



Performance Evaluation of Virtualization Technologies for Server Consolidation

Pradeep Padala, Xiaoyun Zhu, Zhikui Wang, Sharad Singhal, Kang G. Shin
HP Laboratories
HPL-2007-59R1

Keyword(s):

server consolidation, virtualization, multi-tier application, performance, overhead

Abstract:

Server consolidation has become an integral part of IT planning to reduce cost and improve efficiency in today's enterprise datacenters. The advent of resource virtualization allows consolidation of multiple applications into virtual servers hosted on a single or multiple physical servers. However, despite its salient features, this poses new challenges, including selection of the right virtualization technology and consolidation configuration for a particular set of applications. In this paper, we evaluate two representative virtualization technologies, Xen and OpenVZ, in various configurations. We consolidate one or more multi-tiered systems onto one or two nodes and drive the system with an auction workload called RUBiS. We compare both technologies with a base system in terms of application performance, resource consumption, scalability, low-level system metrics like cache misses, and virtualization-specific metrics like Domain-0 consumption in Xen. Our experiments indicate that the average response time can increase by more than 400% in Xen and by a more modest 100% in OpenVZ as the number of application instances grows from one to four. This large discrepancy is found to come from the higher virtualization overhead in Xen, which is likely caused by higher L2 cache misses and larger number of misses per instruction. A similar trend is observed in the CPU consumptions of the virtual servers. We analyze the overhead with kernel-symbol-specific information generated by Oprofile and suggest possible remedies for these problems.

External Posting Date: September 30, 2008 [Fulltext] - Approved for External Publication

Internal Posting Date: September 30, 2008 [Fulltext]

© Copyright 2008 Hewlett-Packard Development Company, L.P.



Performance Evaluation of Virtualization Technologies for Server Consolidation

Pradeep Padala¹, Xiaoyun Zhu², Zhikui Wang², Sharad Singhal², and Kang G. Shin¹

¹University of Michigan

²Hewlett Packard Laboratories

Abstract

Server consolidation has become an integral part of IT planning to reduce cost and improve efficiency in today’s enterprise datacenters. The advent of resource virtualization allows consolidation of multiple applications into virtual servers hosted on a single or multiple physical servers. However, despite its salient features, this poses new challenges, including selection of the right virtualization technology and consolidation configuration for a particular set of applications. In this paper, we evaluate two representative virtualization technologies, Xen and OpenVZ, in various configurations. We consolidate one or more multi-tiered systems onto one or two nodes and drive the system with an auction workload called RUBiS. We compare both technologies with a base system in terms of application performance, resource consumption, scalability, low-level system metrics like cache misses, and virtualization-specific metrics like Domain-0 consumption in Xen. Our experiments indicate that the average response time can increase by more than 400% in Xen and by a more modest 100% in OpenVZ as the number of application instances grows from one to four. This large discrepancy is found to come from the higher virtualization overhead in Xen, which is likely caused by higher L2 cache misses and larger number of misses per instruction. A similar trend is observed in the CPU consumptions of the virtual servers. We analyze the overhead with kernel-symbol-specific information generated by Oprofile and suggest possible remedies for these problems.

1. INTRODUCTION

In the last several years, server consolidation has been a common practice in enterprise datacenters as a means to achieve cost savings in hardware, software, floor space, power and cooling, as well as server administration. As a key technology enabler, server virtualization allows multiple “virtual servers” to be created on each physical machine sharing the server resources. This way, many enterprise applications that traditionally run on dedicated servers can be consolidated onto a smaller

and shared pool of servers. Although server consolidation offers great potential to increase resource utilization, centralize administration, and improve operational efficiency and flexibility, it also introduces new challenges in managing the applications running in the virtual servers in a consolidated environment.

In particular, capacity planning and workload management for the virtual servers are non-trivial tasks for system administrators. The common approach of using resource-utilization traces collected from the pre-consolidated servers as a predictor for the resource demands of the virtual servers running the same applications can be inaccurate and problematic due to the virtualization overhead. Moreover, today’s enterprise applications typically employ a multi-tiered architecture, where the Web and the application tiers serve static files and implement business logic, and the database tier executes data queries and interacts with the storage devices. During server consolidation, each tier of an application can be moved from a physical server into a virtual server, and the various tiers of the same application may be hosted on the same physical node or across different nodes. A key question is then how application-level performance, such as transaction response time and throughput, will be impacted. There are two potential causes for performance degradation, if any — *consolidation*, as more application tiers run on a single physical server; and *virtualization*, for there may be additional layers between an application tier and the physical hardware besides the operating system.

In this paper, we focus on the performance degradation caused by virtualization on Linux-based systems in a consolidated environment. We refer to the difference in application performance between a virtualized system and a native Linux system under the same configuration as the *performance overhead* by the underlying virtualization technology. Such performance overhead is likely dependent upon many factors, including the type of virtualization used. In general, there are two main types of virtualization technologies today — *hypervisor-based virtualization* including VMware ESX Server [1], Microsoft Virtual Server [2], and Xen [3];

and *OS-level virtualization* including OpenVZ [4], Linux VServer [5], and Solaris Zones (or Containers) [6]. Determining which virtualization technology to use for server consolidation requires a careful evaluation of the tradeoff between the pros and cons of different technologies, because no single technology will excel at everything. For each technology chosen, the datacenter administrators need to determine how many applications can be hosted on a given number of physical servers, and for each application, what is the maximum load it can take without violating its service-level objectives (SLOs). Moreover, how to place the various tiers of the multi-tiered applications on the shared servers in order to minimize potential performance overhead?

We will address these issues using RUBiS [7] as an example of a multi-tiered application in a case study. We focus on Xen and OpenVZ as the representatives for the hypervisor-based and OS-level virtualization technologies, respectively. It is to be expected that OS-level virtualization incurs lower performance overhead whereas hypervisor-based virtualization provides better fault-isolation between the virtual servers [8]. However, to the best of our knowledge, there is little published work quantifying the difference in performance overheads between the two technologies, especially for consolidation of multi-tiered applications and in terms of application-level metrics, such as transaction response time. In this paper, we present the experimental results that aim to evaluate the performance overhead incurred by Xen or OpenVZ under different consolidation configurations and workload conditions. Our experiments indicate that, in spite of their comparable throughput, the average response times of the consolidated applications on a Xen system and an OpenVZ system can be significantly different, especially as the workload intensity or the scale of consolidation increases. For example, we observe that the average response time increases by over 400% in Xen and by a more modest 100% in OpenVZ as the number of consolidated applications grows from one to four.

In addition, using Oprofile [9], a system profiler for Linux, we report the values of a few selected hardware performance counters and kernel-symbol-specific information, and infer that the higher performance overhead in Xen is likely caused by higher L2 cache misses in the virtual servers. And in most cases, the majority of these cache misses occur in the `vmlinux` kernels in Dom0 and the guest domains. Finally, based on the analysis, we suggest a number of potential ways to reduce the overhead so that we can take better advantage of the virtualization technologies for server consolidation.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 introduces the architecture of our testbed and the various tools used. The details of the experiments are described in

Section 4, and the results along with our analysis are presented in Sections 5 and 6. Section 7 presents a plausible interpretation of the performance difference between the two virtualization technologies. Finally, we summarize our key findings and discuss future work in Section 8.

