Benchmark Object for Virtual Machines

Xia Xie, Qingcha Chen, Wenzhi Cao, Pingpeng Yuan, Hai Jin
Cluster and Grid Computing Lab
Services Computing Technology and System Lab
School of Computer Science and Technology
Huazhong University of Science and Technology, Wuhan, 430074, China
hjin@hust.edu.cn

Abstract—**The most cost effective approach to study the performance of virtualization technology is to use the benchmark. In this paper, we present a new evaluation methodology to deploy a benchmark evaluation with a Benchmark Object. By encapsulating three steps of traditional benchmark processes and implementing an instance from a Benchmark Object, users need not evaluate manually. Furthermore, a Benchmark Object simplifies the use of benchmarks and saves a great deal of duplicate work to deploy, configure and execute benchmarks and analyze their results. We develop the virtualization system performance evaluation management facility to implement and manage the concept of a Benchmark Object. We discuss this performance evaluation tool in detail, test the performance of the tool, and draw some conclusions.**

Keywords-Benchmark Object; Virtualization; Performance; Virtual Machine

I. INTRODUCTION

It has been a long time since computing system performance evaluation technology first appeared, and many performance evaluation methodologies and tools have been devised [10]. To reuse the traditional benchmark and evaluate all the VMs simultaneously, we present the new evaluation methodology: evaluating with a Benchmark Object. We develop a virtualization system performance evaluation management facility (VEMF) to implement the concept of a benchmark object. The VEMF includes three components: a graphical and command interface (called the user interface), supervisor, and daemon.

II. RELATED WORKS

Traditional performance benchmarks, such as those available through SPEC [7] or TPC [9], are developed without virtual machines. Virtualization first became available with the IBM VM/370 to allow legacy code to run on new hardware platform. Currently the most popular full virtualization system is VMware [5][8]. Several studies have documented the cost of full virtualization, especially on architectures as Intel x86 [9].

To address these performance problems, para-virtualization has been introduced, with Xen [2] as its most popular representative. Several benchmarks for virtual machines have been developed since the birth of Xen. Xenoprof [1] is a system-wide statistical profiling toolkit implemented for the Xen virtual machine environment. Another toolkit, XenMon [5], is more of a performance monitor than a performance evaluation tool. They both only support Xen.

Recently two import virtual machine performance evaluation tools, VMmark [10] and vConsolidated [4], are published. VMmark is a tile-based benchmark consisting of several familiar workloads running simultaneously in separate virtual machines. vConsolidated is a kind of workload for characterizing the performance of servers using virtualization to consolidate multiple physical servers [3][6].

Some single-workload benchmarks have been useful for quantifying virtualization overheads within a single virtual machine, and can be used for tuning portions of the virtualization layer.

III. BENCHMARK OBJECTS

A benchmark object is a prototype or parent of all the instances. The instances which implement a benchmark object deploy on different VM nodes (Fig. 1).

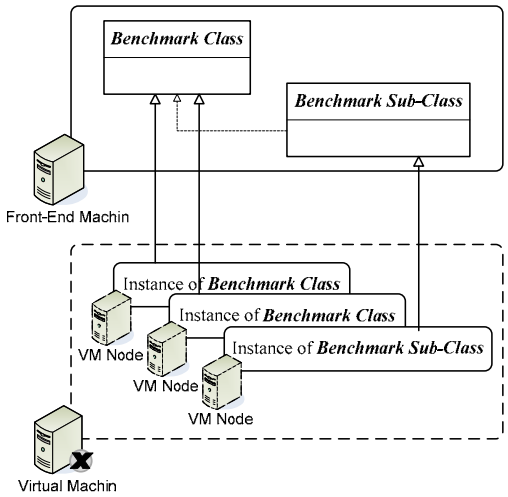


Figure 1. A benchmark object

A. Components of Benchmark Object

A benchmark object encapsulates the essential factors of traditional evaluation. It contains a set of benchmarks, the data declaration and an analysis chain. A benchmark set consists of varied benchmarks, such as LMBENCH, DBENCH and WBENCH. In a benchmark object, the data, generated by the

978-0-7695-3987-4/10 $26.00 © 2010 IEEE
DOI 10.1109/ETCS.2010.598

benchmark and fed into or coming out of the analyzer, is a significant component because it is the key contact between the benchmark and analyzer. Data is generated during the benchmark execution, so the benchmark object does not include any data. The benchmark object's components have three modification symbols: abstract, static and normal.

B. Extending Benchmark Object

Inheritance is an important method for reusing a benchmark object. This can be divided into single inheritance and multiple inheritances. Single-inherit style is a way of inheriting a single benchmark object. Combining it with a static component is an ideal way to deal with multiple virtual machines' data at the same time. The objective of multiple inheritances is to reuse benchmark objects.

