Performance Evaluation of Virtualization Technologies for Server Consolidation

Pradeep Padala[1], Xiaoyun Zhu[2], Zhikui Wang[2], Sharad Singhal[2], and Kang G. Shin[1]

[1]University of Michigan
[2]Hewlett Packard Laboratories

Abstract

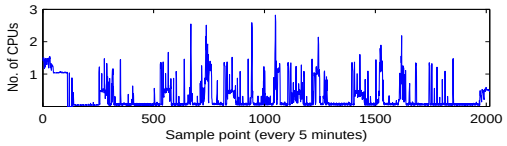
Server consolidation has become an integral part of IT planning to reduce cost and improve efficiency in today's enterprise data centers. The advent of virtualization allows consolidation of multiple applications into virtual containers hosted on a single or multiple physical servers. However, this poses new challenges, including choosing 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 over 400% in Xen and only a modest 100% in OpenVZ as the number of application instances grows from one to four. This large discrepancy is caused by the higher virtualization overhead in Xen, which is likely due to higher L2 cache misses and misses per instruction. A similar trend is observed in CPU consumptions of virtual containers. We present an overhead analysis with kernel-symbol-specific information generated by Oprofile.

1 Introduction

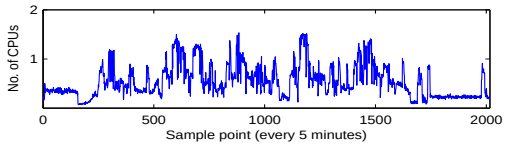
There has been a rapid growth in servers within data centers driven by growth of enterprises since the late nineties. The servers are commonly used for running business-critical applications such as enterprise resource planning, database, customer relationship management, and e-commerce applications. Because these servers and applications involve high labor cost in maintenance, upgrades, and operation, there is a significant interest in reducing the number of servers necessary for the applications. This strategy is supported by the fact that many servers in enterprise data centers are under-utilized most of the time, with a typical average utilization below 30%. On the other hand, some servers in a data center may also become overloaded under peak demands, resulting in lower application throughput and longer latency.

Server consolidation has become a common practice in enterprise data centers because of the need to cut cost and increase return on IT investment. Many enterprise applications that traditionally ran on dedicated servers are consolidated onto a smaller and shared pool of servers. Although server consolidation offers great potential to increase resource utilization and improve application performance, it may also introduce new complexity in managing the consolidated servers. This has given rise to a re-surging interest in virtualization technology. There are two main types of virtualization technologies today — *hypervisor-based technology* including VMware [1], Microsoft Virtual Server [2], and Xen [3]; and *operating system (OS) level virtualization* including OpenVZ [4], Linux VServer [5], and Solaris Zones [6]. These technologies allow a single physical server to be partitioned into multiple isolated virtual containers for running multiple applications at the same time. This enables easier centralized server administration and higher operational efficiency.

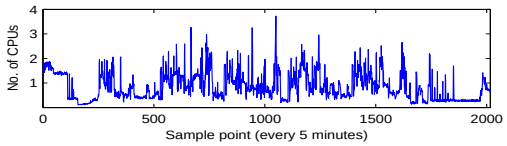
However, capacity management for the virtual containers is not a trivial task for system administrators. One reason is that enterprise applications often have resource demands that vary over time and may shift from one tier to another in a multi-tiered system. Figures 1(a) and 1(b) show the CPU consumptions of two servers in an enterprise data center for a week. Both have a high peak-to-mean ratio in their resource usage, and their peaks are not synchronized. This means if the two servers were to be consolidated into two virtual containers on a shared server, the resources may be dynamically allocated to the two containers such that both of the hosted applications



(a) CPU consumption of node 1



(b) CPU consumption of node 2



(c) Sum of CPU consumptions from both nodes

Figure 1: An example of data center server consumptions

could meet their quality-of-service (QoS) goals while utilizing server resources more efficiently. An adaptive CPU resource controller was described in [7] to achieve this goal. Similar algorithms were developed for dynamic memory management in VMware ESX server [8].

There is another important issue that is worth considering in terms of capacity management. Figure 1(c) shows the total CPU consumption from the two nodes. As we can see, the peak consumption is at about 3.8 CPUs. However, it does not necessarily imply that a total of 3.8 CPUs are sufficient to run the two virtual containers after consolidation due to potential virtualization overhead. Such overhead may vary from one virtualization technology to another. In general, hypervisor-based virtualization incurs higher performance overhead than OS-level virtualization does, with the benefit of providing better isolation between the containers. To the best of our knowledge, there is little published work quantifying how big this difference is between various virtualization technologies, especially for multi-tiered applications. In this paper, we focus on two representative virtualization technologies, Xen from hypervisor-based virtualization, and OpenVZ from OS-level virtualization. Both are open-source, widely available, and based on the Linux operating system. We use RUBiS [9] as an example of a multi-tiered application and evaluate its performance in the context of server consolidation using these two virtualization technologies.

In particular, we present the results of our experiments that answer the following questions, and compare the answers to each question between OpenVZ and Xen.