2. RELATED WORK

There is rich literature on the Xen virtualization system. A performance evaluation of Xen was provided in the first SOSP paper on Xen [3] using SPEC CPU2000, OSDB, dbench and SPECWeb benchmarks. The performance of Xen was compared to VMWare, native Linux and User-mode Linux in terms of the total system throughput. The results have been re-produced by a separate group of researchers [10]. In this paper, we extend this evaluation to include OpenVZ as another virtualization platform, and test both Xen and OpenVZ under different scenarios including multiple VMs and multi-tiered systems. We also take a deeper look into some of these scenarios using OProfile [9] to provide insights into the possible causes of the performance overhead observed.

Menon *et al.* conducted a similar performance evaluation of the Xen environment and found various overheads in the networking stack [11]. The work provides an invaluable performance analysis tool Xenoprof that allows detailed analysis of a Xen system. The authors identified the specific kernel subsystems that were causing the overheads. We perform a similar analysis at a macro level and apply it to different configurations specifically in the context of server consolidation. We also investigate the differences between OpenVZ and Xen specifically related to performance overheads.

Menon *et al.* used the information gathered in the above work and investigated causes of the network overhead in Xen [12]. They proposed three techniques for optimizing network virtualization in Xen. We believe that our work can help develop similar optimizations that help server consolidation in both OpenVZ and Xen.

Gupta *et al.* have studied the performance isolation provided by Xen [13]. In their work, they developed a set of primitives to address the problem of proper accounting of work done in device drivers by a particular domain. Similar to our work, they used XenMon [14] to detect performance anomalies. Our work is orthogonal to theirs by providing insights into performance overhead observed in Xen or OpenVZ systems in a consolidated environment.

There has also been published work on OS-level (or *container-based*) virtualization. Osman *et al.* have developed Zap, a container-based virtualization system that allows easy migration of applications running in pods (process domains) [15]. The primary focus of that paper was on migration of virtual containers, which differs from the focus of our paper. In the paper, the au-

thors also compared Zap to VMware Workstation 3.2 that provides hardware emulation. We believe that hypervisor-based virtualization is fundamentally different from hardware emulation.

Soltész *et al.* have developed Linux VServer, which is another implementation of OS-level virtualization on Linux [5]. They have done a comparison study between VServer and Xen in terms of performance and isolation capabilities [8]. In particular, average throughput was used as the metric for comparison of application-level performance between VServer and Xen. In this paper, we compare performance overheads between OpenVZ and Xen instead, and demonstrate that, even when throughput is comparable between two virtualization technologies, the difference in response time can be fairly significant. This needs to be taken into account in server consolidation, especially when there are SLOs defined in terms of transaction response times.

In all the prior work mentioned above, there has been no specific evaluation of virtualization technologies in the context of server consolidation for multi-tiered applications. In particular, performance overheads as a result of virtualization in a consolidated environment have not been quantified. In this paper, we comprehensively evaluate two representative virtualization technologies in terms of their performance overheads relative to a base Linux system and demonstrate how the overheads may change in different consolidation scenarios.

3. TESTBED ARCHITECTURE

Figure 1(a) shows the architecture of our testbed, where a Xen-based system, an OpenVZ-based system, and a vanilla Linux system (referred to as *base system* hereafter) with an identical hardware setup are used for our experimentation. We ran the same application(s) on the three systems under various configuration scenarios. In each scenario, each physical node may host one or more virtual servers supported by Xen or OpenVZ as shown in Figure 1(b). And two separate nodes may be used for the Web and the database tiers of one or more multi-tier applications. Each node is equipped with a sensor collecting various system-level metrics including CPU consumption, memory consumption as well as hardware-level events. And each application has a sensor that measures its application-level performance including throughput and response times. All of this data is transferred to a separate machine for performance analysis later.

We use HP Proliant DL385 G1 for all our servers and client machines. Every server has two 2.6 GHz processors, each with 1MB of L2 cache, 8 GB of RAM, and 2 Gigabit network interfaces.

3.1 System configurations

We conduct our experiments on three different sys-

tems as explained below. All systems are set up to be as similar as possible with the same amount of resources (memory and CPU) allocated to a particular virtual server. A complete description of all system configurations is available at <http://www.eecs.umich.edu/~ppadala/research/perfeval/config.html>

3.1.1 The base system

We use a plain vanilla 2.6 Linux kernel that comes with the Fedora Core 5 standard distribution as our base system. Standard packages available from Fedora repository are used to set up various applications.

3.1.2 Xen system

Xen is a *paravirtualization* [16] technology for x86 that allows multiple guest OSes to be run in virtual servers (called *domains*). The Xen hypervisor provides a thin software virtualization layer between the guest OS and the underlying hardware. Each guest OS is a modified version of the base Linux (*XenoLinux*) because the hardware abstraction presented by the hypervisor is similar but not identical to the raw hardware. The hypervisor contains a CPU scheduler (see Figure 1(b)) that implements various scheduling policies including proportional fair-share, along with other modules such as the memory management unit.

The Xen 3.0.3 unstable branch [17] is used for our experiments as it provides a credit-based CPU scheduler (in short, credit scheduler), which, in our experiments, provides better performance than the earlier SEDF scheduler. The credit scheduler allows each domain to be assigned a *cap* and a *weight*. A non-zero cap implements a non-work-conserving policy for the CPU by specifying the maximum share of CPU time a domain can consume, even if there are idle CPU cycles available. When the cap is zero, the scheduler switches to a work-conserving mode, where weights for multiple domains determine their relative shares of CPU time when the CPU is under contention. At the same time, a domain can use extra CPU time beyond its share if other domains do not need it. In all our experiments, we use the *non-capped* mode of the credit scheduler, and the system is compiled using the uni-processor architecture. In this case, Dom0 and all the guest domains share the full capacity of a single processor.

3.1.3 OpenVZ system

OpenVZ [4] is a Linux-based OS-level server virtualization technology. It is the open source basis of Virtuozzo, a commercial virtualization tool from SWsoft [18]. It allows creation of multiple secure, isolated virtual environments (VEs) on a single node enabling server consolidation. Each VE behaves exactly like a stand-alone server. They can be rebooted independently and a different distribution with separate root directory can be

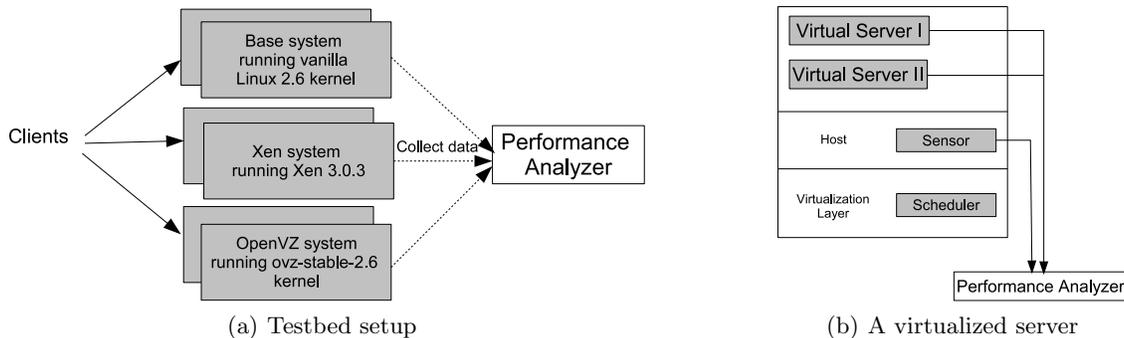


Figure 1: System architecture

set up. Different from Xen, OpenVZ uses a single kernel shared by all the VEs. Therefore, it does not provide the same level of fault-isolation as well as customization that Xen offers.

