

Woes of the Art of Virtualization Benchmarking

Zachary Shepherd, Todd Deshane, Cyrus Katrak, and Martin McDermott
{shephezj,deshantm,katrack,mcdermmn}@clarkson.edu

March 14th, 2008

Abstract

Virtualization is a growing field with new technologies constantly emerging. Because of this increase in the number of options, virtualization benchmarking is becoming more important. The current system for virtualization benchmarking is antiquated; it uses techniques developed for single-machine benchmarking which are not reasonable due to the increase in the number of possible configurations due to virtualization. A new system must be adopted. One example of such a system would be a completely modular definition-based benchmarking suite. Such a suite has been determined to be feasible and pieces have been implemented as a proof-of-concept through the benchvm project¹. This new system allows for users to perform fair, unbiased, and repeatable benchmarks without a significant amount of work. Such a system could provide more accurate and useful results for the virtualization community.

1 Introduction

Software performance benchmarking is a growing issue in the field of virtualization. More virtualization technologies are emerging and the relative popularity of each of the various technologies is in constant flux. One concern to potential users is performance, both relative to an unvirtualized environment and between different technologies.

Presently, there is no broad-spectrum comparison² of virtualization technologies available. This is not

to say there has not been research on the topic, rather each piece of research has focused on a few specific traits of a particular system, or set of systems, and is not compatible enough to provide a consistent broad-spectrum evaluation.[MST⁺05] [GGC05]

The goals of this paper are to identify the problems that have prevented a consistent multi-variable comparison, to outline one end-to-end flexible approach to virtualization benchmarking, and to evaluate its feasibility. The solution outlined here is just one of many possible solutions, but it

¹Benchvm, a benchmarking suite being developed at Clarkson University, is an implementation of some of the ideas discussed in this paper. (<http://benchvm.googlecode.com>)

²In this case, "broad-spectrum comparison" refers to a comparison across a large number of variables.

has a solid foundation in the principles of the experimental sciences and is based on the experiences discussed in various virtualization benchmarking related publications.[BDF⁺03] [CDD⁺04] [MHH⁺07]

2 Current Problem

Developing a set of fair, unbiased, repeatable benchmarks is a difficult task in any field. [Mog99] The independent variables involved must first be identified, and then a mechanism for systematically varying them must be developed. Biases must be considered and the objectivity of the tests maximized. [VMw] [Xen] [Mica] [Micb] Documentation must be complete, allowing future researchers to repeat experiments in the future.

Looking at performance benchmarking in general, there are a variety of variables to consider, such as hardware, operating system (including version), and type of benchmark being performed. When testing performance, benchmarks can evaluate the performance of individual components of a system (e.g. processing or network speed), a specific combination of components (e.g. memory-intensive computations), or the system's suitability for a specific task (e.g. web server testing).

Virtualization adds several levels of complexity to the testing process. Even in the case of homogeneity in both guest configuration and tests,

several additional variables, including virtualization technology being used, number of guests, guest resource allocation, and guest operating system (including version), are added to an already long list. In many real world cases, guests are heterogeneous in both load and configuration. Because the goal of benchmarking is to predict relative real-world performance, duplication of this scenario is necessary for a complete set of benchmarks.

Including heterogeneous guest configurations causes the number of variables to increase drastically beyond the point at which the traditional approach³ to benchmarking, which has become the defacto virtualization research approach, is remotely feasible. Even when making the assumption that the base configuration and hypervisor⁴ will have the same configuration for every test, there is a large number of possible configurations. For example, in the simple case of wanting to vary the numbers of guests between one, two, four, and eight, and two possible guest operating systems for each, there are over one hundred configurations. The same number of configurations would apply for varying the number of guests and having a binary choice for tests (either a guest is being tested or not). Adding another possible guest operating system or another test option increases the number of configurations to several thousand. When varying both parameters, the number of configurations increases

³For the purpose of this discussion, we'll consider a "traditional approach" to be one where the researcher configures a machine and its guests, record that configuration, and then start a script of tests

⁴In this case, "hypervisor" is used as a general term to refer to the base machine running the virtualization software. Other terms for this include "Virtual Machine Monitor" and "Domain0"

⁵Assuming both parameters are varied in a binary manner

to several tens of thousands⁵.

