Monitoring VirtualBox Performance

Siyuan Jiang and Haipeng Cai
Department of Computer Science and Engineering, University of Notre Dame
Email: sjiang1@nd.edu, hcai@nd.edu

Abstract—Virtualizers on Type II VMMs have been popular among non-commercial users due to their easier installation and use along with lower cost than those on Type I VMMs. However, the overall performance of these virtualizers was mostly found worse than the latter from VM user's point of view and, the reasons remain yet to be fully investigated. In this report, we present a quantitative study of VMM performance in VirtualBox in order to examine the performance bottleneck in this representative Type II virtualizer. Primary performance metrics for VMM and VMs are separately collected and analyzed and, implications of the results are discussed, with the monitoring overhead quantified through well-known CPU, memory and I/O benchmarks. Results show that VMM takes only a marginal portion of resources within the whole virtualizer for the case of VirtualBox and, our monitoring comes with merely a negligible performance overhead and perturbation to the virtualizer.

1 INTRODUCTION

Virtual machine monitor (VMM) is a system that provides virtual environments where other programs can run in a same manner as they run directly in the real environments. A virtual machine (VM) is used to represent the virtual environment that a VMM provides. The software running upon a VM is called guest software. The operating system running upon a VM is called guest operating system (guest OS). VMMs are categorized into two types [1]. Type I VMMs run directly on hardware, which means they have specific hardware requirements. Type II VMMs run upon operating systems, which means they are like other programs and do not require extra effort in installation and use.

Although convenient and widely used by common users, Type II VMMs suffer from significant performance issues. As King [2] has showed, VMs running on a Type II VMM (UMLinux) take more than 250 times more time than those running on a hybrid between Type I and Type II (VMware Workstation [3]) to execute a null system call in average. In this work, the performance of VMMs is estimated by running benchmarks or some particular system calls upon VMs and comparing their running time under different VMMs.

In contrast to this approach, we focus on investigating the performance bottleneck caused by unbalanced resource usage. We aim at monitoring performance metrics of VMM and VM separately, because we believe a better understanding of the overhead of Type II VMMs can lead to a practical and effective improvement on VMM design.

For this study, firstly, we choose VirtualBox as our object because it is an professional, open-source VMM project with a large user group. Secondly, we implement several performance collectors inside VirtualBox to record performance metrics, such as memory usage, of VMM itself and the VMs, which are running on the VMM. Thirdly, we implement a performance monitor to organize and aggregate the data collected from performance collectors. By comparing the resource usage of the VMM and that of the VMs, we investigate how much the VMM costs compared with the total cost.

Figure 1 shows the overall architecture of our project. We inspect the internal running state of VirtualBox VMM by instrumenting performance monitoring agents in the VMM source code. Pertinent information collected by those monitoring agents is then gathered in Performance Collectors. Then, Performance Collectors sends information to our Performance Monitor where designated performance metrics are calculated in runtime.

The experimentation includes three parts. The first one is running one or two VMs on the instrumented VMM. Monitored metrics of the VMM and those of the VMs are collected respectively. The monitored metrics are examined along with the running situations of VMs at the time, e.g. the startup phase of operating system. The second part is to see how the overhead of VMM may increase if VM uses more resources. Lastly, the overhead introduced by the instrumentation will also be gauged roughly by comparing major performance indicators attributed to VirtualBox processes from the legacy system monitor running on the host OS prior to and after the instrumentation is applied.

1. VirtualBox is an open source software under the terms of the GNU General Public License (GPL) version 2.
2 RELATED WORK

We address two categories of previous work related to our project topic including those on VMM performance, which is the main theme of our project, and those on source code instrumentation, which is the primary approach to our implementation of the project proposal.

The performance characteristics of virtual machines are one of the major concerns in VMM design [1]. However, virtual machines running on Type II VMMs can suffer a great performance lost compared to running directly on standalone systems [2] to the extent that the efficiency property has been treated as one of the three properties, as formal requirements, of any VMM [4].

