기계학습 기반 가상머신 I/O 스레드 CPU 요구량 예측 기법*

이지유○, 이관훈, 유 혁†
고려대학교 정보대학 컴퓨터학과
eguku@korea.ac.kr, {khlee, chuckyoo}@os.korea.ac.kr

A Machine-Learning-Based Approach towards Modeling the CPU Quota Requirements of Virtual Machine I/O Threads

Ji You Lee ○, Kwanhoon Lee, Chuck Yoo
Department of Computer Science and Engineering, Korea University

Abstract

With cloud computing gaining popularity, providing reliable network performance within virtualized environments has become a major issue. Previous research states that the resource allocation of I/O threads (i.e., kernel threads that manage the I/O operations of a virtual machine) plays an important role in determining network performance. Therefore, in this paper, we aim to predict the amount of CPU bandwidth I/O threads utilize in various different networking situations by adopting a machine learning approach. Specifically, we experiment with building efficient machine learning models that estimate the CPU quota requirements of the I/O threads of virtual machines with specific network performance needs.

1. INTRODUCTION

With cloud computing gaining popularity, providing reliable network performance within virtualized environments has become a major issue. For example, many studies regarding network performance isolation for virtual machines have been conducted [1, 2, 3, 4]. Their solutions consist primarily of either the reallocation of virtual CPU cores [2, 3] or the redistribution of the network bandwidth [1, 4].

However, previous research [5] states that the resource allocation of I/O threads (i.e., kernel threads that manage the I/O operations of a virtual machine) plays an important role in determining network performance. Since the I/O threads form producer-consumer relationships, I/O requests are completed at the expense of high synchronization costs when the processing speeds of the threads become unbalanced. Thus, misallocation of system resources such as the CPU bandwidth to I/O threads can cause critical problems in providing network performance isolation, regardless of the distribution methodologies about the virtual CPU cores and the network bandwidth.

Therefore, in this paper, we aim to predict the amount of CPU bandwidth I/O threads utilize in various different networking situations by adopting a machine learning approach. Specifically, we experiment with building efficient machine learning models that estimate the CPU quota requirements of the I/O threads of virtual machines with specific network performance needs.

The rest of this paper is composed as follows. In Section 2, we explain the concept of *vhost* in relation to VirtIO and the notion of the CPU quota in the context of the CFS bandwidth control mechanism so as to provide a better understanding of the problem of our interest. In Section 3, we describe our machine learning experiments, from generating and preprocessing data to training and validating models. In Section 4, we evaluate the performance of our models. Finally, in Section 5, we report our conclusions and plans for future research.

2. BACKGROUND

2.1 VirtIO and Vhost

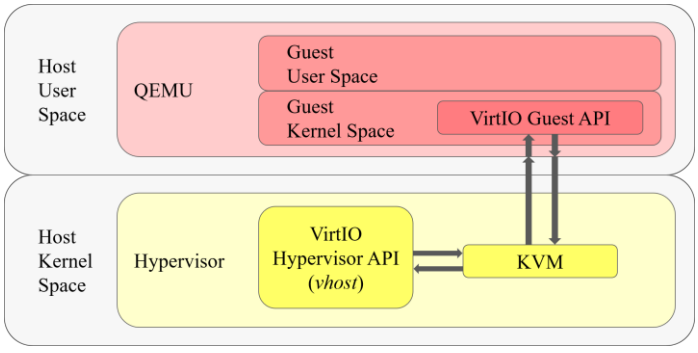


Figure 1. VirtIO architecture

VirtIO [6] is an abstraction technique utilized by the Linux kernel that provides a unified means of efficiently managing the different virtual device drivers of various hypervisor systems such as Xen, KVM, and

* This work was supported by the Institute of Information & Communications Technology planning & Evaluation (IITP) grant, funded by the Government of Korea (MSIT) (No. 2015-0-00280, (SW Starlab) Next Generation Cloud Infra-software Toward the Guarantee of Performance and Security SLA), and the Next Generation Engineering Researcher Program of the National Research Foundation of Korea (NRF), funded by the Ministry of Science, ICT (No. NRF-2019H1D8A2105513).

† Corresponding author.

VirtualBox, using a shim layer. As can be seen in Figure 1, the *VirtIO* API consists of two parts: the hypervisor-side API, for use by hypervisors, and the guest-side API, for use by guests. In the KVM hypervisor, the hypervisor-side API is implemented as the *vhost-net* driver. The driver runs in a dedicated kernel thread called *vhost*, which emulates the network requests from the guest-side API and delivers them to their corresponding hypervisor network interfaces. *Vhost* communicates with the guest-side API through interactions with the KVM kernel module, using a ring buffer data structure.

2.2 CFS Bandwidth Control

The CFS scheduler, the default CPU scheduler of Linux, controls the maximum CPU bandwidth of process groups by using two parameters: the quota and the period [7]. For each new period, a group is assigned as its bandwidth the amount of time specified in its quota parameter. Once its bandwidth has been fully consumed, the group is throttled, meaning corresponding threads are prevented from running again until the next period.