In our experiments, we use the stable, version 2.6, uni-processor OpenVZ kernel that provides an FSS scheduler, which also allows the CPU share of each VE to be either capped or not capped. As in the Xen system, the non-capped option of the scheduler is used in the OpenVZ system.

We will henceforth use the term *virtual server* to refer to either a domain in the Xen system or a VE in the OpenVZ system.

3.2 Instrumentation

To measure the CPU consumption accurately, we wrote scripts that use existing tools to collect data. In the base system, the output from command `top -b` is gathered and then analyzed later. Similarly, `xentop -b` is used in the Xen system, which provides information on the CPU consumptions of individual domains. For OpenVZ, there is no existing tool to directly measure the CPU consumption by a particular VE. We use the data provided from `/proc/vz/vestat` to obtain the CPU time spent by individual VEs.

3.2.1 Oprofile

Oprofile [9] is a tool for measuring certain hardware events using hardware performance counters. For example, one can measure the number of cache misses that happen in a particular application. The profiles generated by Oprofile are very detailed and provide a wealth of information. Menon *et al.* [11] have modified Oprofile to support Xen. The resulting tool, *Xenoprof*, allows us to profile multiple domains in a Xen system.

We focus on the following two aspects when analyzing the data generated by Oprofile:

- Comparing hardware performance counters for various configurations;
- Understanding differences in overheads experienced

within specific kernels. We want to identify particular kernel sub-systems where most of the overheads occur, and quantify the overheads using related hardware counter values.

We monitor three hardware counters for our analysis:

- **CPU_CLK_UNHALT**: The number of cycles outside of halt state. It provides a rough estimate of the CPU time used by a particular binary or a symbol.
- **RETIRED_INSTRUCTIONS**: The number of instructions that are retired. It is a rough estimate of the number of instructions executed by a binary or a symbol.
- **L2_CACHE_MISS**: The number of L2 cache misses. It measures the number of times the memory references in an instruction miss the L2 cache and thus access main memory.

For each set of experiments, we analyze differences in these counter values along with the observed application performance in various configurations, and infer possible sources of the performance overhead observed in the corresponding virtualization technology.

4. EXPERIMENT DESIGN

The experiments are designed with the goal of quantitatively evaluating the impact of virtualization on server consolidation. Specifically, we are not interested in performing micro benchmarks that compare the performance of system calls, page miss penalties, etc. Instead, we focus more on typical multi-tiered applications and how application-level performance, particularly transaction response time, is affected when the consolidated servers are virtualized using Xen or OpenVZ.

We have chosen RUBiS [7], an online auction site benchmark, as an example of multi-tiered applications. We use a version of the application that has a two-tiered structure: the Web tier contains an Apache Web server with PHP, and the DB tier uses a MySQL database server. A workload generator is used to emulate a specified number of RUBiS clients who connect to the Web

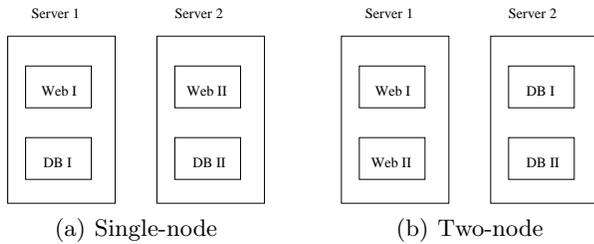


Figure 2: Two placement configurations for consolidating two multi-tiered applications

tier and perform various operations. More specifically, each client starts a session in which the client browses through items, looks at prices, and buys or sells items. In each session, the client waits for a request to complete before sending out the next request. If the request fails due to time-outs, the session is aborted and a new session is started. This gives rise to a closed-loop behavior where the clients wait for the server when it is overloaded. RUBiS provides different workload mixes, and we use the *browsing mix* in our experiments, which introduces a higher resource demand on the Web tier than on the DB tier.

We consider the scenario where one or more instances of RUBiS are running on virtualized, consolidated servers, and design a series of experiments to evaluate the performance overhead incurred by Xen or OpenVZ. In particular, we observe how the overhead changes as we vary the following three parameters:

- **Workload intensity** - the number of threads in the RUBiS workload emulating the number of concurrent clients accessing the same application;
- **Scale of consolidation** - the number of applications consolidated onto a given number of servers;
- **Placement configuration** - the way multiple tiers of an application are placed on the consolidated servers;

With respect to placement configuration, we compare the following two options for hosting two RUBiS applications on two physical nodes:

- *Single-node*: Both the Web and the DB tiers of each RUBiS application are hosted on a single physical node (Figure 2(a));
- *Two-node*: The Web and the DB tiers of each RUBiS application are distributed on two separate nodes (Figure 2(b)).

There are additional reasons why one configuration may be chosen over the other in a practical scenario. For example, the single-node configuration offers the benefit of reduced network traffic by handling tier-to-tier communication of an application within a node, whereas the

two-node option may be preferable in a case where it can reduce software licensing cost. In this work, we focus on the difference in the observed performance overheads between the two configurations.

5. EVALUATION RESULTS

This section presents the results of our experimental evaluation using the RUBiS benchmark. In particular, we evaluate the application performance along three dimensions: workload intensity, scale of consolidation, and placement configuration. Each experiment is continuously run for 15 minutes, and both application- and system-level metrics are collected. For each dimension, a three-way comparison of the results from the base Linux, OpenVZ, and Xen systems is presented.

5.1 Impact of workload intensity

We start with the single-node configuration and evaluate how the performance overhead changes with the intensity of the workload coming into the application. In this case, both the Web and the DB tiers are hosted on the same node, as shown in Figure 2(a). For each of the base, Xen, and OpenVZ systems, we scale up the workload by increasing the number of concurrent threads in the RUBiS client from 500 to 800.