Another issue with the current paradigm is that benchmarking, which is supposed to consist of scientific measurements, is not treated like an experiment in the hard sciences. Currently, most research does not take all variables into account; it is assumed that it is enough to simply hold constant all variables other than the one being tested. While this provides valid benchmark data, it is conclusive only for the set of variables that were used in the original tests. To be able to extrapolate to other configurations, tests on a variety of configurations must be performed. In math, this would be comparable to examining a two-variable relationship by holding one variable (x) constant and varying the other (y) and measuring the output (z). If there is no data taken for other values of the x , then extrapolation to other values of y is valid only for the value of x used in the initial test. If the equation were $z = y^{(x-c)}$ and x were held constant at $c - 1$, the researcher might come to the conclusion that $z \propto y$, which is clearly only valid for the tested case of $x = c - 1$. There are several probable reasons for this lack of a scientific approach to benchmarking, one of which is the difficulty of getting a single set of constant variables under which all things to be benchmarked will function. Because a single set of functional constant variables is difficult to determine, researchers are disinclined to attempt to find another set.

3 Proposed Solution

3.1 Process

The ideal benchmarking process, from the point of view of a would-be benchmarker, might be: install the benchmarking program, select the desired configuration, and receive notification when the tests are complete.

In order to be customizable, the process would, of course, need to support users who wish to do things in a new or unique way. Ideally, the suite would, by default, support everything a user could ever want to do. A close, but more feasible, alternative is to include support for all common options and provide a mechanism for modular addition of options.

A compromise between ease-of-use and feasibility, from a development standpoint, provides a modular benchmarking program with the most common options included by default. These modules could be in the form of definitions – files in a standard format that describe everything from the installation process for specific configurations to the individual tests themselves. The suite would read in the definition files and command line arguments, set up the configurations, run the tests on each guest, and output a set of result files, complete with a description of the machine configuration.

3.2 Components

3.2.1 Physical

The physical components of the test setup would consist of two or more machines and, depending on the tests to be run, an isolated network. One machine would be taking the place of a human research assistant; it would

configure the test machine, run the tests, and record the configuration and test results. The remaining machines would be the machines to be benchmarked.

3.2.2 Software

One of the primary features of an ideal testing suite would be its complete modularity. Machine instructions for the installation of the software for the hypervisor, the creation and management of guests, and the tests themselves, would be in the form of definition files (common definitions would, of course, be included with the suite).

The tests to be run by the management machine would be queued in the form of a manifest file. The manifest would contain information about each test to be run and would have support for both sequential and simultaneous testing.

Installation definition

The installation definition would consist of a script to install a specific technology on a specific base distribution. This piece of modularity would allow for support for a variety of base operating systems.

Hypervisor definition

The hypervisor definition would consist of basic information about the desired configuration for the base machine. The definition would contain information about the base operating system and the installation definitions to be used to set up the system. This piece of modularity would allow for support for a variety of hypervisor configurations.

Guest definition

The guest definition would consist of the code for functions to create, start, stop, and clean up the guest based on the technology being used to store the guest and the virtualization technology being used. This piece of modularity would allow for support for a variety of storage systems and contribute to the support for a variety of virtualization technologies.

Test definition

The test definition would consist of the commands to run for testing and for output of the results in a standard format. This piece of modularity would allow for use of a standard output format and a standard way to interface with the tests, removing the need for every researcher to become intimately familiar with each benchmark interface.

Manifest file

The manifest file would be a file that allowed for controlling which guests and hypervisors run which commands in what order. The highest level of control would be test sets. Test sets would be run sequentially, with all tests in a test set being run simultaneously. Each test declaration would include one or more hypervisor declarations and a test definition. Each hypervisor declaration would include an IP address or hostname, a hypervisor definition, and one or more guest declarations. Each guest declaration would include the guest name and a guest definition.

This structure would allow for absolute flexibility. It would allow everything from one test on one guest on one

machine at a time, to one test on a variety of guests on a variety of machines at a time, to multiple tests on the same guest at the same time. Examples of each of these three uses would include: developing a baseline, testing scalability, and testing isolation.

