# An XML-Based Automation Framework For Benchmarking Multiple Virtual Machines

Travis Finch, St. Edward's University

## Abstract

Server virtualization provides a mechanism to use system resources more efficiently by executing multiple, separate instances of operating systems on virtual hardware simultaneously. As more organizations shift toward a virtualized infrastructure, tools are needed that accurately measures the system performance and overhead requirements demanded by it. The purpose of this research was to introduce the Message-Passing Interface Automation Framework (MAF) as a lightweight, yet flexible framework for simultaneously executing a diverse sequence of applications on discrete virtual machines to test system capacity. MAF proved to provide an excellent mechanism for VM communication and synchronization.

## Introduction

As Moore's Law continues to hold, causing system performance to roughly double every 24 months, the core functionality of applications running on enterprise servers has not changed as drastically. Although hardware improvements, such as multiple cores in CPUs, have become commonplace and inexpensive, traditional operating systems and software applications have not changed much to take full advantage of these abundant resources. Indeed a primary goal in most Information Technology organizations is how to best utilize the available computing resources and reduce overall costs [1]. Server virtualization provides a mechanism to use system resources more efficiently by executing multiple, separate instances of operating systems on virtual hardware. The container, called a virtual machine monitor (VMM), accomplishes this by multiplexing the physical hardware, to simulate the execution of multiple operating systems at the same time [2]. Virtualization also provides a way to separate logical server roles into discrete parts. With proper planning this practice can ease the duties of system administration and maintenance.

As more organizations shift toward a virtualized infrastructure, utilities are needed to stress and accurately measure the overall system performance and overhead requirements so end-user expectations remain realistic. The majority of benchmarks are not designed to quantify performance measures in server consolidation because they primarily focus on measuring a well-defined target component of the system [3]. To harness the capability of these pre-existing benchmarks in an environment consisting of multiple virtual machines, a mechanism is needed that allows execution of them, and communication between each virtual node without continuous user intervention.

The purpose of this research is to introduce the Message-Passing Interface Automation Framework (MAF) as a lightweight, yet flexible framework for simultaneously executing a diverse sequence of applications on discrete virtual machines to test system capacity. The Message-Passing Interface (MPI) will be used as a synchronization mechanism that allows discrete virtual machines to communicate. MPI is a portable, language-independent application-programming interface used to allow a group of computers to transfer data over a network [4]. This research will use MAF and the benchmark environment to answer two questions: 1.) Does the presented framework provide an adequate mechanism for virtual machine setup and synchronization; 2.) What is a reasonable limit for the number of virtual machines executing simultaneously without causing severe performance penalties?

## Related Work

Padala et. al [1] evaluated two VMMs, Xen and OpenVZ, by consolidating multi-tiered systems. They compared application performance, resource consumption, scalability, and low-level system metrics to the original system configuration. The experimental results indicated that Xen incurred higher virtualization overhead across the board with an average response time increase of over 400%. OpenVZ measured an increase of 100% in average response time. The work also showed that virtualization performance degradation increased as application workload became more CPU intensive.

A performance evaluation of storage sub-systems on the VMware ESX VMM was conducted by Ahmad et. al [5], which used microbenchmarks (Iometer) to measure virtual machine behavior on several different storage sub-systems (direct-attached disk, RAID array, SAN). Iometer is an I/O subsystem measurement and characterization tool for single and clustered systems. It can be configured to emulate the I/O load of any program or generate synthetic load [6]. Results showed that I/O behavior of virtual machines closely matched that of the native server, but the effects of simultaneous virtual machine instances were not measured.

The majority of benchmarks are not designed to quantify performance measures in multiple virtual machines and server consolidation. Makhija et. al [3] have worked to develop VMmark, a benchmark application that measures the capability of a virtual server environment consisting of many virtual machines and a diverse work load that targets various physical hardware components. The unit of work for a benchmark of virtualized consolidation environments, referred to as a tile, is defined as a collection of virtual machines executing a set of diverse workloads. The total number of tiles running or a higher score for a single tile provides an estimate of the physical system's capacity. Although VMmark provides accurate and consistent results over many runs, deployment of a test environment can take more than a week. Future work includes the capability to measure the performance of larger systems.

To communicate between each virtual node VMmark uses the Software Test Automation Framework (STAF), an open source, multi-platform, multi-language framework designed around the idea of reusable components [7]. STAF automates the distribution, execution, and results analysis of the test cases. While this may sound like a useable solution, it was designed only with software testing in mind. Its extensive capability could create a footprint that interferes with the system benchmark results. Also, initial configuration and functionality can be an obstacle for achieving fast deployment.

## Setup & Configuration

The first task before a user can benchmark a virtual machine system is to configure a test environment. A problem with performance analysis of virtualization software is that the cost of initial test environment setup can be extraordinarily high. A primary goal of this research was to create a method that allowed deployment of a test environment within a matter of a few hours. The host operating system (OS) being used is a minimum Debian 4.0 distribution (Linux 2.6 kernel) and OpenVZ [8] as the VMM. The host physical system consists of an AMD Athlon 2500+ w/NVIDIA nForce2 chipset, Kingston 1 GB DDR400 RAM, and a Maxtor 200 GB ATA100 7200 RPM HDD with a transfer speed rated up to 100 MB/sec.

