the number of VM migrations were recorded.In each experiment round, we simulated 500 timestamps (nearly 2500 minutes) and get the cumulative numbers of overload PMs and VM migrations to represent the performance of different algorithms. We repeatedly implement the experiments for 10 times to calculate the average performance.

4.3 Experiment setting up-light Workload (1000 PM hosting 2000 VM)

In the light-workload setup, we set the number of PMs in the cloud to 1000, with capacities of 1.5GHz GPU and 512 MB memory. The threshold of resource utilization to check if a PM is overloaded was set to 0.9. The VM number in the experiment is 2000 (per PM hosting average 2 VM). We assume that the historical VM demand records are available at the beginning of simulation. At the beginning of the simulation, we randomly allocated 2000 VMs to the 1000 PMs, which made a portion of the PMs overloaded.

5 Evaluation Metrics

For the evaluations of the experiment results, we use three metrics. The first metric is the SLO violation rate. The SLO violation rate is determined by the percentage of time during which an active PM experience CPU utilization above 95% and leads to performance degradation. The calculation of SLO violation rate is defined as:

$$SLO\_Violation\_Rate = \frac{\sum_{t=1}^{T} PM\_Overloaded(t)}{T * PM\_num}$$

The second metric is the migration number during the time period of simulations. Since the VM migrations may cause the latency and network bandwidth cost, the load balance algorithms should minize the VM migration numbers under certain SLO condition. The VM migration number is defined as:

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

The third metric is the overhead for different algorithms. The overhead is defined as the execution time for each algorithm. In the experiment, we record the execution time for each timestamp and calculate the average execution time based on the a period of time records as the overhead of different algorithm. The overhead is defined as:

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

6 Results

In our experiment, we compare the evaluation metrics of different algorithms to represent the performance. There are five algorithms in our experiment. They are respectively prediction workload with Domain Knowledge (DK) based Matching (Predict-DK), non prediction workload with DK based matching (Active-DK), prediction workload with RL matching (Predict-RL), non prediction workload with RL matching (Active-RL) and none load balance (None LB) .

6.1 Prediction Accuracy

Based on the prediction models in section 3.2, we predict the workload trace of each VM in the dataset and calculate the accuracy of the workload. In order to represent the distribution of the prediction results, we use the statistical cdf (cumulative distributive function) curve of the accuracy shown in figure 3. We compare the CPU usage workload prediction accuracy between the FFT, improved FFT and LSTM model, and the result shows that the LSTM obtains the highest prediction

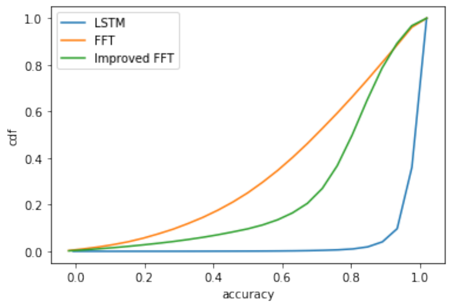


Figure 3: Prediction accuracy for three models

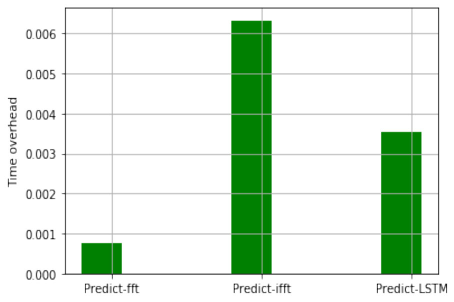
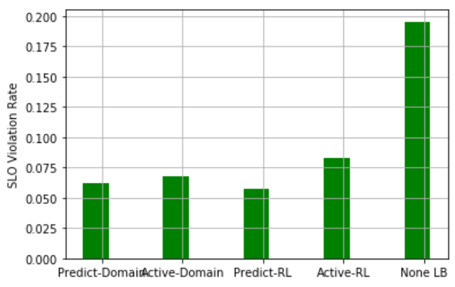


Figure 4: Prediction overhead for three models

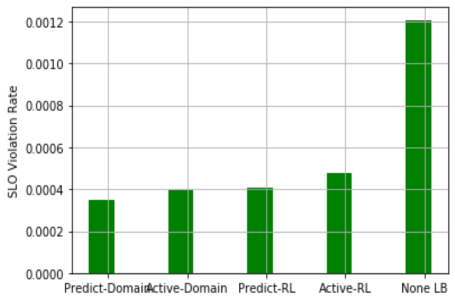
accuracy, where the majority of its prediction could achieve around 90% accuracy. Besides, we also compute the online prediction time (i.e. prediction overhead) for each of the prediction model, and the results are shown in figure 4. Since LSTM has a moderate overhead and a high accuracy, we decide to choose LSTM as our workload prediction model, and generate workload predictions for the following migration matching algorithm.