In this context, performance of various VMMs has been analyzed and compared independently beyond simple functionalities for running statistics that are ported together with the release of the complete virtualizer package. To compare the performance of the VMM of VMWare and VirtualBox, Vasudevan et. al. create two virtual machines with each of the two visualizers, one running Windows and another Ubuntu Linux. They then measure the peak value of floating point computing power in GFLOPS using the LINPACK benchmark and, the bandwidth in Mbps using the Iperf benchmarking tool [5]. A similar but earlier work was done by Che et. al. in [6], where the performance of VMM in Xen and KVM was contrasted using benchmarking tools also including the LINPACK package. Different from these performance evaluations conducted in an indirect manner via running user-level applications on the top level of the VM system hierarchy, we directly gauge the runtime dynamics of the VMM’s internals with respect to its scheduling and controlling tasks with virtual machines running on it.

Another differentiation lies in the approach to measurements. While both works above are done through application-level (benchmarking) tools without modifying the VMM or other components in the visualizer, we aim at probing VMM through source code instrumentation. Among previous examples of approaches related to ours is to instrument in an operating system kernel so as to capture processor counters, which is applied to calculate performance metrics [7]. Further, the authors embed benchmarks into Linux kernel modules to eliminate most interferences of operating system and interrupts thus to reduce the perturbations of the instrumentation. Applied at the application-level, another example of source code instrumentation is to map dynamic events to transformations at the program source code level in the Aspect-oriented programming (AOP) [8]. By contrast, we instrument VMM also at the source code level but for collecting performance and resource usage information at runtime.

In fact, this instrumentation approach has been applied to many other areas. In SubVirt [9], a virtual machine monitor was instrumented for building a rootkit for the purpose of malware simulation. The virtual machine based rootkit (VMBR) was implemented to subvert a Window XP and Linux target systems in order to study various ways of defending the real-world rootkit attacks. For similar security research purpose, Garfinkel and Rosenblum use virtual machine monitor to isolate intrusion detection system (IDS) from monitored host in their virtual machine introspection architecture [10]. We focused on performance issues of the particular Type II VMMs and adopted the instrumentation approach solely for this purpose.

3 IMPLEMENTATION

To investigate the performance bottleneck of VirtualBox, we implement a Performance Monitor of VirtualBox to record performance metrics of the VMM and VMs. To retrieve the relative resource usage of different parts of VMM and VMs, we implement Performance Collectors inside VirtualBox, which collects resource usage information and sends it to Performance Monitor. The project is developed in C++ under Fedora 17 Linux with GCC 4.4.1. GUI is developed by using Qt 4.8.3.

3.1 Architecture of VirtualBox

Our project is built upon VirtualBox, which is a representative of Type II VMM products. The architecture of VirtualBox [11] is showed in Figure 2. As a Type II VMM, VirtualBox is a software running upon a host operating system (host OS). Above the host OS, there is a system service, VBoxSVC, which is the VMM of VirtualBox, maintaining all VMs that are currently running. Each VM is working with a VirtualBox client, which helps the VM interact with the VBoxSVC.

3.2 Overall Approach

Figure 2 shows how our project is implemented inside VirtualBox and how the data is transferred among the different components. Our implementation mainly has three parts: (1) Performance Collectors, (2) Performance Monitor and (3) VMMMonitor (GUI).
4 Evaluation

We have implemented the source code instrumentation approach to performance monitoring for VirtualBox VMM. With current configurations of the platform (see Section 4.3) where we develop and run all experiments, a complete build of the source package costs about 15 minutes without noticeable extra overhead in this regard introduced by our work. Figure 3 shows a screenshot of the running instrumented VirtualBox.

4.1 Metrics of Measurement

Currently two major categories of result have been collected and analyzed: (1) performance measurement of both VMM and running VMs; and (2) performance overhead and VMM perturbation of our instrumentation approach.

