The Heisenberg Measuring Uncertainty in Lightweight Virtualization Testbeds

Quan Jia, Zhaohui Wang, and Angelos Stavrou
Department of Computer Science, George Mason University, Fairfax, VA
{*qjia, zwange,astavrou*}*@gmu.edu*

Abstract

The need for large-scale experimentation testbeds involving several hundred, or even thousands, of nodes is undeniable. Testbeds including Emulab [10], and Deter [5] are heavily used for both research and application testing. To scale even further and shed some of the limitations that the relative small number of physical nodes impose, researchers have turned to full virtualization [8] and lightweight, container-based virtualization [16, 10]. Virtualization allows running multiple virtual execution environments (VEEs) per physical host.

In this paper, we evaluate the use of hundreds of lightweight containers as a testbed to measure the performance of simple applications. We show that, although economically and technically compelling, virtualization has some limitations due the sharing of host resources (CPU, network, memory and disk) among same-host VEEs. Determining the number of VEEs that can be deployed in a physical machine without interfering with the fidelity of the experiment is not a trivial task and it cannot be estimated or computed ahead of time using aggregate utilization of individual resource. Furthermore, monitoring the health of an experiment by measuring the individual resource utilization can affect the behavior of the service under test. Therefore, we observed what we call a "weak" form of the Heisenberg uncertainty principle for host resource measurements: increasing the precision and fidelity of the resource measurements can interfere with the behavior of the experiment. We believe that this observation holds in general but it becomes more pronounced when we instantiate hundreds of VEEs due to the necessary context switching.

1 Introduction

Nowadays, the overwhelming majority of applications and services are being performed by large-scale distributed software systems. Being able to emulate the scalability, quality of service, fault tolerance, security properties, and steady state behavior of such planetary size systems ahead of their deployment is highly desirable. Testbeds including Emulab [10], and Deter [5] offer researchers and practitioners the only viable platforms to test their ideas and application beyond custom built corporate clusters.

Although a step forward, current testbed platforms fall short when it comes to scaling to tens or even hundreds thousands of application or services instances. To address this shortcoming and to allow existing testbed infrastructure to scale even further, researchers have turned to virtualization technologies [16, 10, 8]. Indeed, Virtual Execution Environments (VEEs) are an economical way of scaling beyond the limitations imposed by the relative small number of physical nodes. VEEs can dramatically increase the perceived number of application or service instances by one or two orders of magnitude depending on the available host resources and the type of virtualization.

In this paper, we study the advantages in terms of scalability but also the limitation from the use of hundreds of lightweight containers as a performance evaluation testbed. We do so by performing a range of simple experiments on a single but relatively powerful host. Although superior in terms of scalability, lightweight virtualization has itself limitations: determining the optimal number of VEEs that can be accommodated in the single host without interfering with the fidelity of the experiment is not a trivial task. Moreover, we show that it is infeasible to compute the VEE capacity of the underlying hardware based on static or even aggregate measurements of host resources (CPU, network, memory, and disk I/O).

Initially, we attributed this result to the lack of precise, per-VEE measurements that would enable us to effectively monitor the behavior of the application in each of the containers. However, in our effort to increase the fidelity of our measurement collection infrastructure to track the health of each individual VEE, we run into another limitation: increasing the frequency of measurement can decrease the number of concurrent containers we can utilize without interfering with the performance of the experiment itself. Therefore, we observed what we call a "weak" form of the Heisenberg uncertainty principle for

host resource measurements: increasing the precision and fidelity of the resource measurements can interfere with the behavior of the experiment, reducing the number of "usable" VEEs. Although uncertainty principle was originated from physics, a system that has to share its resources among many different tasks can exhibit the same behavior. Therefore, we believe that our results holds for any process and any system but it becomes more pronounced for a testbed with hundreds of VEEs due to the necessary context switching among the different CPU tasks.

2 Testbed System Architecture

Our initial goal was to setup a single machine testbed where we use Virtual Execution Environments (VEEs) to perform experiments. We were interested in generating experimental scenarios that would be able to scale to potentially thousands of VEE instances. We would like to do so without exceeding the available host resources. With host resources, we refer to CPU, network, memory and disk for our Dell PowerEdge 1950 server equipped with two Quad-Core Intel Xeon E5430 (2.66GHz) processors, 8GB RAM and Gigabit Ethernet NIC. The overall system architecture is illustrated in figure 1. For the purpose of our experiments, we used OpenVZ [15] (kernel patch) version: ovz009.1 on a vanilla Linux kernel version 2.6.24 to instantiate the VEEs. However, any other lightweight container-based systems such as VServers [16] would suffice and the produced results are not dependent on the specific container solution.