6.2 SLO Violation Rate

The SLO violation rate for different load balance algorithms is shown in the figure 5. It shows that the workload prediction could always improve the performance compared with non workload prediction. For different matching algorithms, the performance of reinforcement learning is almost the same as the domain-knowledge based migration matching algorithm. In some cases, RL could perform a little better than domain-knowledge based algorithm.



(a).SLO violation rate(100PM,1000VM)



(b). SLO violation rate(1000PM,2000VM)

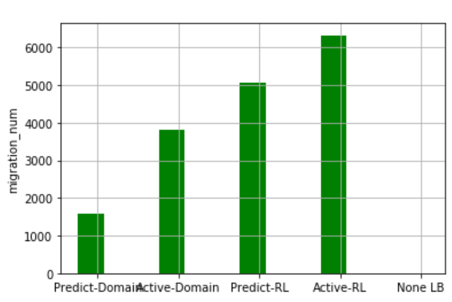
Figure 5: SLO Violation Rate for Different Load Balance Algorithms

6.3 Migration Number

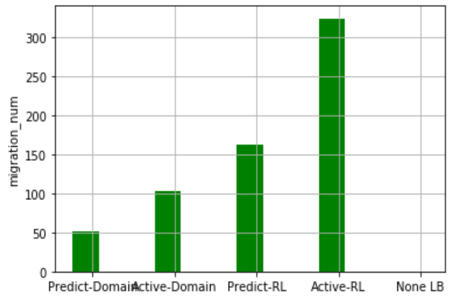
Figure6 shows the migration number for different load balance algorithms. It demonstrates that using workload prediction could obviously decrease the migration number for load balance, which could save the bandwidth cost and reduce the migration latency in the real data center. Besides, the reinforcement learning matching algorithm performed much more VM migrations compared with the domain knowledge based matching algorithm.

6.4 Overhead

The overhead represents the execution time for different algorithms. As the figure 7 shows, compared with the non workload prediction, the workload prediction increased a little overhead. However, for different matching algorithm, the overhead for reinforcement learning is much higher that domain knowledge based matching algorithm. Firstly, the execution for reinforcement learning such as action chosen from the policy network, state transition introduces the time overhead. Besides, the higher migration numbers for reinforcement learning will increase the execution time obviously.

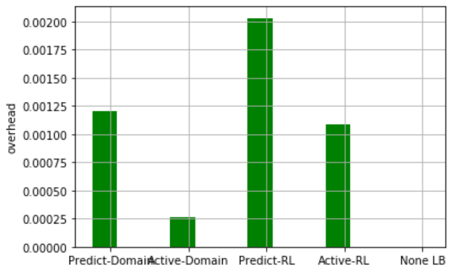


(a).Migration$_num(100PM, 1000VM)$

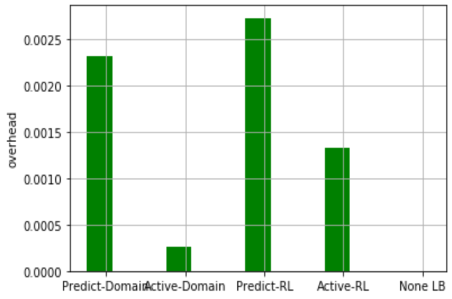


(b). Migration$_num(1000PM, 2000VM)$

Figure 6: Migration Number for Different Load Balance Algorithms



(a).Overhead(100PM,1000VM)



(b). Overhead(1000PM,2000VM)

Figure 7: Overhead for Different Load Balance Algorithms

7 Summary

In this project, we implement different load balance algorithms and evaluate their performance in different evaluation metrics. From the comparison results, using workload prediction could improve the load balance performance since it could decrease the SLo violation rate and the migration numbers with the prediction information. For reinforcement learning, it had not represented an obvious advantage compared with the domain knowledge based matching algorithm. For the future work, we would further optimize the reinforcement learning algorithms such as the reward function optimization, more feature engineering for the state representation and more effective learning approach etc.

References

[1] H Shen, L Chen, Distributed autonomous virtual resource management in data centers using finite Markov Decision Process, IEEE/ACM Transactions on Networking 2017.

[2] H. Mao, M. Schwarzkopf, Learning Scheduling Algorithms for Data Processing Clusters. SIGCOMM 2019

[3] H. Mao, M. Alizadeh, Resource Management for Deep Reinforcement Learning. HotNets 2016

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

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

[6] A. Beloglazov and R. Buyya. Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints. TPDS, 24(7): 1366–1379, 2013.

[7] Shen, H., & Chen, L. (2017). Distributed autonomous virtual resource management in datacenters using finite-markov decision process. IEEE/ACM Transactions on Networking, 25(6), 3836-3849.

[8] Chen, Liuhua, and Haiying Shen. "Towards resource-efficient cloud systems: Avoiding over-provisioning in demand-prediction based resource provisioning." 2016 IEEE International Conference on Big Data (Big Data). IEEE, 2016.