Figures 3(a) and 3(b) show the throughput and average response time as a function of workload intensity for the three systems. In all three cases the throughput increases linearly as the number of threads increases, and there is little difference among the three systems. However, we observed a marked difference in the response time between the Xen system and the other two systems. As the workload intensity increases from 500 to 800 threads, the response time increases only slightly in both the base and OpenVZ systems, whereas in the Xen case, it grows from 18 ms to 130 ms, an increase of over 600%. For 800 threads, the response time for Xen is approximately 4 times that for OpenVZ. Therefore, in the single-node case, the observed performance overhead is higher in Xen than in OpenVZ, which as expected. However, the difference between the two goes up significantly as the workload increases. As a result, the Xen system is less scalable with the workload intensity than OpenVZ or a non-virtualized system.

Figure 4 shows the average CPU consumptions of the Web and the DB tiers as a function of workload in the three systems. For both tiers in all the three cases, the CPU consumption increases linearly with the number of threads in the workload. The DB tier’s consumption remains very low at about 1-4% of total CPU capacity in all the cases, due to the Web-intensive nature of the browsing mix workload. A bigger difference can be seen in the Web tier’s consumption from the three systems. For each workload, the Web tier’s consumption in Xen is roughly twice the consumption by the

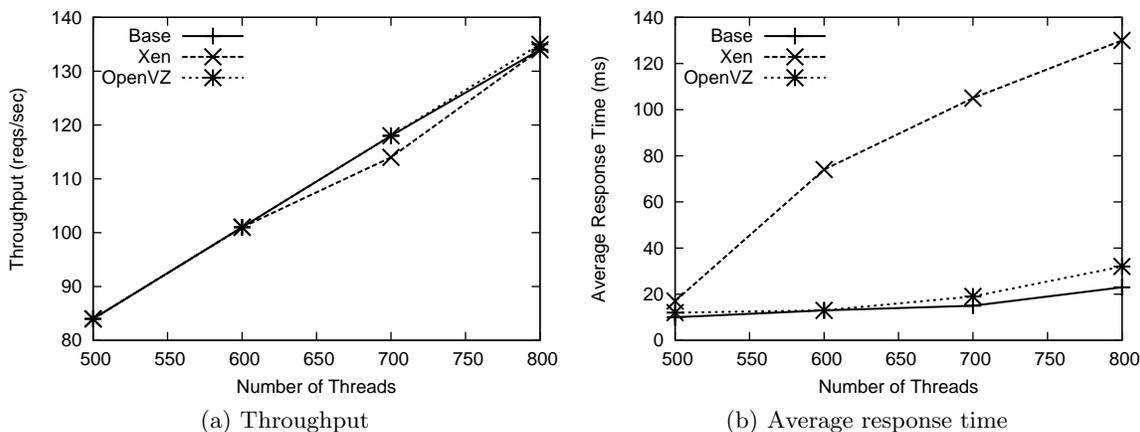


Figure 3: Single-node configuration - application performance as a function of workload intensity

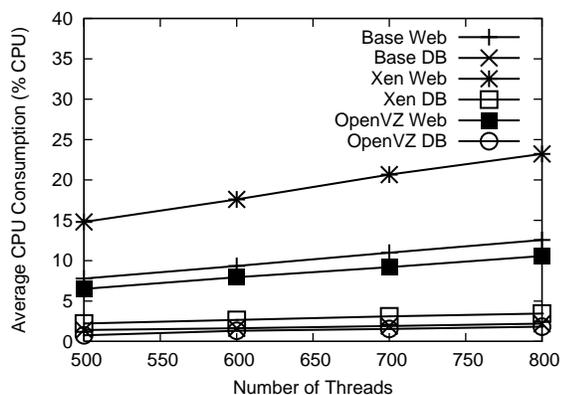


Figure 4: Single-node configuration — average CPU consumption per tier as a function of workload intensity

base system, whereas the OpenVZ’s consumption stays close to the base case. As the workload increases, the slope of increase is higher in the case of Xen than the other two systems. More specifically, 100 more concurrent threads consume roughly 3% more CPU capacity for Xen and only 1% more for the other two systems. Since the throughput is comparable among the three systems, this indicates higher CPU overhead per unit of workload in Xen than in OpenVZ, and should be related to the higher response times observed in the Xen case.

5.2 Evaluation of scalability

We now consider the two-node configuration shown in Figure 2(b) and investigate the scale of consolidation that can be achieved by the two virtualization technologies. We increase the number of RUBiS instances sharing the two physical nodes from one to two, then to four. For example, in the case of four instances, one

node is used to host four instances of the Web tier and the other node is used to host four instances of the DB tier. We compare the scalability of Xen and OpenVZ with respect to application performance and resource consumption.

We omit figures for application throughput. Even as the number of RUBiS instances is increased to four, we still observe a linear increase in the throughput as a function of workload, and approximately the same throughput from both the OpenVZ and the Xen systems.

Figures 5(a), 5(b), and 5(c) show the average response time as a function of workload when running one, two, or four instances of RUBiS on two nodes, respectively. In the cases where there are more than one instances of the application, multiple curves are shown in the figure for either Xen or OpenVZ, where each curve represents one of the multiple application instances. For all the cases tested, the average response time for OpenVZ varies at most by 10 ms and remains near or below 30 ms at all times. Instead, the response time for Xen increases significantly along with the workload. For example, in the cases of one, two, and four application instances, as the workload increases from 500 to 800 threads, the average response time experienced by each application instance increases by 133% (from 12 ms to 28 ms), 500% (from 15 ms to 90 ms on average), and nearly 700% (from 20 ms to 160 ms on average), respectively.

Figure 6(a) compares the mean response times of Xen and OpenVZ averaged across all the application instances as the number of consolidated applications increases from one to four in the two-node configuration. In the Xen system, the average response time per application grows from 28 ms for one instance to 158 ms for four instances, an over 400% increase. In contrast, this increase is only about 100% in the OpenVZ case. This indicates much better scalability of OpenVZ with