For the first category, primary metrics, CPU and memory usage, were monitored over a period of time. With a user-defined interval $t$, measurement of these metrics was updated in the runtime every $t$ seconds by retrieving the related dynamic records received from the instrumented IPerformanceCollector service and, the results were pushed to the VBoxPerfMon frontend hosting simple time-varying visualizations. These metrics have been chosen because they are the well-recognized strong indicators connected to the overall performance of the holistic virtualizer that common users can directly experience. Therefore, exploring these metrics, in particular those attributed to the VMM, is what answering our motivating questions asks for.

With the second category of metrics, we were concerned about the aggregate instrumentation overhead and perturbations to the VMM, including those of the performance collector interface extension and the VBox-PerfMon frontend. This was realized by running the original VirtualBox and the instrumented one on a given set of VM workloads separately and then comparing the CPU and memory statistics provided by top on host OS and benchmark scores obtained on guest OSes. These tests were included in our experimental design because it is important to inform if our work would cause or not too much overall performance penalties and how acceptable our approach would be in terms of the extraneous costs concerned by both system analysts and end users. More importantly, a heavy overhead of our work itself would even affect the accuracy of the performance metrics we obtained for VMM and VMs in addition to the whole virtualizer.

4.2 Experimental Design

Since our goal is to investigate possible reasons that would account for the unideal performance lying in the core VMM that is we supposed attributed to the unsatisfactory performance of the Type II virtualizer like VirtualBox, we separately measured performance metrics associated with running VMs and those purely dedicated to the core VMM alone. To do this, we tested the fluctuations of related metrics as responses to the gradual increase in the number of running VMs from 0 but up to 2 (our tests were limited to 2 VMs running concurrently due to the processor and physical memory limitations of our test platform). During the tests, we observed the metric changes on the part of VMM against that of the whole virtualizer at different overloads, including hosting the VM without applications running inside (i.e. on the guest OS) and that with benchmarks running inside. We used SciMark2 [12] as CPU benchmark and h2 and fop in the Dacapo benchmark suite [13] as memory and I/O disk benchmarks respectively for our experiments.
When comparing the performance original and instrumented virtualizer on a same set of tasks in order to measure the instrumentation overhead and perturbation, we run two VMs, one running guest OS of Windows XP sp3 and another Fedora 17 Linux, concurrently on the virtualizer with both executing a same benchmark for each test, totalling 3 groups of tests each on one of the three benchmarks described above. In each test, we collected both the aggregate CPU and memory usage stats associated with all VirtualBox processes from the host OS’s point of view, and the time spent finishing the benchmark test in both VMs reported by the corresponding benchmark program. Due to the limited test platform resources, we run each benchmark 10 times and took the average as the quantities for analysis.

4.3 Experimental Setup

The VirtualBox source code was instrumented and then rebuilt using the building scripts ported with the source package. During the experimentation, host machine was a portable HP® Compaq Presario CQ60 Notebook running Fedora 17 while mounted a single-core Intel® Celeron® 2.20GHZ processor with a 1024KB cache and, 2GB DDR2 physical memory. The Windows XP sp3 VM was assigned 512MB main memory, 16MB VRAM and 10G IDE virtual HDD. For the Linux VM, we configured 768MB main memory for it along with a 12MB VRAM and 10G IDE virtual HDD.

4.4 Results and Analysis

To demonstrate the different resource usage patterns, we monitoring VirtualBox under four situations, which are running VirtualBox with no VM started, running VirtualBox starting one VM, running VirtualBox starting two VMs, running VirtualBox with two VMs starting one benchmark, scimark2. Figure 4 and Figure 5 exhibit the results of our monitoring memory usage and cpu usage in the four situations. In Figure 4a, 4b, 5a, 5b, time slots spanning a period of 67 seconds are represented on the x-axes while in Figure 4c, 4d, 5c, 5d, time slots span a period of 111 seconds. In Figure 4, the y-axes are the memory costs of corresponding series. In the legends of Figure 4, VMM is the core VMM in VirtualBox; VirtualBox total is the entire VirtualBox which includes VMM and all other components, such as VMs, front-ends; Guest-XP+VMM is the sum of the memory cost of VM(with Windows XP operating system) and one of the core VMM; Guest-Fedora+XP+VMM is the sum of the memory cost of the two VMs(one with XP operating system and one with Fedora operating system) and one of the core VMM. In Figure 5, the y-axes are the CPU usage percentage of corresponding series. In the legends of Figure 5, VMM still is the core VMM in VirtualBox; user-level includes all the components in VirtualBox that are not VMs and the core VMM; Guest-XP is the VM with XP operating system; Guest-Fedora is the VM with Fedora operating system.