To lower the disk and memory requirements, Unionfs[23] plays a critical part in our system. It is a stackable filesystem service which "merges several directories into a single unified view" from each containers' point of view. There are three major advantages of using Unionfs in our system. First, instead of creating a separate copy of filesystem for each container, we only create one template filesystem and share it among all containers. This results in tremendous disk space savings which enables to scale to thousands of simultaneous containers instances in a single machine. Second, Unionfs facilitates memory sharing. When the same user program is running in multiple containers with Unionfs mounted filesystem, its binary image would be loaded only once in memory. Thus, we have automatic savings both in terms of memory access time and space allocation. Third, any changes in system configuration and software is quickly propagated to the containers through modifications on the base template. The base template is mounted to each container as read-only root "/" while a write-enabled slice is mounted on top of the root allowing each container to store its state in a separate directory on the host.

Except the use of Unionfs, our testbed was composed from out-of-the-box software components. Our which, at first glance, would have enabled the quick estimation of the required resources through both static and dynamic (on-line) analysis. We present this analysis in the next section.

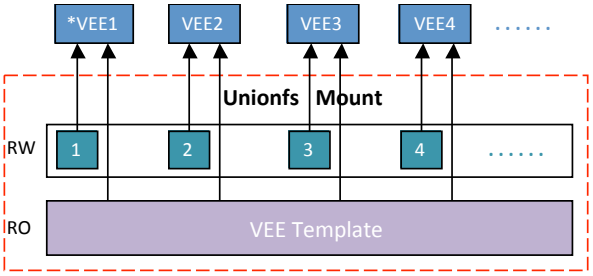

Figure 1: Lightweight Virtual Execution Environment (VEE) single-host testbed architecture.

process name	RSS	shared	non-shared
init	860	604	256
syslogd	640	508	132
dbus-daemon	684	508	176
sshd	992	644	348
sum		2264	912

Table 1: ubuntu-8.04-i386-minimal container template process memory consumption in kB

process name	RSS	shared	non-shared
init	616	528	88
minilogd	632	208	424
sshd	1144	836	308
sum		1572	820

Table 2: centos-4-i386-minimal container template process memory consumption in kB

3 Estimating Resource Utilization

3.1 Pre-experiment Resource Computation

In order to roughly determine the number of VEEs that can be held in our testbed, we want to estimate the system resource utilization before carrying out any experiment. In our testbed, the size of the Linux operating system is 371MB. Therefore, if we need to instantiate N containers without using UnionFS, they would require a total of 371*N MB. However, with UnionFS, we only require one copy dramatically decreasing the amount of required disk space. This is a significant difference especially for high density testbed systems since such system usually run exactly the same copy of the programs and configurations. Due to the copy-on-write slices, the disk usage for each container also remains small.

Lightweight virtualization has a small memory footprint which allows to save the state of each container faster and inexpensive in terms of memory overhead. After bootstrapping the containers, there is a small number system processes running. Of course, the number of initial processes depends on the system configuration. In tables 1 and 2 depict the number of processes and their memory utilization.

These tables allow us to measure how much resident physical memory pages are actually used by a single container. For example, for ubuntu 8.04 guest OS template to populate the containers, 3 basic process are launched, and they are allocated with 2264kB shared memory pages and 912kB non-shared pages. Using a physical memory analysis tool [13], we notice that the shared pages are not only shared within container (the .so lib files within the same container), but also across the containers. The code analysis shows that the inode numbers of the shared files remain identical from the kernel's perspective: when kernel performs the ELF loading, kernel will map the same binary file with identical inode number to the same memory region.

Using this, we can derive the simple formula to compute the memory consumption per container:

The estimate memory requirements for C containers is:

$$M = 912 \cdot C + 2264 \text{ (in kB)}$$

Thus, the Memory consumption per container is:

$$912 + \frac{2264}{C} \text{ (in kB)}$$

where C is the number of containers launched.

The second part of the above formula indicates that the shared memory page consumption is not actually linear: the more containers launched, the less it pays for shared memory. The container based virtualization only needs 3.1MB memory for a basic running container.

Unfortunately, the pre-computation of resources for each VEE can only be applied for memory and disk space, to calculate the CPU and network requirements, we have to rely on the run-time analysis of the running application.