The initial step in configuration after installation of the host OS is to install the OpenVZ kernel. This can be done with either precompiled packages or compiling your own kernel. Next, a template cache of a guest OS must be created and exported. Again, a minimum Debian 4.0 package was used. To install the software on the guest system (SSH, GNU GCC, MPICH, MAF, benchmarks), the user can enter the VM and install the necessary tools as if he were on any traditional command line interface. After this is accomplished, the framework source code must be copied into the VM file system and compiled using **mpicc**.

Now, a tar ball can be created from the private VM file system, usually located in the **/var/lib/vz/private/<VM ID>** directory, and must be copied into the **/var/lib/vz/template/cache** directory. Then, a custom shell script is used on the host OS to start multiple guest OS instances using the template cache created above. The script configures network settings and also creates and deploys custom **machines.LINUX** and DNS **hosts** files to each operating system instance to ensure proper network communication. Another shell script is used to shutdown and destroy the VMs. Once the host OS and OpenVZ is installed, this process can be replicated in less than an hour on a different physical machine.

## Design & Implementation

The design for MAF extends the principle that the actual benchmark application and the automation framework are totally separate entities [9]. The framework does not concern itself with the results of the benchmark, but only initializes the execution of it and checks for completion. Benchmark utilities should only detail how the particular target component should be stressed, while the framework is only the execution environment. The development of each component requires completely different technical skills and design considerations. The design is also based on the principle that the automation framework should be application-independent [9]. Many different benchmarks already exist that test different aspects of a computer system's performance. This software is written in many different programming languages, and MAF's design will allow them to be fully utilized.

MAF uses MPI as a synchronization mechanism that allowed discrete virtual machines to communicate. Each virtual node will receive an XML data file with various instructions to perform. The instructions will include:

**invoke:** The virtual node will invoke a shell command using the **system** function and wait for its completion

**invoke_nonblock:** The virtual node will invoke a shell command using the **system** function in a separate thread and immediately continue its instruction sequence

**timeout:** The virtual node will block for the amount of time specified in seconds in the XML data

**synchronize:** All of the virtual nodes in the communication group will block until they reach a barrier. This enables them to begin execution again at the same time.

The rules for deploying different XML files to each virtual node are contained in another XML configuration file. It specifies an XML instruction set for a particular host name. When the framework is initialized, each virtual node passes its unique host name to the master node. The master node then responds with the instruction set for that particular virtual machine. Each node blocks until the entire cluster has received its execution directions, and then begins concurrently.

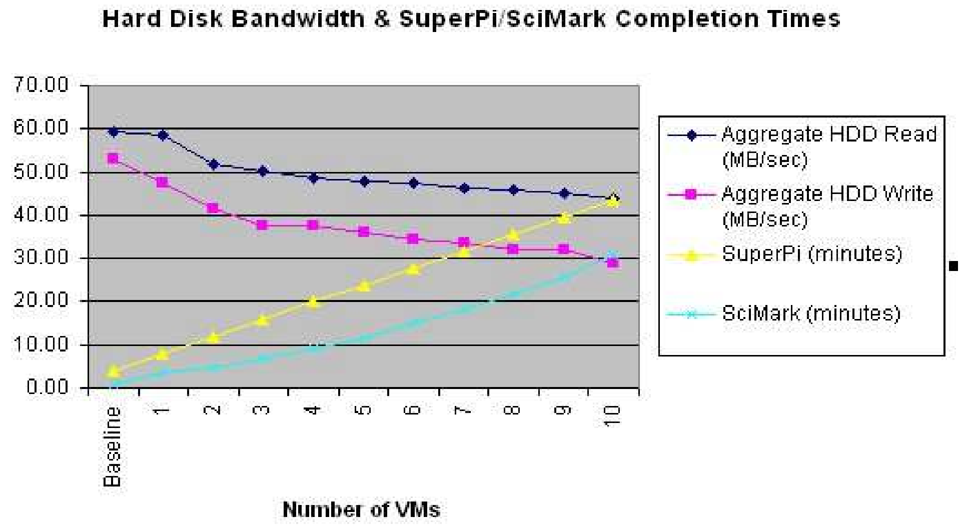
A major obstacle encountered with the design and development of this framework is that the implementation needs to be lightweight and developed with cross-platform compatibility in mind. Organizations have the capability of virtualizing different operating systems, so it seems logical that they would also want to capture the performance aspects of them all. Rather than developing an XML parser from scratch, Simple XML Parser [10], an open source OS independent parsing engine designed for devices with limited resources was used. To invoke a process, the **system** function from the C Standard Library was used due to predictable behavior across multiple platforms. POSIX thread libraries are also used to invoke a process without blocking.