Comparing the four figures in Figure 4, we can see there is always a around 0.4GB gap between the total memory cost of VirtualBox and the memory of cost of VMM and VMs. The gap is slightly reduced after two VMs are launched in VirtualBox which is understandable because VMs use some of memory that VirtualBox has already allocated. The main observation in Figure 4 is the steady low memory cost of core VMM except when there is VM to start. When there is one VM to start, VMM only uses memory less than 0.1 GB for 10 seconds, while VMM uses almost the same amount of memory but for more than 60 seconds to start two VMs at the same time. Overall, the memory cost of VMM is almost zero to other components.

For CPU usage analysis, different from memory usage, we can see VMM is the major CPU resource user of VirtualBox in Figure 5a which is reasonable because with no VM started, VirtualBox has only started the VMM service underneath while other higher-level components are not launched. On the other hand, in the other three figures in Figure 5, we can see the CPU usage percentage of VMs and VMM are stay low all the time while the total CPU usage of VirtualBox is random and much higher. This leads to a conclusion that the CPU cost of VMM is low in most situations.

The second evaluation is conducted for estimating the overhead of our monitoring. We ran three benchmarks on the two VMs and recorded the finish time of each benchmarks, as showed in Figure 6. There are six columns in Figure 6, each represents a finish time of one benchmark in one VM. The solid black area represents the amount of time by which our monitoring has increased the finish time. Two VMs are running benchmarks at the same time under the same VirtualBox, so the overhead of our monitoring showed in Figur 6 is bigger than the overhead when VMs does not run concurrently. The proportion of the overhead in running fop benchmark is larger than other benchmarks, because the finish time is relatively short while there is certain fixed overhead f our method, such as initialization of metrics collection. Overall, the overhead of monitoring is between 1.6% and 39.0%.

Additionally, we also logged the entire resource usage of VirtualBox being monitored and not being monitored respectively, under the four circumstances: (1) two VMs running fop (2) two VMs running h2 (3) two VMs running scimark2 and (4) two VMs doing nothing. Table 1 and Table 2 shows the average usage of responding resources comparing the situations with our monitoring with those without our monitoring. The average usage is increased by around 5%.

5 Conclusion

We have presented a preliminary quantitative study on the performance-wise dynamics in the VMM component of the open source virtualizer VirtualBox as a representative of the Type II VMM that has been reported
to have performance issues in practical application. To do this, we developed a runtime performance monitor of the VirtualBox VMM by instrumenting the source code of the VMM, inserting performance inspectors that communicate with the core VMM module via COM interfaces. The monitor itself was designed as a separate module running as a child thread created by the main VirtualBox thread (VBoxSVC) and launched performance monitoring along with the start of the VirtualBox Virtual Machine Manager, the typical bootstrapping interface of the client used by common users.

We have measured the primary performance metrics, including memory usage and CPU usage, collected on the basis of the resources usage solely consumed by the VMM compared to that of the whole virtualizer. And, based on the data we have retrieved, we have described an analysis that is expected to inform the answers to our research questions proposed as what has motivated this project at the beginning. Our results obtained have implied that VMM should not be the real culprit of the overall unsatisfactory performance of the Type II virtualizer.

In addition, our measure of the overhead incurred by the instrumentation approach we implemented has been an evidence of a negligible cost, hence the promising practicality of the present work.

### References

Fig. 5: Cpu usage percentage monitoring in four situations