Hooks

Another modular feature of an ideal implementation would be support for hooks. The ability to hook scripts to run at certain times, such as on the completion of a test, test set, or manifest, would provide the ability for the implementation of useful features like email notification and automatic transfer of data from the management machine to an external server.

3.2.3 Benefits

While the benefits of each of the individual components have been discussed in the preceding section, there are some far-reaching benefits of this approach that are not credited to an individual feature and may not be immediately apparent.

Sharing and collaboration

Because of the modular nature of definitions and manifests, a set of tests or configuration definitions developed by one researcher could be shared easily with other researchers, or submitted for inclusion as a default option.

Because of the uniform nature of the result sets and the meta-data automatically included, results could be assembled into large repositories. This would allow for ease in sharing of data and could allow for conclusions to be drawn from sets of data larger than any one researcher could feasibly collect.

Repeatability

If modularity and the uniform nature of results are both taken into account, tests can be perfectly repeatable. Information about the exact hardware configuration and software version can be found through use of the meta-data, and the exact test configuration can be recreated simply by using the same definition and manifest files.

Ease of customization

No one person or team of people can design a program to foresee every possible need of every researcher, but, given this solution's modular nature and the current state of the virtualization benchmarking community, it could be used to perform most, if not all, virtualization benchmarks that have been performed to date. Simply by creating custom definition and manifest files one can run almost any test structure.

4 Implementation

One of the best features of this solution – its modularity – makes implementation extremely feasible. There are no bleeding edge or untested technologies required; many of the principles at work have existed and have been used in other settings. Here, one example of a process for implementing the solution will be given as a way to demonstrate its absolute feasibility.

4.1 An Example

4.1.1 Definitions

The first step in this implementation of the proposed solution would be to

create a standard format for each of the definition files. Some could be designated as simply being executable scripts (e.g. the installation and guest definitions) implementing a specified set of functions while others (e.g. the hypervisor and test definitions) could be XML descriptions of the definition. These formats would allow for a wide range of flexibility.

An installation definition could simply be an executable script with an install flag and an uninstall flag. This would allow an installation of any level of complexity (from downloading packages to compiling from source) to be performed.

A guest definition would have flags for creation, deletion, starting, and stopping. Because the script could contain any code, it could be as simple as something that downloads a pre-built image or runs an external command, or as complicated as something that creates a storage space, installs a minimal operating system, installs requisite packages, and makes various configuration changes.

A hypervisor definition could contain the information about the hypervisor and which installation definitions need to be used. This would allow for maximum reuse of code.

A test definition would have flags for starting the test and converting the results to a standard format. This would allow the user ultimate flexibility in what a test could consist of.

4.1.2 The Manifest

The next step would be to define a format for the manifest file. One logical way to implement it would be as an XML file. The top-level element for the schema would contain one or

more test sets. Each test set would be run sequentially. Within each test set, there would be one or more tests. All tests within a given test set would be run simultaneously. This would allow for both sequential and concurrent testing. Each test element would contain a reference to the test definition to be used and one or more hypervisor elements. Each hypervisor element would consist of an IP address and one or more guest elements. Each guest element would have a name (to allow for specifying which guest to run which tests on) and a reference to the guest definition.

This configuration would allow for the greatest amount of flexibility, as any number of tests could be run on any number of guests running on any number of hypervisors. The tests could be run sequentially, simultaneously, or a combination of the two.

4.1.3 The Management Server

The final step in implementation would be to write the program that runs on the management server. This program would be to parse the definitions and manifest in order to figure out what needs to be done, connect to the hypervisors, run the installation commands, create the guests, and run the tests.

4.2 Evaluation of the Example

4.2.1 Feasibility

As part of a proof-of-concept prototyping exercise, development began on *benchvm*, a virtualization benchmarking utility. It is hoped that *benchvm* will eventually become a full implementation of the ideas discussed in this

paper. Benchvm implemented a simplistic version of the key concepts outlined in the proposed solution. Many of the problems mentioned in Section 2 were encountered and had to be addressed properly. The conclusion of this first phase of the benchvm project was that the implementation of the proposed solution was, in every respect, technically feasible.