C. Deploying Benchmark Object to VM

For evaluation on virtual machine nodes, instances must implement a benchmark object. There are three ways to implement the instance and deploy it on the VMs: normal-implementation, extended-implementation and static-implementation. Correspondingly, instances implemented from a benchmark object are divided into normal-instance, extended-instance and static-instance.

In the normal-instance implemented from a benchmark object, all analyzer members are normal members created by normal-implementation. As indicated in Fig. 2, a benchmark object copies a benchmark set and analysis chain to the target VMs, and an instance creates a data pool according to the data declaration.

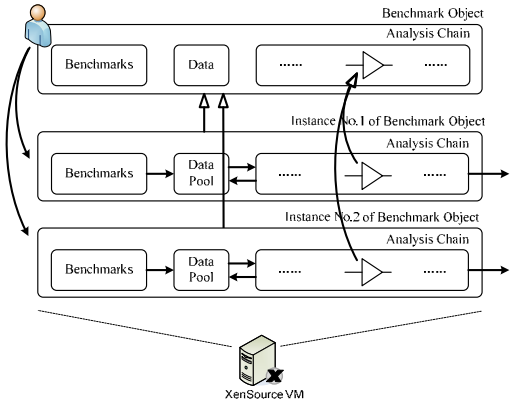


Figure 2. Deploying an instance by normal-implementation

Extended-implementation is an implementation and deployment type which is used to implement and deploy a benchmark object which includes an abstract analyzer member. Extended-instance implements a standard benchmark object and its own analyzers, as indicated in Fig. 3.

The last instance type, static-instance, is implemented from a benchmark object which involves a static analyzer. Static analysis is included in the benchmark object, and the instance uses the result by static-implementation. Static-instance is shown in Fig. 4.

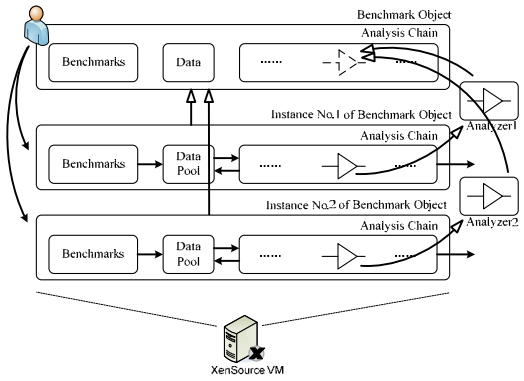


Figure 3. Deploying an instance by extended-implementation

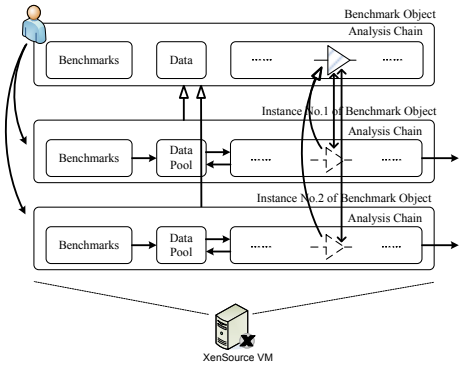


Figure 4. Deploy instance by static-implementation

IV. IMPLEMENTATION OF BENCHMARK OBJECT: VEMF

In this section, we discuss the implementation of benchmark object.

A. Architecture of VEMF

VEMF includes three components: a user interface (both graphical and command-based), a supervisor and a daemon. The user interface and the supervisor are deployed on the front-end machine which monitors the guests and instances of them. The daemon program is deployed on the virtual machine nodes, and monitors running the benchmarks and analyzing results. The detail system architecture is shown in Fig. 5.

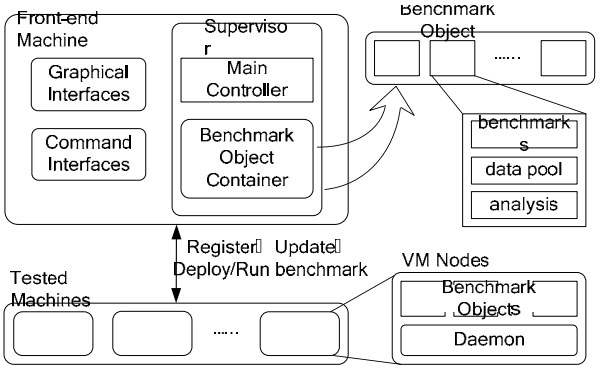


Figure 5. Architecture of VEMF

B. Supervisor

The supervisor consists of a main controller and benchmark object container. The main controller, which is the kernel of VEMF, interacts with the user through the user interface, executes the functions controlling the benchmark object, such as adding new benchmark objects to the container, sending control commands to the daemon on the tested virtual machine, receiving results from the daemon. It implements three forms of benchmark object.