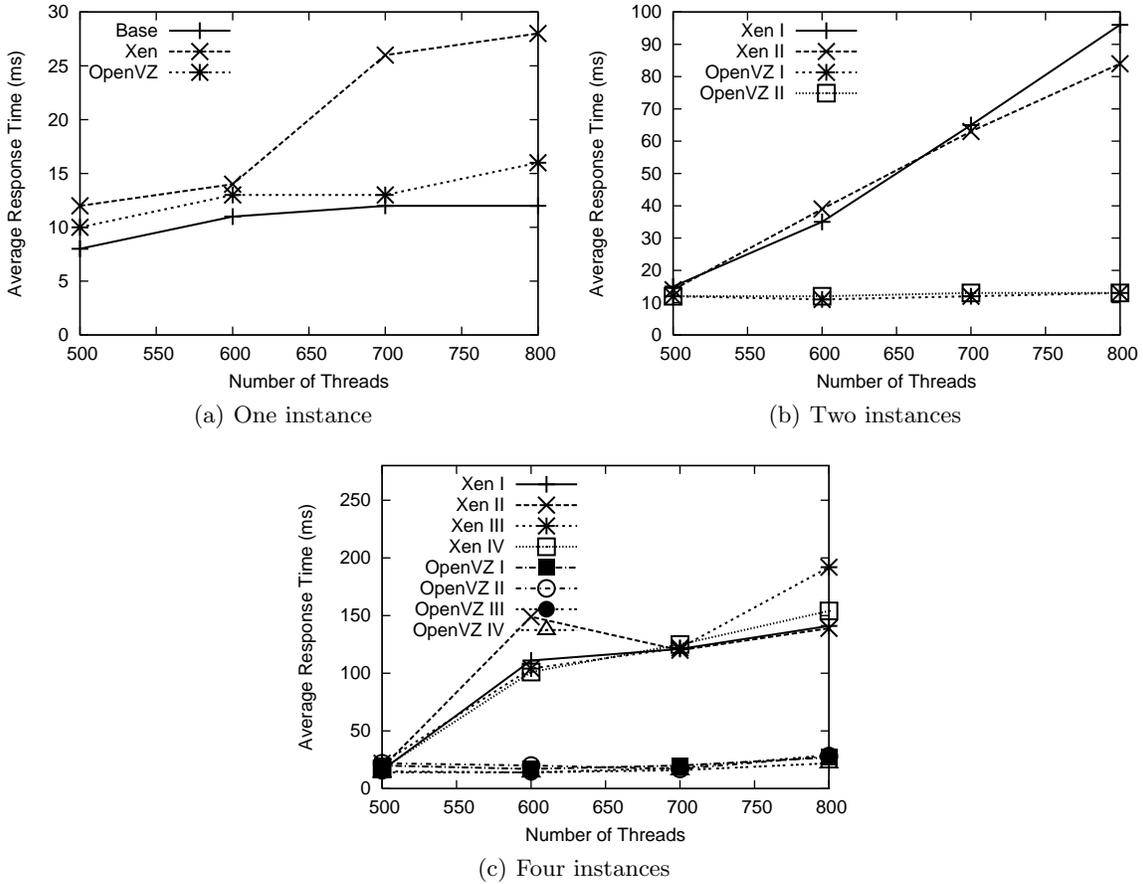


Figure 5: Two-node configuration — average response time as a function of workload intensity and nscale of consolidation

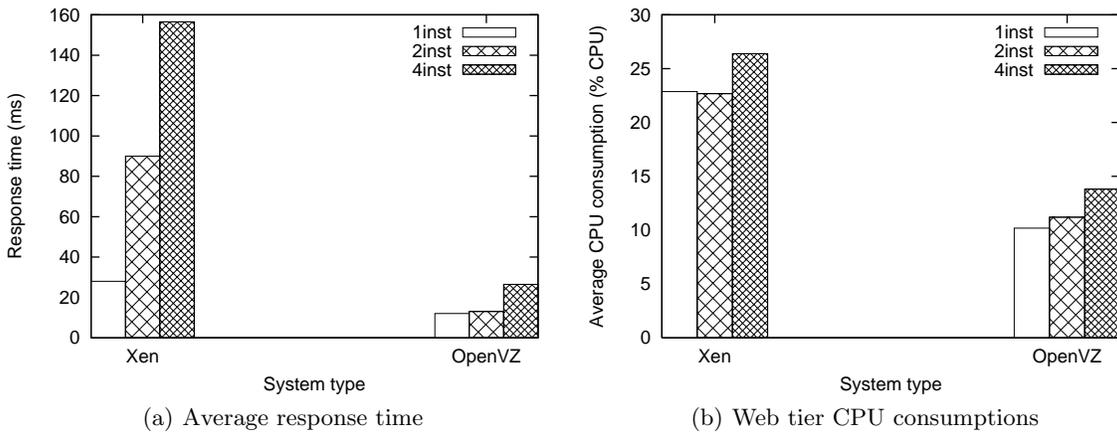


Figure 6: Two-node configuration — average response times and Web tier's CPU consumptions for different scales of consolidation (800 threads)

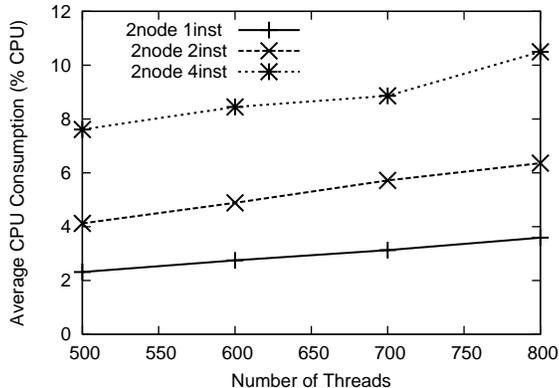


Figure 7: Two-node configuration — Xen Dom0 CPU consumption as a function of workload intensity and number of consolidated applications

respect to application-level performance.

Figure 6(b) compares the average Web tier’s CPU consumptions seen by all the application instances in the Xen and the OpenVZ systems. We can see that the average consumption per application instance for Xen is roughly twice that for OpenVZ. Moreover, with four instances of RUBiS, the Xen system is already becoming overloaded (with the sum of all four instances exceeding 100%), whereas the OpenVZ system has the total consumption below 60% and should be able to accommodate at least two more instances of the RUBiS application without sacrificing the throughput of each application instance.

Figure 7 shows the Xen Dom0 CPU consumption as a function of workload in the two-node case. The different lines in the graph correspond to different numbers of RUBiS instances hosted. For each scenario, the Dom0 consumption increases approximately linearly as the workload increases from 500 to 800 threads. This was expected because Dom0 handles I/O operations on behalf of the guest domains causing its consumption to scale linearly with the workload. Interestingly, as the number of instances grows from one to four, the Dom0 consumption increases by a factor of 3 instead of 4, showing a certain degree of multiplexing in Dom0.

To investigate how much more overhead can be incurred in the Xen system, we ran the four instances on two CPUs running the SMP kernel. We observed that the average CPU consumption by the Web tiers is 25% and the average Dom0 consumption is 14%. The latter is higher than that obtained using the UP kernel.

5.3 Impact of placement configuration

In this subsection, we compare the two placement configurations for hosting two RUBiS applications using two physical nodes. In the single-node configuration (Figure 2(a)), each node is used to host both the Web

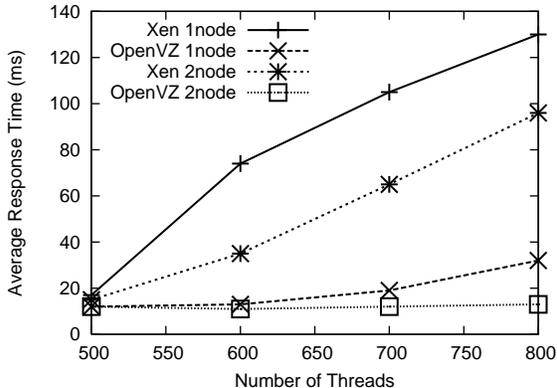


Figure 8: Two-node vs. single-node — average response time as a function of workload intensity

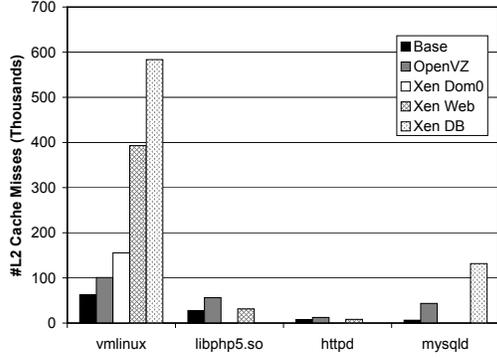
tier and the DB tier of an application. In the two-node configuration (Figure 2(b)), one node is used to host the two Web tiers from both applications, and the other node is used to host the two DB tiers. The goal of this study is to evaluate the impact of placement configuration on the application-level performance.