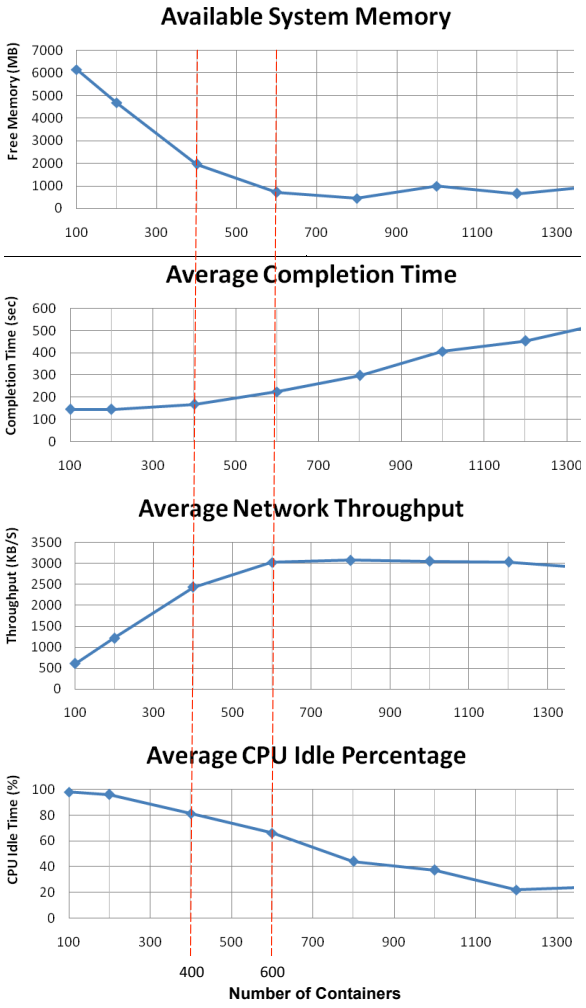

Figure 2: Aggregate resource utilization for wget and for a varying number of containers. Notice that although there is a noticeable change in the experiment completion time for approximately 400 containers, the rest of the resources appear to be nominal. It is not until running 600 concurrent VEEs that the network throughput reaches a maximal point.

3.2 Runtime Evaluation of Concurrent VEEs

To quantify the number of VEEs that a single host testbed can support, we had to perform a set of controlled experiments using a variable number of VEEs and measure our capability to estimate the number of VEEs that the underlying hardware can support. To alleviate measurement errors due to heterogeneous load and resource allocation requirements, we chose to run the the same application inside all VEEs. In addition, we used a light monitoring process running outside the containers (*i.e.on the host)* to sample the real-time values of resources including network throughput, CPU and, memory utilization for each of the VEEs. Our initial goal was to identify the number of containers we can execute in parallel on the same host without depleting the host resources or otherwise contaminate the obtained experimental and performance results. To that end, we implemented a monitoring process that depends on the Linux proc file system [2]. This monitoring process was performing periodic reading of the resources status using the available "/proc" entries. To avoid erroneous measurements, we used a real-time kernel scheduler setting to optimize the performace of our measuring process. We caught network data from file "/proc/dev/net" within the directory of each container under "/vz/root". Container CPU usage was obtained from "/proc/vz/vestat" on the host while memory utilization was calculated from "/proc/user_beancounters".

Initially, we set our measuring sampling interval to one

measurement per three seconds and we started GNU wget in each container. Wget is a simple console application designed to retrieve files using HTTP protocol among others. To prevent initial resource contention, we initiated wget in randomized intervals ranging from ten (10) to twenty (20) seconds. Each instance downloaded several web pages approximately 400 KB in size from an Apache web server hosted on another physical machine in local network. The capacity of the web server was configured to 4000 requests, which was never attained in our experiments, meaning that the bottleneck was on the client side. We selected wget as our application for our performance evaluation because it is not CPU or memory intensive but rather exercises the network and disk I/O. In addition, the network behavior of wget is very sensitive to scarcity of resources and thus provides an easy aggregate measurement for the overall system performance.

