A Memory Resource Prediction Method Based on Machine Learning

Gaoxiang Zhang
Institute of Network Computing and Information Systems
School of Electronics Engineering and Computer Science, Peking University
Beijing, China
1100012899@pku.edu.cn

***Abstract*—The para-virtualization architecture of Xen results in the incapability of acquiring memory demand. The method equivalent to software simulation is widely used but gains high overhead. In this paper, we propose a low cost method which uses machine learning to build the memory resource prediction model via getting performance counter. This method can integrate with other optimizing approach and reduce the overhead.**

Keywords—Virtualization environment; machine learning; memory resource prediction; working set

I. Introduction

Virtualization enables multiple operating systems to run on their own *virtual machine* (VM) separately and efficiently for users. One of the challenges is effective management of shared physical resources, which has a premise of the precise and low-cost measurement of resources.

As the representation of para-virtualization, Xen modifies the kernel of operating system and realizes a light software layer namely *virtual machine monitor* (VMM). Owing to no host OS participation in resource management, VMs are almost directly upon hardware like real systems, leading to high performance. However such fidelity causes the VMM is unaware of resource demand of VMs and thus unable to conduct management.

To be informed of the usage, the wide-spread method [1] is to simulate the page scheduling process in real time. However, this method results in high overhead. Our work is motivated by the algorithms in machine learning. We hope to use machine learning to predict memory resource precisely and efficiently.

II. Related Work

A. Working Set

The concept of *working set* [2] is the set of memory pages referenced by a process during a time interval. *Working set size* (WSS) is the amount of memory that a process (or a VM) needs without paging, so it is a good representative of memory resource usage.

B. Miss Ratio Curve Based Working Set Size Estimation

The miss ratio curve (MRC) based WSS estimation is a widely used technique [5]. MRC reflect the correlation of memory size and page miss ratio. With a MRC, we can redefine WSS as the memory size with a predefined tolerable page miss rate. By tracking the physical memory address of accesses, we can build the LRU histogram and calculate MRC, then getting the WSS with a given threshold.

As we discussed before, it is inevitable to overcome the transparent accesses through VMM. The first method is hardware approach that use memory bus to get the address. [3] Though little overhead, it is unavailable on current processors. The second is *OS approach*, the OS revokes access permission for pages, which will cause a protection fault so that VMM is able to get the address. Due to frequent page faults and the cost of updating LRU, this method has high overhead, shown as (1):

$$\textit{Interrupt Times} * (T_{\textit{page fault disposal}} + T_{\textit{LRU update}}) \qquad (1)$$

To reduce the overhead, former research proposed AVL tree based technique to decrease LRU update time, and ‘hot-set’ to decrease interrupt times [4]. Our approach will not only decrease the interrupt times, but is also orthogonal to these techniques. By synthetizing optimizing means, we can greatly reduce the overhead of memory resource tracking.

C. Hardware Counter Characters

Zhao *et al.* [5] found that the variation trend of WSS matches some hardware performance counter and made use of this trend. Motivated by this discovery, we not only use the similarity, but also the value relationship. Thanks to low cost of getting hardware counter and conducting prediction, our target is to build the suitable model with available algorithm.

III. Inplementation

A. Feature and Algorithm Selection

Choosing appropriate features is critical to our approach. The architecture today supports hundreds of performance indexes, but it restrict the number of indexes gained at runtime, attributing to limited registers. So when implemented, we must find very features that can describe memory usage. After sampling some indexes that are related to cache and TLB, we find the trend of the access number of level 1 cache, the miss number of level 2 and 3 cache, as well as the miss number of TLB are more fitted to WSS, so in our implementation, we use them as our training features.

After testing various regression algorithms, we decide to choose Lasso Regression [6] as our trainer.

B. Limitations of Other Models

The basic idea is simply using system instructions such as ‘ps’ or ‘top’ to get the memory usage of process. Though the overhead is low, the result is quite inaccurate.

It is also natural to intend to find a single model that is trained on one program and tested on another. But actually, we note and the experiment shows that different program have a bunch of memory access pattern, so direct analysis cannot work. This restriction also blocks the classification of program behave. So it is unlikely to find a global model, which means it is essential to use runtime data to assist.