C. Daemon

Currently the daemon service is implemented for both Linux and Windows. The daemon is similar to the main controller of the supervisor although it implements significantly fewer functions. It holds an instance of a benchmark object deployed on the local virtual machine. The daemon runs the benchmarks of an instance and collects the test results. It sends the results of its analysis for an instance, or sends them back for static analysis. A communication module provides communication with the supervisor service.

D. Workflow of VEMF

The workflow of VEMF contains four steps.

1) Install

The new benchmark object will be added into the container of the supervisor. For new benchmark objects, user uses or develops the related benchmark and analyzer according to the requirement. The new benchmark and analyzer will be added into related container. For inherit benchmark objects, user should indicate the father benchmark objects and explain modification to it. Both of them will update the object relationship graph.

2) Register

When the daemons start up, the daemons collect system information about the virtual machine node and then register this information with the supervisor which started up before all the daemons.

3) Implement benchmark objects

After choosing and submitting one or more benchmark objects and one or more virtual machine nodes, the user can implement all the chosen benchmark objects on the related virtual machine nodes according to the corresponding implementation type. The benchmarks for the two types of instance are implemented by way of deploy-implement. For absolute benchmark objects, supervisor copies normal members from container to target node, gets extended members from the user-appointed directory and inserts placeholder to static members. Daemon judges whether a member is a static member by the placeholder and transfers data to supervisor when it invokes.

4) Startup testing

The supervisor ensures that the selected benchmark objects have been deployed on their respective virtual machines' nodes. Then the supervisor simultaneously sends a benchmark startup command to the daemons on all of the selected virtual machine nodes. Due to the network delay, the command will arrive at the nodes at different times. On receiving the startup command, the daemon on each virtual machine creates a new process to execute the selected benchmarks and waits until the test finishes.

5) Analyze results

When the benchmarks are finished, daemons send signal that evaluation is over to supervisor. Supervisor waits signal of all the nodes and synchronizes process of all nodes. Then the daemons execute analysis chain and control the data flow. In the instance which does not have static analyzer, local analysis will be invoked to refine the data. When the refinement process is finished, the acquired data is sent to the data pool by daemons. Then the next analyzer process will be invoked and acquires the new data.

V. EXPERIMENTS

The physical computer on which we run all the evaluations has an Intel Xero E5310 with 1.6GHz CPU with 2 sockets and each having 4 cores and 4GB memory. We employ lmbench, a micro-benchmark system, which not only tests the CPU, but also the memory and I/O of computing system. In our case, lmbench is only used to test the CPU and memory because the VEMF has only a tiny influence on the I/O performance. We add a default configuration file and run a static test results analysis and a static performance analysis.

A. Impact of Performance

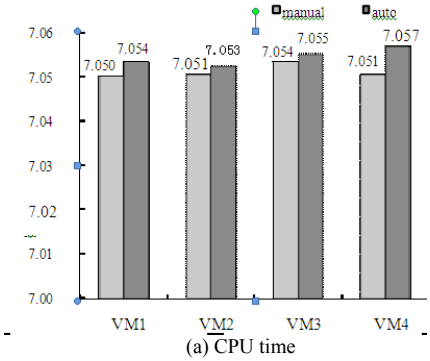
We boot four virtual machines on the physical computer. The CPU performance, memory throughput and memory response time are shown in Fig. 6.

Fig. 6a shows that CPU performance loss due to our system is between 0.001-0.006ns, and is insignificant compared with the CPU performance. This means that VEMF has very little influence on CPU overhead and does not take many CPU resources during evaluation. As for memory throughput, the difference between results generated manually and by our system is from 15 to 25MB/s, which is a very small value compared with the memory throughput (Fig. 6b). Fig. 6c shows a comparison of the memory response time. The difference between the two methods is less than 0.04ns and can also be ignored relative to memory response time. Fig. 6d summarizes all the performance impacts of using VEMF for virtual machines. We can see that the performance loss is no more than 0.7% for CPU time, memory throughput and memory response time. This means that using VEMF has a minimal influence on test results and can be ignored, especially for CPU time where the performance loss is less than 0.1%. For memory, the performance loss is 0.3-0.7%.