4.2.2 Advantages

There are many reasons the proposed implementation, and the proposed solution in general, is better than the current paradigm. The current paradigm is often unscientific, results are difficult to reproduce, time is wasted in that many researchers perform the same menial tasks to make their testing possible, and bias is rampant [VMw] [Xen] [Mica] [Micb]. The proposed implementation would, even in the worst case of a completely proprietary implementation, be open enough that tests and results could be shared, research could be repeated easily, and additional tests could be added if a biased test-selection were suspected. In the ideal case of a completely open source implementation, all stages of the testing could be reviewed for potential bias and modifications could be made to add unforeseeable functionality.

4.2.3 Potential Issues

One issue with any modular system such as this is the potential loss of creativity; if researchers use the same system to structure their tests, tests will become more and more similar, and new creative tests might be dismissed because implementation would

be more difficult. Another potential issue is that using the system for purposes beyond the scope of the included definitions would require users to write their own. While this is still much easier than implementing a complete solution, as with the case of hard-to-implement tests, it might discourage users from including the tests in their research.

4.2.4 Further Improvement

There are, of course, things that could be done to improve this implementation. One area currently lacking features is the results parsing and management portion of the implementation. One example of a feature that could be added is a web-accessible results repository that automatically indexes uploaded data based on the meta information. Another possibility for further improvement would be the development of a large-scale results database that researchers from multiple institutions could contribute to. A third example in the same vein would be support for hooks that analyze the data based on specified parameters and generate comparative graphs on the fly. All of these would be valuable features, but their implementation is beyond the scope of this paper.

5 Related Work

There has been a substantial amount of work performed in the fields of virtualization and benchmarking and the combined field of virtualization benchmarking. Much of this research is relevant to this proposed solution and the primary developments are highlighted below.

In [CGS06], some of the challenges associated with virtualization benchmarking (consolidation characteristics, virtualization characteristics, and implementation challenges) are discussed. Consolidation characteristics are problematic since heterogeneous workloads may have different requirements and running them separately does not simulate realistic consolidated workloads. Virtualization characteristics refer to the challenges associated with the different virtualization environments. Implementation challenges include the lack of a hardware clock, hardware-specific features that may or may not need to be addressed, and hypervisor-specific differences (e.g. monitoring/profiling support).

[CGS06] presents the vConsolidate benchmark as an example of a generic virtualization benchmark and suggests that an industry standard for virtualization benchmarking should be created. SPEC has been developing a new industry standard benchmark that evaluates virtualization performance for data centers since 2006.[SPE] The vConsolidate developers suggest a scoring system based on the output of a “typical” workload. The paper fails to address the problem of a generic framework that the solution and example implementation proposed in Sections 3 and 4 provide.

Other attempts at virtualization benchmark solutions include VMmark [MHS⁺06] developed by VMware and vmbench developed by Kim-Thomas Möller at Universität Karlsruhe (TH) (TH)[MÖ7]. VMmark claims to be industry’s first virtualization benchmark. VMmark introduces the concept of a tile: a collection of virtual machines that run a set of di-

verse workloads. In [MHS⁺06], the use of real-world workloads in the categories of a mail server, java server, standby server, web server, database server, and file server are discussed. The hardware system requirements to use VMmark are substantial (e.g. 6GB of memory), making it unreasonable for many. Like vConsolidate, it fails to provide a generic virtualization benchmarking framework. The vmbench suite has many of the same goals as the suite proposed in Section 3 does, but the approach outlined in the paper is different in nature. The approach uses pre-virtualization techniques [LUC⁺06] that prepare the guest operating system kernel, a custom kernel based on the native kernel, to allow the hypervisor to cooperate with the guest kernel in the benchmarking process. This is similar in nature to paravirtualization, but its purpose, which is to provide a hypervisor-independent interface, is quite different.

One of the more recent developments is virtbench [Rus], developed by Rusty Russell, which is a set of primarily microbenchmarks used to assist programmers in optimizing their hypervisors. The virtbench suite is not a competitor to benchvm; the virtbench suite would fit well within the outlined framework in the form of a test definition.