We performed the same experiment on groups of 100, 200, 400, 600, 800, 1000, 1200 and 1400 VEEs, and collected the real-time system network throughput, CPU and memory usage. The completion time for each container was recorded in memory and the averages are calculated after the end of each experiment. Figure 2 illustrates our experimental results using the aggregate resource measurement. For each of the graphs, we measured the resource values using our real-time measuring process every three-second intervals and for the entire duration of the experiment. Our aim was to identify a resource that would show signs of depletion or otherwise indicate that the number of VEEs is not sustainable. As a ground-truth for the actual concurrent VEE capacity of the host, we measured the completion time for each experiment. If the VEEs do not interfere with each other in any way, the experiment would finish at approximately that same time.

Our initial expectation was that we would be able to pinpoint the number of VEEs that we can scale to without losing precision by just looking at the aggregate resource utilization for wget and for a varying number of containers. Indeed, although there is a noticeable change in the experiment completion time for approximately 400 containers, the rest of the resources appear to be nominal and certainly within their installed capacity. The completion time for each container should be approximately 150 seconds. It is not until we reach 600 concurrent VEEs that we notice that the network throughput reaches a maximal point and the completion time is well above 200 seconds, almost a 25% increase over the value for 400 VEEs. Therefore, we are unable to estimate the number of concurrent VEEs that the host can support by just observing the aggregate measurements of the utilization of the primary host resources. Also, while for wget the increase in completion time might appear satisfactory or even desirable since there is an increase in network throughput, for latency-sensitive applications it might produce erroneous results.

Interpreting the VEE scalability results as a deficiency of our measuring methodology, we decided to observe the experiment more closely and decide if there is a resource scarcity not based on aggregates but rather peak values of individual resources in any of the VEEs.

4 Heisenmeasure Uncertainty

Convinced that the context switching between containers and short term peaks in CPU and network utilization was the cause of the increase of the overall experimental time, we decided to increase the frequency of resource sampling and to keep individual and not aggregate measurements for each containers. To that end, our monitoring process would open the individual proc structures for each of the containers and traverse them at regular intervals every 0.1, 0.01, 0.005 and 0.001 seconds. Since applications in our experiment can be affected by a lot of different resources, we measured all host resources for completeness. We stored the individual measurements in memory and we performed statistical analysis after the completion of the experiment. As a side-note, it is worth mentioning that it is imperative to assign a real-time scheduling policy to the monitoring process. Otherwise, it would fail to run on time producing erroneous results.

We performed the same experiments as before but with different number of containers, from 100 to 700. We also reduced the initial calling interval of wget to one (1) to ten (10) seconds. For every number of containers, the same experiment is done with different sampling intervals (0.1, 0.01, 0.005 and 0.001 seconds). We were aiming in identifying utilization peaks within a small window of time and estimate when the experimental results become corrupted due to lack of readily available resources. Therefore, by carefully monitoring the health of each VEE we should be able to detect “anomalies” in resource consumption and potentially react in a dynamic fashion. However, our results show frequent, or measurement collected on a small window of time, can adversely affect the available resources for the application under test. In essence, we observed what we call a “weak” form of the Heisenberg uncertainty principle for host resource measurements: increasing the precision and fidelity of the resource measurements can interfere with the behavior of the experiment.

The completion times for different values of measurement frequencies (0.1-0.001 seconds) is shown in Figure 3. Each of the graph lines corresponds to a different sampling intervals. Frequently polling the resource values for each VEE can interfere with the performance of the experiment and increase its completion time. The more frequent the measurements, the smaller the number

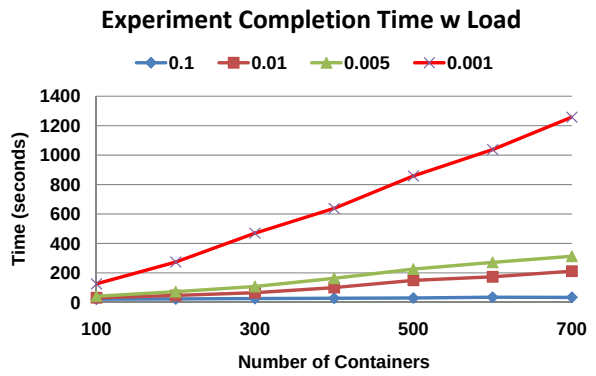

Figure 3: Completion times when we vary and the number of containers. Each line corresponds to different sampling intervals (0.1-0.001 seconds). Measuring the resource utilization for the VEEs can affect the experimental completion time. The more frequent the measurements, the less containers we can instantiate without interfering with the experiment.