We plotted the observed application performance in both the single-node (Figure 3(b)) as well as the two-node (Figure 5(b)) configurations in the previous two subsections. Figure 8 shows a comparison of placement configurations in terms of the average response time seen by the two application instances as a function of the workload intensity in the Xen or the OpenVZ system. We observe that, when two nodes are used to host two multi-tiered RUBiS applications, the two-node configuration where each node hosts multiple instances of the same application tier provides lower average response times for the consolidated applications for both Xen and OpenVZ. In the case of Xen, for example, an SLO that indicates a maximum average response time of 100 ms for a workload up to 800 concurrent clients can be met by the two-node configuration but not by the single-node configuration. The Oprofile analysis shown in the next section helps uncover possible causes for this observed difference in the two placement configurations.

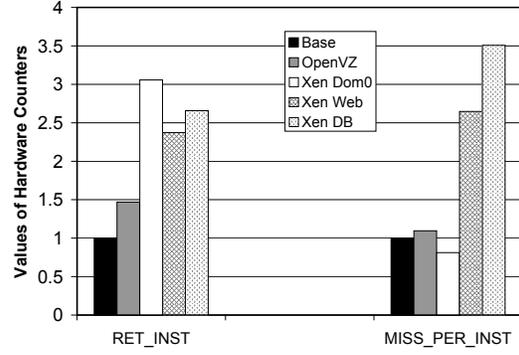
6. OPROFILE ANALYSIS

In this section, we consider some of the test scenarios from the previous section where interesting differences in performance overheads have been observed. All of these scenarios correspond to a workload intensity of 800 threads such that the observed difference is the largest. Here we provide a detailed analysis of the corresponding Oprofile statistics for each scenario and point to plausible causes for the observed performance overheads.

6.1 Single-node



(a) L2 cache misses break down



(b) Normalized counter values for vmlinux only

Figure 10: Single-node — counter values break down for three different systems

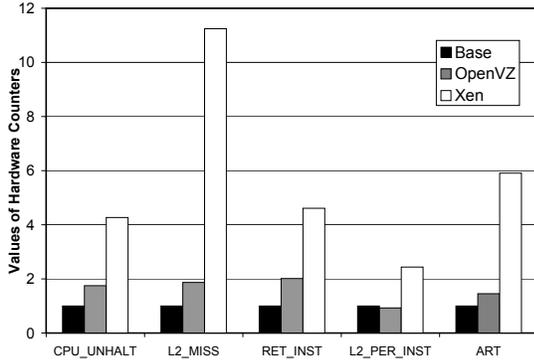


Figure 9: Single-node — normalized hardware counter values from Oprofile

In the single-node configuration, both the Web and the DB tiers of a RUBiS application are hosted on a single node. To understand the possible causes for the observed performance overheads in Xen and OpenVZ, we use the selected hardware counters described in Section 3 and compare their aggregate values for the three systems in Figure 9. For OpenVZ, each of the counter values is for the whole system including the shared kernel and the two virtual servers hosting the Web tier and the DB tier. For Xen, Oprofile provides us with one counter value for each domain, and we calculate the sum of the values for all the domains, including Dom0. We also calculate the number of L2 cache misses per instruction for each system. Finally, the observed average response times (ART) from the three systems are also included in the figure for easy comparison. In particular, the average response time observed from the Xen

system is almost six times that from the base system, whereas the average response time from the OpenVZ system is only slightly higher than that from the base system. All of the values shown are normalized with respect to the base case.

For OpenVZ, we observe twice as many instructions executed as that from the base system, and almost twice as many L2 cache misses as well. Therefore, the OpenVZ and the base systems have comparable numbers of L2 cache misses per instruction. On the other hand, relative to the base system, there are roughly 4.5 times as many instructions executed and more than 11 times as many L2 cache misses in the Xen system. Effectively, the number of L2 cache misses per instruction is more than twice as large in Xen as in OpenVZ or in base Linux. Therefore, we conjecture that L2 cache misses is the main cause of the observed difference in transaction response times between Xen and OpenVZ. Furthermore, the data suggests two reasons for the higher number of L2 cache misses in Xen: (i), there are many more instructions being executed in Xen compared to OpenVZ and base Linux, most likely due to the modified kernel for Dom0 and the two guest domains; (ii), the number of L2 cache misses per instruction is higher in Xen, possibly due to context switching between the two guest domains. Note that more cache misses per instruction causes the instructions to take longer to execute, which leads to higher latency at the application level.

Since the normalized values of the CPU_CLK_UNHALT counter and the RETIRED_INSTRUCTIONS counter are similar for the three systems, we infer that these three systems have average instructions per cycle values that are comparable. Therefore, in the subsequent figures, we omit the CPU_CLK_UNHALT values in our analysis and focus on the values of the other two hardware counters.

We also did a breakdown of the total number of L2 cache misses by different binaries (like `httpd`, `mysqld`) running in each system, and the result is shown in Figure 10(a). The dynamic library `libphp5.so` is responsible for executing the PHP scripts on the Apache side. The data clearly indicates that the majority of the L2 cache misses seen by the Xen system are from the `vmlinux` kernels running in Dom0 and the two guest domains rather than from the application. Moreover, the kernels in Xen experience more L2 cache misses than the kernel for base Linux or OpenVZ, and OpenVZ has slightly higher cache misses from the kernel than base Linux does. To see why this is the case, we further show in Figure 10(b) the retired instructions value and the number of L2 cache misses per instruction associated with `vmlinux` in all the cases. Both sets of values have been normalized with respect to the base value. In the case of Xen guest domains, we see both a larger number of instructions executed and a higher number of cache misses per instruction. But for Xen Dom0 or OpenVZ, the higher cache misses relative to the base system is mainly due to a larger number of instructions executed, because its number of cache misses per instruction is either lower than or comparable to the number for the base system.

We are also interested in what specific kernel functions are causing more L2 cache misses. For example, Table 1 shows the percentage of L2 cache misses for the OpenVZ and the base systems for high overhead kernel functions. The function `do_anonymous_page` is used to allocate pages for a particular application by the kernel. The functions `__copy_to_user_11` and `__copy_from_user_11` copy data back and forth from the user-mode to kernel-mode pages. These three functions together account for over 50% and 35% of total cache misses in the OpenVZ kernel and the base Linux kernel, respectively. This suggests that the cache misses in OpenVZ result mainly from the usual context switching that happens between the processes. The higher percentage of cache misses spent on these functions in OpenVZ could be contributing to the slightly higher response times we observe in Figure 3(b).