The former technology [5] called “Intermittent Memory Tracking” (IMT) which filters fluctuation and simply regards WSS as flat when TLB miss value stay in a relatively stable situation. This strategy works quite well in program with long and flat phases, but cannot gain precise value of wavy phases.

C. Self Adaptive Model

To overcome the limitations posed by former models, we refer to the idea of IMT so as to predict precisely and the result is compliant with original data.

To prevent excessive tracking, we need to collect partial data. One idea is to open tracking for a while and use this single model to predict the rest. It is simple but also not proper to inter-periodic prediction. Our idea is to intermittently open tracking and dynamically change the model, then we use the sub-model to predict the WSS in a sub region. Because of the stable in-phase character, this idea is more suitable.

So tracking interval is the key parameter. The first strategy sets a fixed interval. The second strategy uses dynamic interval. With an initial interval, when meeting a tracking point, the algorithm will test the gap between real data and predict data. If the gap is too large, the algorithm will decrease the interval in order to collect more neighbor data to correct the model. Or the gap is small enough, which means the model is stable, so the frequent tracking is not useful and the interval will increase.

After the prediction, we need to amend the data. Firstly, the negative data should go to zero. Secondly, we judge a ‘noise’ if its value is far larger or smaller than its neighbor, these noises should be eliminated and go to the mean value of its neighbor.

It is obvious that neighbor data may affect more on prediction, so we add the weight of closer data by simply setting a limited FIFO auxiliary set with latest data. When a training data arrives, it will be added to both training set and this auxiliary set, with the old data in auxiliary dequeuing (if needed). And the regression use both sets to train the model.

IV. Evaluation

A. Evaluation Criteria

We define the tracking rate and error rate as our evaluation criteria, showed in (2) and (3). Tracking rate reflects the overhead, and error rate represents the correctness. We aim to restrict the error rate to a low level, and try decreasing tracking rate as much as possible.

$$\textit{Tracking Rate} = \textit{Tracking Point} / \textit{Total Point} \tag{2}$$

$$\textit{Error Rate} = (\textit{true value} - \textit{predict value}) / \max(\textit{true} - \textit{predict}) \tag{3}$$

B. Experimental Evaluation

We use modified Xen 4.2 and Linux 3.10 as our experiment environment. And our benchmark is SPEC CPU 2006, which contains various programming language and memory access pattern.

TABLE I. EVALUATION OF SELF ADAPTIVE MODEL

Tracking Interval	**Evaluation Criteria of SPEC CPU 2006**	
	Average Tracking Rate	***Average Error Rate***
Fixed T=2	50%	11.39%
Fixed T=5	20%	17.00%
Fixed T=10	10%	19.95%
Dynamic T	7.1%	9.60%
RSS	N/A	34.2%

V. Conclusion

In this paper, we propose an approach using machine learning to reduce the frequency of original tracking and acquire relatively precise working set size value. It overcomes the defects of existing methods with the assist of raw data and Lasso algorithm. By testing on SPEC, we verify that our method can synthetize existing optimizing techniques to gain much better performance.

Acknowledgment

This work is supported by the Peking University Principal Undergraduate Research Foundation (URTP2013PKU004).

My sincere thanks go to my advisor: Dr. Yingwei Luo, Dr. Xiaolin Wang and Dr. Zhenlin Wang. And I would like to thank members in GIS Lab.

References

[1] Mattson RL, Gecsei J, Slutz D, Traiger IL. Evaluation techniques for storage hierarchies. IBM System Journal. 1970;9(2):78–117.

[2] Denning PJ. The working set model for program behavior. Communications of the ACM. May 1968; 11:323–333.

[3] Zhou P, Pandey V, Sundaresan J, Raghuraman A, Zhou Y, Kumar S. Dynamic tracking of page miss ratio curve for memory management. In Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems. 2004:177–188.

[4] Zhao W, Jin X,Wang Z, XiaolinW, Yingwei L. Efficient LRU-based working set size tracking. Technical Report CS-TR-11-01, Houghton, MI, USA, 2011.

[5] Zhao W, Jin X, Wang Z, Wang X, Luo Y, Li X. Low cost working set size tracking. In Proceedings of the 2011 annual conference on USENIX Annual Technical Conference. 2011.

[6] Tibshirani R. Regression shrinkage and selection via the lasso[J]. Journal of the Royal Statistical Society. Series B (Methodological), 1996: 267-2