Users can manually set the maximum CPU bandwidth of a group by specifying the values of *cpu.cfs_period_us* and *cpu.cfs_quota_us* of the Linux *cpu* subsystem. For each group, default values are given as 100,000 and -1 respectively, indicating that the group has no quota restrictions within each period of 100ms. Writing any valid positive integer to *cpu.cfs_quota_us* automatically sets the maximum bandwidth limit. Both variables have a lower bound of 1ms; the upper bound of *cpu.cfs_period_us* is 1s, and that of *cpu.cfs_quota_us* is the corresponding value of *cpu.cfs_period_us*.

3. EXPERIMENT

3.1 Experiment Setup

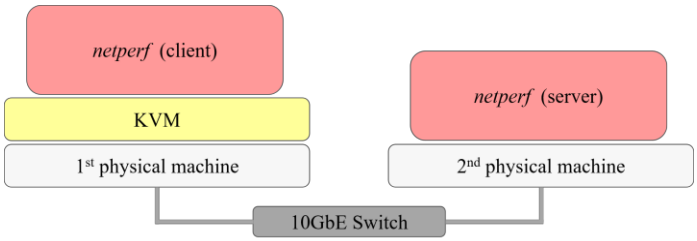


Figure 2. A model of our experiment environment

To create the dataset for our machine learning experiments, we utilize two 10-core physical machines and the benchmark tool *netperf* [8]. On the first physical machine, we create a single Linux virtual machine (VM) and allocate it one virtual CPU core. Then, we connect it to the second physical machine via a 10GbE switch. For our experiment parameters, we use the size of the TCP packet transmitted via *netperf* (64, 128, …, 1024B) and the amount of the CPU quota allocated to *vhost* (10000, 20000, 30000, …, 100000μs). Also, we use *scikit-learn* [9], an open-source Python library equipped with state-of-the-art machine learning tools, for data processing and model development.

3.2 Dataset Generation

For each parameter setting, we execute *netperf* such that the Linux virtual machine (i.e., the *netperf* client) transmits TCP packets continuously for one minute to the second physical machine (i.e., the *netperf* server). While the packet transmissions take place, we run *vnstat* [10] and *pidstat* [11] simultaneously on the first physical machine. *Vnstat* captures the network throughput data of the Linux VM while *pidstat* measures the CPU usage of the Linux VM. As a result of our experiments, 500 samples are generated.

3.3 Feature Selection

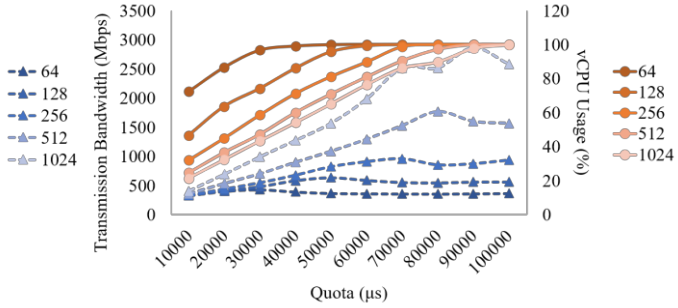


Figure 3. The average transmission bandwidth and vCPU usage per allotted CPU quota and transmission packet size

From each of the samples, we use the transmission bandwidth, the number of transmitted packets per second, the virtual CPU core usage rate and the size of the transmitted TCP packet as our machine learning features. The amount of the CPU quota allocated to *vhost* we utilize as our target value.

3.4 Data Preprocessing

In preparation for the model building phase, we preprocess our raw data as follows. First, because *vnstat* uses both Gbps and Mbps to report the network bandwidth, we scale all CPU bandwidth data in units of Gbps to units of Mbps. Then, we normalize the data using the min-max normalization method such that all values are between 0 and 1 to unify the range of each feature so that the model prediction accuracy is increased. Finally, we randomly split the data by following a 4:1 ratio, forming a training set of 400 samples and a test set of 100 samples.

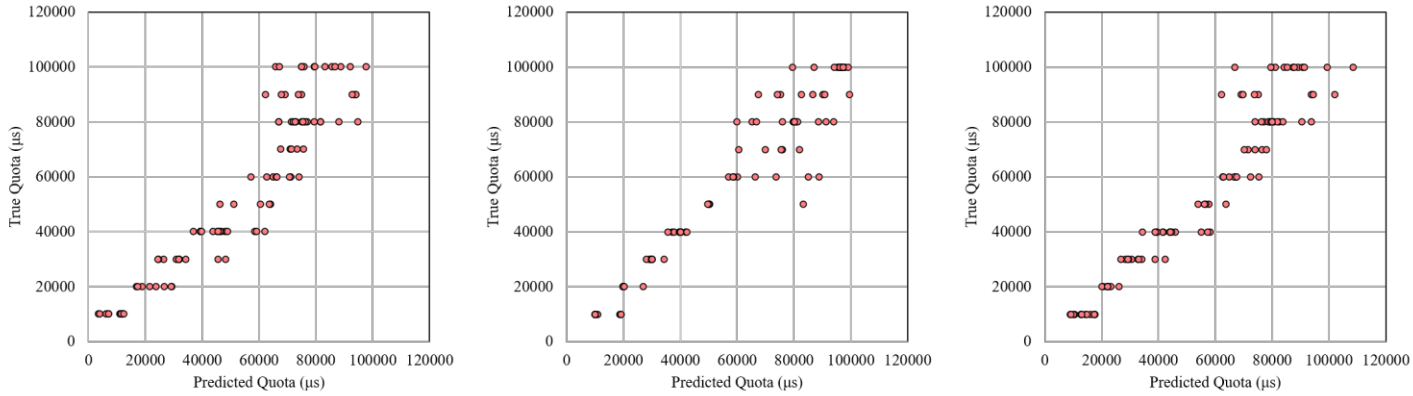
3.5 Model Building

We conduct model training using the following three popular supervised machine learning regression algorithms:

Linear regression A model composed of a set of predictive variables and an outcome variable such that they form a linear relationship.

Random forest regression A model [12] composed of a group of numerical tree predictors, each grown using the given training set and a random vector, which is independently generated from past vectors but shares the same distribution. The output is formed by taking the average of the predictions of all the trees.

Support vector regression (with a polynomial kernel) A model [13] that constructs a set of hyperplanes in a high-dimensional space out of which the plane that is the farthest in distance from the nearest training



(a) Linear Regression (b) Random Forest Regression (c) Support Vector Regression

Figure 4. Relationships between the predicted and true CPU quota

data point of each class is used to generate predictions.

We use the random grid search technique for hyperparameter tuning. For each set of hyperparameters, we perform 5-fold cross validation on the training set. This means we separate the training set into five smaller sets of 80 samples each, generate a prediction error on each set after fitting the model to the other four sets, then calculate the validation estimate as the average of the prediction errors.

4. EVALUATION

In analyzing model performance, we use the root mean square log error (RMSLE) as our evaluation metric for its robustness against outliers and lighter penalization of big differences between large-scaled test and predicted values. Out of the three models, random forest regression generates the best predictions (0.1587 RMSLE). Support vector regression performs second best (0.1901 RMSLE), followed by linear regression (0.2436 RMSLE).

Figure 4 depicts the relationships between the CPU quota values of the test set and the CPU quota values predicted by each of the models. Ideal model performance should result in a clear linear relationship between the two factors. We observe that Figure 4-(b) displays the greatest linearity.

5. SUMMARY AND FUTURE WORKS

We have experimented with building an efficient machine learning model that predicts the CPU quota requirements of *vhost* to properly ensure the performance needs of its virtual machine. By doing so, we have found that, out of the algorithms tested, random forest regression best models our situation.

We believe our work lays the foundation for incorporating machine learning techniques where controlling the system resources allocated to I/O threads is concerned in managing the network performance of virtual machines. Therefore, with the results obtained from our experiments, we plan to design a scheduler that provides reliable network performance isolation in virtual machines by adjusting the amount of CPU quota allocated to its I/O threads using machine learning algorithms.

REFERENCES

[1] C. Hong, K. Lee, H. Park, C. Yoo, "ANCS: Achieving QoS through Dynamic Allocation of Network Resources in Virtualized Clouds," *Scientific Programming*, 2016.

[2] C. Xu, S. Gamage, H. Lu, R. Kompella, and D. Xu, "vTurbo: Accelerating Virtual Machine I/O Processing Using Designated Turbo-Sliced Core," in *Proceedings of the USENIX Annual Technical Conference*, pp.243-254, 2013.

[3] J. Ahn, C. H. Park, T. Heo, J. Huh, "Accelerating critical OS services in virtualized systems with flexible micro-sliced cores," in *Proceedings of the 13th EuroSys Conference*, pp. 1-14, 2018.

[4] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and A. Greenberg, "EyeQ: Practical Network Performance Isolation at the Edge," in *Proceedings of the 10th USENIX Conference of Network System Design and* Implementation, pp. 297-312, 2013.

[5] G. Lettieri, V. Maffione, and L. Rizzo, "A Study of I/O Performance of Virtual Machines," in *The Computer Journal*, 61, pp. 808-831, 2017.

[6] R. Russell, "virtio: Towards a De-Facto Standard For Virtual I/O Devices," in *ACM SIGOPS Operating Systems Review*, 42, pp. 95-103, 2008.

[7] "CFS Bandwidth Control" [Online]. Available: https://www.kernel.org/doc/Documentation/scheduler/sched-bwc.txt. [Accessed: 13-October-2020].

[8] "netperf" [Online]. Available: https://github.com/HewlettPackard/netperf. [Accessed: 22-August-2020].

[9] Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," in *JMLR*, 12, pp. 2825-2830, 2011.

[10] "vnstat" [Online]. Available: https://linux.die.net/man/1/vnstat. [Accessed: 22-August-2020].

[11] "pidstat" [Online]. Available: https://man7.org/linux/man-pages/man1/pidstat.1.html. [Accessed: 22-August-2020].

[12] L. Breiman, "Random Forests," *Machine Learning*, 45, pp. 5-32, 2001.

[13] A. J. Smola, B. Scholkopf, "A tutorial on support vector regression," in *Statistics and Computing*, 14, pp. 199-222, 2004.