Because Xen uses a modified kernel, it is not possible to directly compare calls within it with the numbers obtained from the base system and OpenVZ. Tables 2, 3, and 4 show the kernel functions with the highest cache misses identified using Oprofile for the Web domain, the DB domain, and Dom0, respectively, in the Xen system.

The function `hypervisor_callback` is called when an event occurs that needs hypervisor attention. These events include various activities including page faults, and interrupt handling that usually happen in privileged mode in a normal kernel. After some preliminary processing of stack frames, the function `evtchn_do_upcall` is called to process the event and

Symbol Name	OpenVZ	Base
<code>do_anonymous_page</code>	31.84	25.24
<code>__copy_to_user_11</code>	11.77	9.67
<code>__copy_from_user_11</code>	7.23	0.73

Table 1: Base vs. OpenVZ — % of L2 cache misses

Symbol name	L2 cache misses (%)
<code>evtchn_do_upcall</code>	44.54
<code>hypervisor_callback</code>	32.00
<code>__copy_to_user_11</code>	3.57
<code>__do_IRQ</code>	2.50

Table 2: Xen Web kernel — % of L2 cache misses

to set up the domain to continue normally. These functions are the main source of overhead in Xen and reflect the cost of hypervisor-based virtualization.

For Dom0, the function `get_page_from_freelist` is similar to the function `do_anonymous_page` in OpenVZ and the base kernel. It allocates pages for use by kernel sub systems. The number of L2 cache misses in this function is at least 50% more than what is observed in the the OpenVZ and the base kernels.

6.2 Two-node

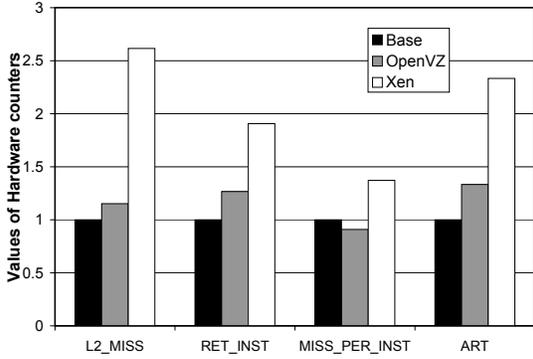
In this subsection, we present the Oprofile statistics for the two-node configuration, and reveal how the values of the hardware counters change as the number of application instances hosted on the two nodes increases from one to two.

We start with the two-node, one-instance scenario. Figure 11(a) shows the total number of L2 cache misses, total number of retired instructions, and the average number of cache misses per instruction for the base, the OpenVZ, and the Xen systems. Similar to the single-node case, the Xen system has the highest value for each of the three metrics, whereas the values for OpenVZ and the base system are more comparable. This is consistent with the trend we observe in the application response times that are also included in the figure (labeled as ART).

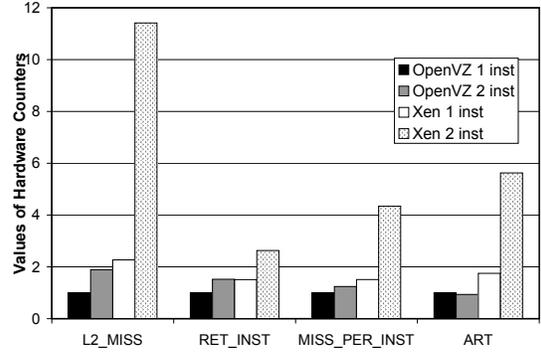
Figure 11(b) shows a comparison of the same metrics as the number of application instances increases from one to two in both the Xen and the OpenVZ systems. As we can see, when the number of applications goes

Symbol name	L2 cache misses (%)
<code>hypervisor_callback</code>	39.27
<code>evtchn_do_upcall</code>	28.71
<code>__do_IRQ</code>	2.43
<code>__copy_to_user_11</code>	2.34

Table 3: Xen DB kernel — % of L2 cache misses



(a) Two-node, one instance



(b) Two-node, from one instance to two instances

Figure 11: Single-node — counter values break down for three different systems

Symbol name	L2 cache misses (%)
hypervisor_callback	30.35
get_page_from_freelist	19.80
net_tx_action	9.55
__do_IRQ	1.61

Table 4: Xen Dom0 — % of L2 cache misses

from one to two, a larger number of instructions were executed in both systems, as expected. The number of L2 cache misses per instruction goes up only slightly in OpenVZ, but more than doubled in Xen. As a result, the total number of L2 cache misses in Xen is more than 5 times higher when there are two applications sharing the two nodes. This again can be the main reason for the significant increase in average response time in Xen as shown in the same figure.

Similar to what was done for the single-node case, we have analyzed a breakdown of the total L2 cache misses seen by the Xen or the OpenVZ system in the two-node configuration by different virtual servers and binaries, and looked into the specific kernel functions that accounted for the majority of the cache misses. Our key observations are similar to what we have made in the single-node case, therefore are not repeated here.

6.3 Two-node vs. single-node

We now compare the two placement configurations shown in Figure 2(a) and 2(b) in terms of the Oprofile statistics collected from both the Xen and the OpenVZ systems. In Figure 12, we show the values of the same hardware counters for the four configurations (Xen one-node, Xen two-node, OpenVZ one-node, and OpenVZ two-node). The metrics shown include the number of L2 cache misses, the number of retired instructions, the calculated number of cache misses per instruction, as

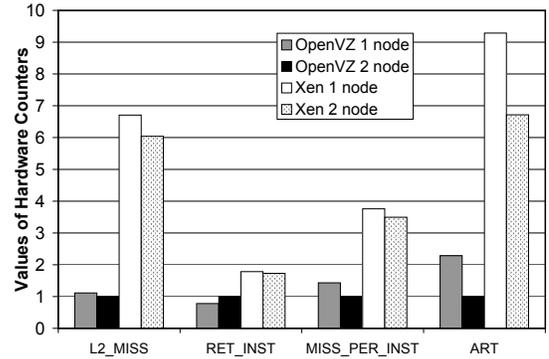


Figure 12: Two-node vs. single-node — Normalized hardware counter values from Oprofile

well as the average response time reported in the previous section. For each metric, the set of values shown have been normalized with respect to the OpenVZ two-node case. As we can see, for each of the four metrics, the value for the one-node configuration is higher than that for the two-node case in the Xen system. In addition, with both configurations, the values from OpenVZ are significantly smaller than those from Xen. This again demonstrates a strong correlation between the number of L2 cache misses and misses per instruction that occurred and the transaction response times observed under various configurations. It further validates that higher number of L2 cache misses is likely the culprit that leads to higher average latency observed at the application level.

7. PLAUSIBLE INTERPRETATION OF PERFORMANCE DIFFERENCES

We would like to discuss the big question: *Why do we have these differences in performance?* We believe that the fundamental difference in hypervisor-based and OS-level virtualization technologies is the trade-off between performance and isolation. They also differ in the goals they are trying to achieve. For example, Xen allows multiple OSes to run and that cannot be achieved by a simple OS-level technology running on a single kernel.