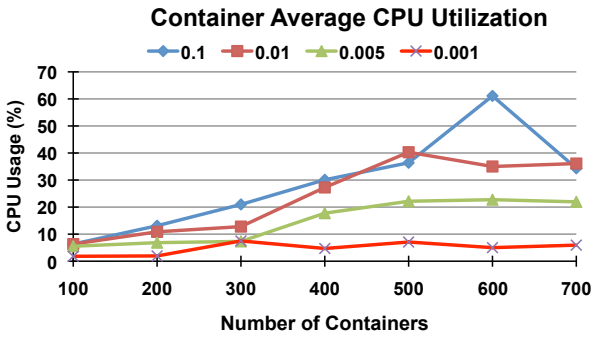

Figure 4: CPU utilization for each container and for different measuring windows (0.1-0.001 seconds). Merely examining this graph we are unable to determine accurately when the experiment is affected: there is a linear CPU increase even for 500 containers and for 0.01 second interval whereas the completion time shows a significant difference.

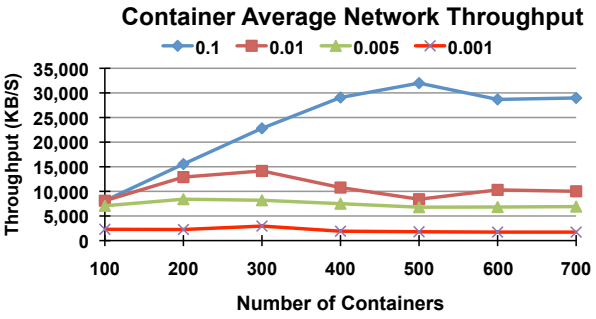

Figure 5: Network utilization clearly indicates the number of VEEs that can be supported when sampling frequency can affect the health of the overall experiment.

of containers we can instantiate without affecting the overall experiment. Observing the measurement of the CPU in Figure 4 is not helpful in determining the "breaking" point for each polling interval. Indeed, we are unable to determine accurately when the experiment is affected just by looking at Figure 4: there is a linear CPU increase even for 500 containers and for 0.01 second interval whereas there is a significant discrepancy in the experiment completion time. Unlike CPU, measuring the network utilization is a good measure of when the experiment is affected by our measurement. In Figure 5, we can see that the number of containers that can be supported for each of the polling intervals is pronounced: 0.1 seconds can support up to 500 containers, 0.01 can sustain up to 300 containers, and 0.001 cannot go beyond 100 containers.

Although our observation holds in principle for any application running in a host, it becomes more pronounced for a system running hundreds of VEEs concurrently. We believe that this is due to the necessary context switching among the different CPU tasks and it imposes a fundamental limitation on the number of VEEs that we can instantiate and use for testing without interfering with the fidelity of the application. The use of an event-driven VEE scheduling for each of the resources including CPU, I/O, and networking would alleviate this allowing us to scale without losing fidelity both in terms of resources and in terms of measuring the health of the experiment.

5 Related Work

Computer virtualization was originally introduced as a product by IBM in 1975 [3]. In recent years it has been revitalized mainly due to the technology introduced by companies and the open source community including VMware [17], Xen [4] , User Mode Linux [7], KVM [1] and VirtualBox [21]. A virtualization approach that provides more efficiency is the OS-level virtualization. This approach has been implemented in several operating systems such as BSD[11], Solaris[14, 12] and Linux [15, 16]. Virtualization has been employed in the past for building network emulation system [10, 9] and provide live migration capabilities [20].

Traditional approach to testing large-scale and multi-tiered network topologies and applications is currently done through simulation models that cannot capture all aspects of system behavior, especially scalability properties and failures.Therefore, simulation cannot substitute real system experimentation and fails to capture software bugs and configuration errors. In addition, the transient state of applications and unexpected component interactions remain unexplored. On the other hand, emulation techniques including Emulab [22, 10], and ModelNet [19]) offer contained execution environments that use unmodified applications and operating systems. Unfortunately,

the scalability of the current testbeds is limited by the number of physical nodes. We cannot expect to see a dramatic increase of the aforementioned testbed to tens or even hundreds of thousands of nodes. For example, emulating 10000 instances of a botnet or a virus propagation using 250 physical machines is currently beyond the capabilities of the most advanced testbeds.

There is a wealth of previous research on dynamic provision of services [8, 18] or to efficient load balancing[6]. However, these efforts typically target already running services and they affect the fidelity of the experiment. This is due to the use of full virtualization which does not scale as efficiently as process containers and does not expose the entire stack of driver code to the application. Moreover, they do not offer a solution to resource utilization peaks but rather to initial provision of services [8].