6 Discussion and Conclusion

The current paradigm for virtualization benchmarking is not a sustainable one. The same menial work must be performed by every group wish-

ing to perform benchmarks, diverting their attention and labors from the more valuable aspect: the results themselves. One solution to this problem, and to many of the problems that a group wishing to do benchmarking would themselves have to overcome, has been outlined here. This solution has been designed based on the experiences documented in a variety of publications on the subject, but it is just one of many. One possible implementation of this solution, based on the solutions to similar problems used in other areas of Computer Science has also been outlined. The most important outcome is not the realization of this particular implementation, or an implementation of this specific solution, but that any good solution to the problem be implemented. Such a solution would save researchers valuable time, produce consistent and comparable results, and allow for easier sharing of information.

7 Acknowledgements

This paper would not have been possible without the support of the Clarkson Open Source Institute⁶, who provided machines for testing, help from Jeremy Bongio, Wenjin Hu, Ryan Lewis, and Keegan Lowenstein, who contributed by helping to brainstorm, test, or proofread, and guidance from Jeanna Matthews, our faculty advisor.

References

- [BDF⁺03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neuge-

bauer, Ian Pratt, and Andrew Warfield, *Xen and the art of virtualization*, SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles (New York, NY, USA), ACM, 2003, pp. 164–177.

- [CDD⁺04] Bryan Clark, Todd De-shane, Eli Dow, Stephen Evanchik, Matthew Finlayson, Jason Herne, and Jeanna Neefe Matthews, *Xen and the art of repeated research*, ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference (Berkeley, CA, USA), USENIX Association, 2004, pp. 47–47.

- [CGS06] J. Casazza, M. Greenfield, and K. Shi, *Redefining server performance characterization for virtualization benchmarking*, Intel Technology Journal (2006).

- [GGC05] Diwaker Gupta, Rob Gardner, and Ludmila Cherkasova, *Xenmon: Qos monitoring and performance profiling tool*, Tech. report, Internet Systems and Storage Laboratory HP Laboratories Palo Alto, 2005.

- [LUC⁺06] J. LeVasseur, V. Uhlig, M. Chapman, P. Chubb, B. Leslie, and G. Heiser, *Pre-virtualization: soft*

⁶<http://cosi.clarkson.edu>

- layering for virtual machines*, Tech. report, Fakultt für Informatik, Universität Karlsruhe (TH), 2006.
- [MÖ7] Kim-Thomas Möller, *Virtual machine benchmarking*, Master's thesis, Universität Karlsruhe (TH), 2007.
- [MHH⁺07] Jeanna Neefe Matthews, Wenjin Hu, Madhujith Hapuarachchi, Todd Deshane, Demetrios Dimatos, Gary Hamilton, Michael McCabe, and James Owens, *Quantifying the performance isolation properties of virtualization systems*, ExpCS '07: Proceedings of the 2007 workshop on Experimental computer science (New York, NY, USA), ACM, 2007, p. 6.
- [MHS⁺06] Vikram Makhija, Bruce Herndon, Paula Smith, Lisa Roderick, Eric Zamost, and Jennifer Anderson, *Vmmark: A scalable benchmark for virtualized systems*, Tech. report, VMware, 2006.
- [Mica] Microsoft, *Comparing web service performance vs test 1.5 benchmark results for .net 3.5/windows server 2008 vs. ibm web-sphere 6.1/red hat linux advanced platform 5*.
- [Micb] Sun Microsystems, *Web services performance comparing javatm 2 enterprise edition (j2eetm platform) and .net framework*.
- [Mog99] Jeffrey C. Mogul, *Brittle metrics in operating systems research*, HOTOS '99: Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems (Washington, DC, USA), IEEE Computer Society, 1999, p. 90.
- [MST⁺05] Aravind Menon, Jose Renato Santos, Yoshio Turner, G. (John) Janakiraman, and Willy Zwaenepoel, *Diagnosing performance overheads in the xen virtual machine environment*, VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments (New York, NY, USA), ACM, 2005, pp. 13–23.
- [Rus] Rusty Russell, *Virtbench*, <http://ozlabs.org/rusty/virtbench/>.
- [SPE] *Spec to develop standard methods of comparing virtualization performance*, <http://spec.org/specvirtualization/>.
- [VMw] VMware, *A performance comparison of hypervisors*.
- [Xen] XenSource, *A performance comparison of commercial hypervisors*.