Xen allows more isolation by separating the multiple virtual machines into separate memory regions and only a selected set of para-virtualized hypervisor calls allow the OS to access the hardware resources. OpenVZ, simply creates a set of process domains that are monitored using beancounters and maintains the isolation at resource accounting level. As a result, whenever a switch happens between different VMs, Xen needs to flush the TLB and re-organize the memory pages incurring large overheads.

How do we improve the performance while maintaining good isolation? We inferred in this paper that the overhead is likely caused by the large number of L2 cache misses. The L2 cache misses can happen in four different ways.

- **Kernel instruction misses:** When the context switch happens, the instruction cache containing one kernel's instructions need to be replaced with the second one. This causes poor locality for the second VM causing memory latencies. We can improve this behavior by sharing parts of kernel instructions that are common to both VMs.
- **Kernel data misses:** This is difficult to reduce, because the data pages are usually constantly changing and a simple read-only memory sharing mechanism cannot be used. Any mechanisms allowing non-read-only memory sharing would compromise the isolation guarantees provided by the hypervisor-based technologies.
- **Application instruction misses:** Similar to kernel instruction misses, one can reduce the overheads by sharing read-only memories of the two applications.
- **Application data misses:** Unlike kernel data misses, application-level data misses can be reduced by user-level sharing mechanisms. For instance, newer versions of Java allow class sharing and this can be extended to allow sharing between different VMs. This can be a good performance boost for Java applications that constantly produce memory pressure.

The read-only memory page sharing is implemented in VMware ESX server [1] and is expected to be implemented in Xen as well. In the case of SMP architecture, the overhead is also caused by weak affinity scheduling of the current VM scheduling algorithms. It is possible to improve the performance by pinning the VMs to one particular processor to reduce the further cache misses created by processor switching. The performance may also improve in multi-core processors where L2 cache is shared between different processors. We plan to conduct further experiments to better understand the behavior of virtualized systems in the above scenarios.

8. CONCLUSIONS AND FUTURE WORK

We now summarize our key findings and provide answers to the questions we raised in the introduction in terms of how Xen and OpenVZ perform when used for consolidating multi-tiered applications, and how performance is impacted by different configurations for consolidation.

- For all the configurations and workloads we have tested, Xen incurs higher virtualization overhead than OpenVZ does, resulting in a larger difference in application response times when compared to the base Linux case.
- Performance degradation in Xen increases as application workloads increase. The average response time can go up by over 600% in the single-node case, and between 133% and 700% in the two-node case depending on the number of applications.
- For all the cases tested, the virtualization overhead observed in OpenVZ is low, and can be neglected in many scenarios.
- For all configurations, the Web tier CPU consumption for Xen is roughly twice that of the base system or OpenVZ. The CPU consumption of all systems and all containers increases linearly as the workload increases. The slope of increase in the case of Xen is higher than in OpenVZ and the base cases.
- The performance overhead in Xen is likely caused by the higher number of L2 cache misses, the majority of which occur in the `vmlinux` kernels of Dom0 and the two guest domains. The higher L2 cache misses can in turn be attributed to the higher number of cache misses per instruction, likely due to the context switches between the different guest domains.
- In the Xen system in a two-node setup, the percentage increase in response time from a single application instance to multiple instances is significant (over 400% for 800 threads per application)

while in OpenVZ this increase is much smaller (100%). With our system setup, the two nodes running Xen become overloaded when hosting four instances of RUBiS, whereas those using OpenVZ can host at least six without being overloaded.

- Hosting multiple tiers of two applications on the same node is not an efficient solution compared to the case of hosting them on different nodes as far as response time and CPU consumption are concerned. This again may be explained by the higher number of L2 cache misses incurred in the single-node configuration.

In conclusion, there are many complex issues involved in consolidating servers running enterprise applications using virtual containers. In this paper, we evaluated different ways of consolidating multi-tiered systems using Xen and OpenVZ as virtualization technologies and provided a quantitative analysis to understand the differences in performance overheads.

More can be done in extending this evaluation for various other complex enterprise applications, including applications with higher memory requirements or database-intensive applications. We hope that systems researchers can use these findings to develop optimizations in virtualization technologies in the future to make them more suited for server consolidation.

9. REFERENCES

- [1] M. Rosenblum. VMware's Virtual Platform: A virtual machine monitor for commodity PCs. In *Hot Chips 11: Stanford University, Stanford, CA, August 15-17, 1999*.
- [2] Microsoft Virtual Server, <http://www.microsoft.com/windowsserver/virtualserver/>.
- [3] 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 (SOSP)*, pages 164-177, October 2003.
- [4] OpenVZ: <http://openvz.org/>.
- [5] Linux VServer, <http://linux-vserver.org/>.
- [6] D. Price and A. Tucker. Solaris Zones: Operating system support for consolidating commercial workloads. In *Proceedings of the 18th Large Installation System Administration Conference (LISA)*, pages 241-254. USENIX, November 2004.
- [7] C. Amza, A. Ch, A.L. Cox, S. Elnikety, R. Gil, K. Rajamani, E. Cecchet, and J. Marguerite. Specification and implementation of dynamic Web site benchmarks. In *Proceedings of WWC-5: IEEE 5th Annual Workshop on Workload Characterization*, October 2002.
- [8] S. Soltesz, H. Potzl, M.E. Fiuczynski, A. Bavier, and L. Paterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. In *Proceedings of the EuroSys 2007*, March 2007.
- [9] Oprofile: A system profiler for Linux, <http://oprofile.sourceforge.net/>.
- [10] B. Clark, T. Deshane, E. Dow, S. Evanchik, M. Finlayson, J. Herne, and J.N. Matthews. Xen and the art of repeated research. In *USENIX Annual Technical Conference, FREENIX Track*, pages 135-144, 2004.
- [11] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel. Diagnosing performance overheads in the Xen virtual machine environment. In *Proceedings of the First ACM/USENIX International Conference on Virtual Execution Environments (VEE)*, pages 13-23, June 2005.
- [12] A. Menon, A.L. Cox, and W. Zwaenepoel. Optimizing network virtualization in Xen. In *Proceedings of the 2006 USENIX Annual Technical Conference*, pages 15-28, June 2006.
- [13] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in Xen. In *Proceedings of the ACM/IFIP/USENIX 7th International Middleware Conference, Melbourne, Australia*, volume 4290 of *Lecture Notes in Computer Science*, pages 342-362. Springer, November 2006.
- [14] D. Gupta, R. Gardner, and L. Cherkasova. XenMon: QoS monitoring and performance profiling tool. In *HP Labs Technical Report HPL-2005-187*, October 2005.
- [15] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. The design and implementation of Zap: A system for migrating computing environments. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, December 2002.
- [16] A. Whitaker, M. Shaw, and S.D. Gribble. Scale and performance in the Denali isolation kernel. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.
- [17] XenSource, <http://www.xensource.com/>.
- [18] Virtuozzo, <http://www.swsoft.com/en/virtuozzo/>.