6 Conclusions

In this paper, we analyze the experimental scalability and fidelity limitations when employing lightweight virtualization as testbed environment to measure the performance of simple, large-scale applications. To that end, we show that it is not a trivial task to determine the maximum number of VEEs that can be run concurrently in a physical machine without perturbing the experimental outcome.

Furthermore, our efforts to monitor the health of each VEE have uncovered a "weak" form of Heisenberg uncertainty principle in measuring lightweight virtualization. The desirable accuracy of measurement is largely dependent on the high sampling frequency, which potentially deprives the containers of available resources and adversely interferes with the experiment.

References

[1] **Kvm: Kernel-based virtual machine system for linux (http://kvm.sourceforge.net/).**

[2] **Linux kernel internals. page 18, Seattle, WA, USA. Specialized Systems Consultants, Inc.**

[3] **J. D. Bagley, E. R. Floto, S. C. Hsieh, and V. Watson. Sharing data and services in a virtual machine system. In *Proceedings of SOSP 1975*, pages 82–88, New York, NY, USA, 1975. ACM.**

[4] **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 SOSP 2003*, pages 164–177, New York, NY, USA, 2003. ACM.**

[5] **T. Benzel, R. Braden, D. Kim, C. Neuman, A. D. Joseph, and K. Sklower. Experience with deter: A testbed for security research. In *TRIDENTCOM*, 2006.**

[6] **J. M. Blanquer, A. Batchelli, K. Schauser, and R. Wolski. Quorum: flexible quality of service for internet services. In *Proceedings of NSDI 2005*, pages 159–174, Berkeley, CA, USA, 2005.**

[7] **J. Dike. A user-mode port of the linux kernel. In *In Proceedings of the 5th Annual Linux Showcase and Conference*, Oakland, California, Nov 2001.**

[8] **R. P. Doyle, J. S. Chase, O. M. Asad, W. Jin, and A. M. Vahdat. Model-based resource provisioning in a web service utility. In *Proceedings of Usenix USITS 2003*, pages 5–5, Berkeley, CA, USA, 2003.**

[9] **D. Gupta, K. V. Vishwanath, and A. Vahdat. Diecast: testing distributed systems with an accurate scale model. In *Proceedings of Usenix NSDI 2008*, pages 407–422, Berkeley, CA, USA, 2008.**

[10] **M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau. Large-scale virtualization in the emulab network testbed. In *ATC'08: USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 113–128, Berkeley, CA, USA, 2008.**

[11] **P. Hope. Using Jails in FreeBSD for fun and profit. volume 27, pages 48–55, 2002.**

[12] **M. Lageman and S. Solutions. Solaris Containers What They Are and How to Use Them. pages 819–2679, 2005.**

[13] **P. Movall, W. Nelson, and S. Wetzstein. Linux physical memory analysis. In *Proceedings of ATC Usenix ATC 2005*, pages 39–39, Berkeley, CA, USA, 2005.**

[14] **D. Price and A. Tucker. Solaris zones: Operating system support for consolidating commercial workloads. In *Proceedings of LISA 2004*, 2004.**

[15] **A. Shoykhet, A. Shoykhet, J. Lange, J. Lange, P. Dinda, and P. Dinda. Virtuoso: A system for virtual machine marketplaces. Technical report, 2004.**

[16] **S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *EuroSys '07*, pages 275–287, New York, NY, USA, 2007. ACM.**

[17] **J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of USENIX ATC 2005*, pages 1–14, Berkeley, CA, USA, 2001.**

[18] **B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *Proceedings of OSDI 2002*, pages 239–254, New York, NY, USA, 2002. ACM.**

[19] **A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proceedings of OSDI 2002*, pages 271–284, New York, NY, USA, 2002. ACM.**

[20] **Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford. Virtual routers on the move: live router migration as a network-management primitive. In *Proceedings of SIGCOMM '08*, pages 231–242, New York, NY, USA, 2008. ACM.**

[21] **J. Watson. Virtualbox: bits and bytes masquerading as machines. In *Linux J.*, volume 2008, page 1, Seattle, WA, USA, 2008.**

[22] **B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proceedings of OSDI '02*, pages 255–270, New York, NY, USA, 2002. ACM.**

[23] **C. P. Wright and E. Zadok. Kernel korner: unionfs: bringing filesystems together. In *Linux J.*, volume 2004, page 8, Seattle, WA, USA, 2004.**