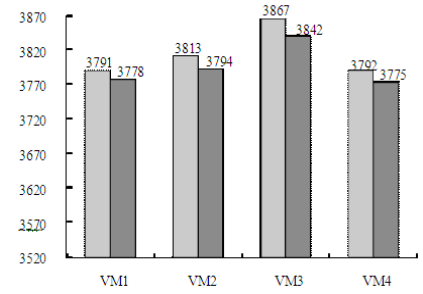
B. Startup Alignment

On the basis of the above testing we can insert some time-recording functions into both the supervisor and daemon. As shown in Fig. 7, once the startup command is sent, it takes 300 milliseconds for the first virtual machine to start the benchmark and extra 70 milliseconds or more for the other virtual machines. System latency will not influence the benchmark results because a delay of 0.3 seconds. It can be ignored since a general benchmark always run several minutes or even longer. Among all the four virtual machines, the time

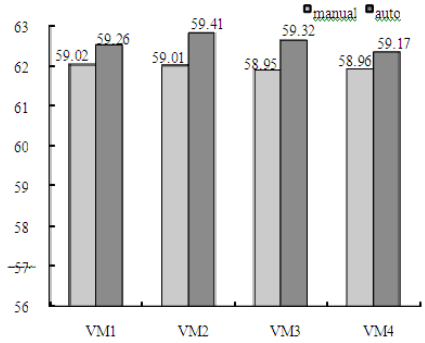
span between the first one to startup and the last one is only 0.082 seconds.



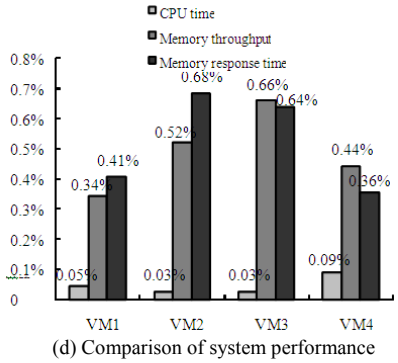
(a) CPU time



(b) Memory throughput



(c) Memory response time



(d) Comparison of system performance

Figure 6. Comparison of system overhead

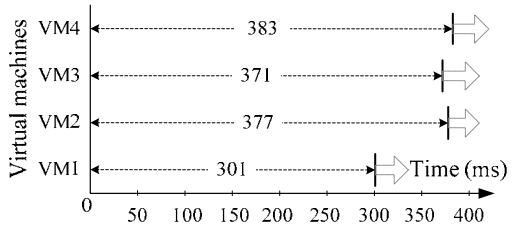


Figure 7. Startup alignment

VI. CONCLUSION

In this paper we present a new evaluation methodology, evaluation with a benchmark object, and develop our VEMF to implement concept of benchmark object. Some testing and analysis of the system has shown that it consumes minimum computing resources and has a minimal influence on the test results.

ACKNOWLEDGMENT

This paper is supported by National 973 Basic Research Program of China under grant 2007CB310900.

REFERENCES

[1] A. Menon, J. R. Santos, Y. Turner, G. Janakiraman, and W. Zwaenepoel, "Diagnosing Performance: Overheads in the Xen Virtual Machine Environment", In: *Proceedings of 1st ACM/USENIX Conference on Virtual Execution Environments (VEE'05)*, June 2005, pp.13-23

[2] D. Gupta, R. Gardner, and L. Cherkasova, "XenMon: QoS Monitoring and Performance Profiling Tool", *Technical Report HPL-2005-187*, HP Labs, 2005

[3] I. Ahmad, J. Anderson, A. Holler, R. Kambo, and V. Makhija, "An Analysis of Disk Performance in VMware ESX Server Virtual Machines", In: *Proceedings of the Sixth Workshop on Workload Characterization (WWC'03)*, October 2003, pp.65-71

[4] J. P. Casazza, M. Greenfield, and K. Shi, "Redefining Server Performance Characterization for Virtualization Benchmarking", *Intel Technology Journal*, Vol.10, No.3, 2006

[5] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, and M. Williamson, "Safe Hardware Access with the Xen Virtual Machine Monitor", In: *Proceedings of 1st workshop on Operating System and Architectural Support for the on demand IT infrastructure (OASIS)*, Oct 2004, pp.1-10

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and The Art of Virtualization", In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, ACM Press, 2003, pp.164-177

[7] Standard Performance Evaluation Corporation (SPEC). http://www.spec.org/

[8] The DBench project. Dependability Benchmarking. http://www.dbench.org

[9] Transaction Processing Performance Council (TPC). http://www.tpc.org/

[10] V. Makhija, B. Herndon, P. Smith, L. Roderick, E. Zamost, and J. Anderson, "VMmark: A Scalable Benchmark for Virtualized Systems", *Technical Report VMware-TR-2006-002*, Palo Alto, CA, USA, September 2006