- How is application-level performance, including throughput and response time, impacted compared to its performance on a base Linux system?
- As workload increases, how does application-level performance scale up and what is the impact on server resource consumption?
- How is application-level performance affected when multiple tiers of each application are placed on virtualized servers in different ways?
- As the number of multi-tiered applications increases, how do application-level performance and resource consumption scale?
- In each scenario, what are the values of some critical underlying system metrics and what do they tell us about plausible causes of the observed virtualization overhead?

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 Section 5 and 6. Finally, we summarize our key findings and discuss future work in Section 7.

2 Related Work

A performance evaluation of Xen was provided in the first SOSP paper on Xen [3] that measured its performance using SPEC CPU2000, OSDB, dbench and SPECWeb benchmarks. The performance was compared to VMWare, Base Linux and User-mode Linux. 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 [11] to provide some insight into the possible causes of the performance overhead observed.

Menon *et al.* [12] conducted a similar performance evaluation of the Xen environment and found various overheads in the networking stack. 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.* [13] use the information gathered in the above work and investigate causes of the network overhead in Xen. They propose 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.

Soltesz *et al.* [14] have developed Linux VServer, which is another implementation of container-based virtualization technology on Linux. They have done a comparison study between VServer and Xen in terms of performance and isolation capabilities. In this paper, we conduct performance evaluation comparing OpenVZ and Xen when used for consolidating multi-tiered applications, and provide detailed analysis of possible overheads.

Gupta *et al.* [15] have studied the performance isolation provided by Xen. In their work, they develop 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 use XenMon [16] to detect performance anomalies. Our work is orthogonal to theirs by providing insight into using Xen vs. OpenVZ and different consolidation techniques.

To the best of our knowledge, the virtualization technologies have not been evaluated in the context of server consolidation. Server consolidation using virtual containers brings new challenges and, we comprehensively evaluate two representative virtualization technologies in a number of different server consolidation scenarios.

3 Testbed Architecture

Figure 2(a) shows the architecture of our testbed. We run experiments in three systems with identical setup. We compare the performance of Xen- and OpenVZ-based systems with that of a vanilla Linux system (referred to as *base system* here after). In a virtualized configuration, each physical node may host one or more virtual containers supported by Xen or OpenVZ as shown in Figure 2(b). Because we are interested in multi-tiered applications, two separate nodes may be used for the Web and the database tiers. Each node is equipped with a sensor collecting various performance metrics including CPU consumption, memory consumption and other performance events collected by Oprofile [11]. This data is collected on a separate machine and analyzed 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 two Gigabit network interfaces.

3.1 System configurations

We conduct our experiments on three different systems as explained below. All systems are carefully set up to be as similar as possible with the same amount of resources (memory and CPU) allocated to a particular virtual container.

3.1.1 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.

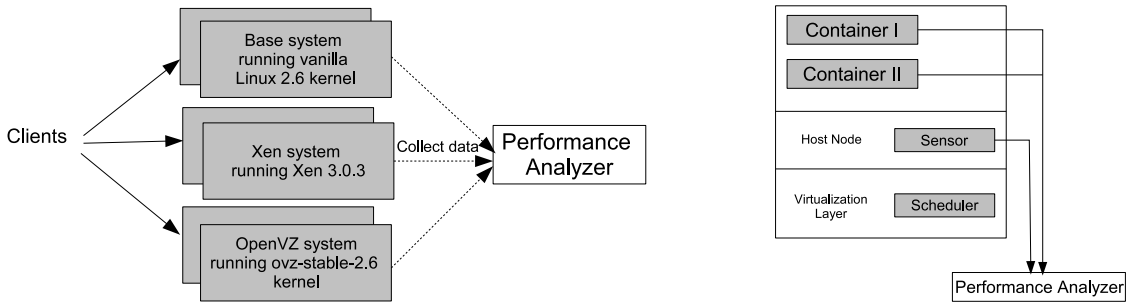
3.1.2 Xen system

Xen is a *paravirtualization* [17] technology that allows multiple guest operating systems to be run in virtual containers (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 (XenLinux) because the hardware abstraction presented by the hypervisor is similar but not identical to the raw hardware. The hypervisor contains a CPU scheduler that implements various scheduling policies including proportional fair-share, along with other modules such as the memory management unit.

We use the Xen 3.0.3 unstable branch [18] 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 exist idle CPU cycles. 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 allows creation of secure, isolated virtual environments (VEs) on a single node enabling server consolidation. Each VE performs and behaves exactly



(a) Testbed setup

(b) A virtualized server

Figure 2: System architecture

like a stand-alone server. They can be rebooted independently and a different distribution with separate root directory can be set up. One distinction between OpenVZ and Xen is that the former uses a single kernel shared by all VEs. Therefore, it does not provide the same level of fault isolation as in the case of Xen.

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

In the remainder of this paper, we use the term *virtual container* to refer to either a domain in Xen or a VE in OpenVZ.