## Benchmarks

In order to test various aspects of the system, three diverse benchmark utilities were chosen. The first test, Super PI [11], calculates pi up to 32 million digits to test the performance and stability of the target system. It uses the Gauss-Legendre algorithm, which is very memory-intensive. The second benchmark, SciMark [12], is a composite Java benchmark measuring the performance of numerical codes occurring in scientific and engineering applications. It consists of five computational kernels: Fast Fournier Transformation, Gauss-Seidel relaxation, Sparse matrix-multiply, Monte Carlo integration, and dense LU factorization. The final benchmark is a custom C program written to test hard disk drive (HDD) bandwidth. The **read** and **write** system calls are used to read and write one kilobyte blocks of data to and from the HDD. In a single iteration of the benchmark, each node writes and reads one gigabyte of data.

## Results

In order to properly analyze the virtual machine performance, baseline results were calculated on the host hardware for comparison. The disk bandwidth measured 59.192 MB/sec for reading, and 53.065 MB/sec for writing. Super PI completed in 3 minutes 53 seconds, and SciMark in 35.67 seconds. After, a single VM was launched and the benchmarks were executed again on the host OS and VM simultaneously. As expected, aggregate completion time for Super PI roughly doubled to 7 minutes 52 seconds. Read disk bandwidth also had comparable performance to the baseline results at 58.58 MB/sec, but write speeds had a steep decay to 47.51 MB/sec, a 10.46% decrease. The results for SciMark were an aggregate completion time of 3 minutes 30.96 seconds, an increase of 491%.



Next, VMs were added one at a time until ten were running the benchmarks simultaneously, and the performance results remained comparable throughout. Super PI scaled linearly as more virtual machines were added. Performance degradation of HDD bandwidth stayed consistent, with write bandwidth decaying at a higher rate than read bandwidth. With SciMark performance continued to worsen, although it was not as severe as the level seen during the initial VM addition. The sharp performance degradation may have been caused by poor memory swapping performance caused by context switches of VMs performing large matrix calculations. While this result was unexpected, investigation of the true cause of this phenomenon is not within the scope of this paper.

## Conclusion

To answer the questions stated above, MAF proved to be an excellent mechanism for virtual machine communication and synchronization. The test environment was very stable throughout the benchmarking process with no system crashes or node communication exceptions. At this point in time to answer the second question from above, it is difficult to quantify an encompassing limit of virtual machines executing simultaneously. Different organizations will require different hardware demands with variation in system load levels. Currently, the framework and benchmarks can help organizations quantify hardware sharing expectations, but future work including more thorough analysis is required in choosing benchmarks that mimic runtime behavior of real-world software applications.

The design and development of the VM environment and MAF successfully met the goal of rapidly deploying a test environment of multiple virtual machines, and utilizing the framework to gather accurate performance data on a specific hardware component. This work represents a beginning in the development of tools designed to test various components multiplexed and stressed by multiple virtual machines concurrently. More accessible tools will allow Information Technology organizations to configure their environment to best utilize the available computing resources and reduce overall costs.

## Cited References

1. Padala, P., Zhu, X., Wang, Z., Singhal, S., Shin, K., Performance Evaulation of Virtual Technologies for Server Consolidation. Available: http://www.hpl.hp.com/techreports/2007/HPL-2007-59.pdf. Accessed June 1, 2007.

2. Smith, J., Nair, R., Virtual Machines: Versatile Platforms for Systems and Processes. San Francisco: Morgan Kaufmann; 2005.

3. Makhija, V., Herndon, B., Smith, P., Roderick, L., Zamost, E., Anderson, J., VMmark: a Scalable Benchmark for Virtualized Systems. Available: http://www.vmware.com/pdf/vmmark_intro.pdf. Accessed May 19, 2007.

4. Gropp, W., Lusk, E., Doss, N., Skjellum A., A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard, Parallel Computing, 1996; 22: 789-828.

5. Ahmad, I., Anderson, J., Holler, A., Kambo R., Makhija, V., An Analysis of Disk Performance in VMware ESX Server Virtual Machines. Available: http://www.vmware.com/pdf/wwc_performance.pdf. Accessed June 1, 2007.

6. Intel, Iometer project [Internet]. 1998; Available from: www.iometer.org.

7. Rankin, C., The Software Testing Automation Framework [Internet]. IBM Research; 2002; Available from: www.research.ibm.com/journal/sj/411/rankin.html.

8. OpenVZ, Server Virtualization Open Source Project [Internet]. 2007; Available from: openvz.org.

9. Nagle, C., Data Driven Test Automation Frameworks [Internet]. 2002; Available from: safsdev.sourceforge.net/FRAMESDataDrivenTestAutomationFrameworks.htm.

10. Essman, B., Simple XML Parser [Internet]. 2002; Available from: sourceforge.net/projects/simplexml.

11. Super PI [Internet]. 2005; Available from: www.xtremesystems.com/pi.

12. Java SciMark 2.0 [Internet]. 2004; Available from: math.nist.gov/scimark2.