3.2 Instrumentation

To measure the CPU consumption accurately, we wrote scripts that use existing tools to gather 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 case, 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 container. We use the data provided from `/proc/vz/vestat` to measure the amount of CPU time spent by a particular VE.

3.2.1 Oprofile

Oprofile [11] 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 can provide a wealth of information. Menon *et al*. [12] have modified Oprofile to support Xen. The resulting tool, *Xenoprof*, allows one to profile multiple domains in a Xen system. For each set of experiments, we analyze the data generated by Oprofile and provide our insights on the performance overheads.

We concentrate on 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 overhead occurs and quantify it.

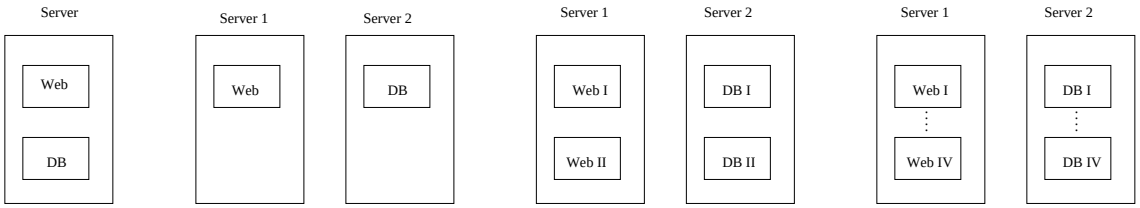
We monitor three hardware counters for our analysis:

- `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 access main memory.

We thoroughly analyze differences in these counter values in various configurations, and infer sources of overhead in respective virtualization technologies.

4 Design of Experiments

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 how application-level performance, including throughput and response time, are affected when using different virtualization technologies as well as different configurations for consolidation.



(a) Single-node (b) Two-node, one-instance (c) Two-node, two-instance (d) Two-node, four-instance

Figure 3: Various configurations for consolidation

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 (DB) tier executes data queries and interacts with the storage devices. During server consolidation, the various tiers of an application that are traditionally hosted on dedicated servers are moved onto shared servers. Moreover, when virtualization is involved, each of these tiers is hosted in a virtual container, which can have performance implications for the application.

We have chosen RUBiS [9], an online auction site benchmark, as an example of a multi-tiered application. We use a version of the application that has a two-tier structure: the Web tier contains an Apache Web server with PHP, and the DB tier uses a MySQL database server. RUBiS clients connect to the Web tier and perform various operations simulating auctions. 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 timed-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. Although RUBiS provides different workload mixes, we use the *browsing mix* in our experiments, which introduces higher load on the Web tier than on the DB tier.

We consider running RUBiS on consolidated servers. In addition to the comparison between OpenVZ and Xen, we also need to understand how application performance is impacted by different placement of application tiers on the consolidated servers. We evaluate the following two configurations for consolidation.

- **Single-node:** Both the Web and the DB tiers of a single RUBiS application are hosted on a single physical node (Figure 3(a)).
- **Two-node:** The Web and the DB tiers of a single RUBiS application are distributed on two separate nodes (Figure 3(b)).

There are additional reasons why one configuration may be chosen over the other in a practical scenario. For example, the single-node configuration may be chosen since it reduces network traffic by hosting multiple tiers of the same application on a single server, whereas the two-node option may be preferable in a case where it can reduce software licensing cost. In this work, we only focus on application performance differences in the two configurations and how performance scales as the workload increases.

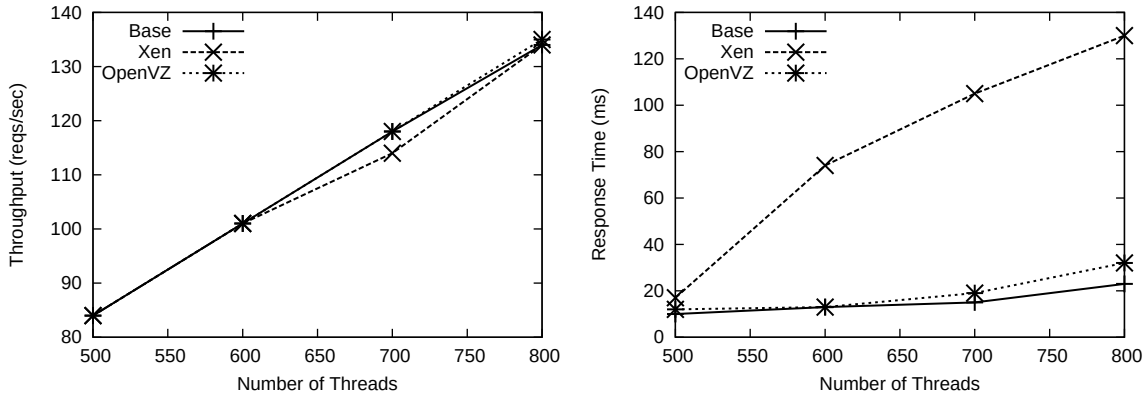
In addition, we are also interested in scalability of the virtualized systems when the number of applications hosted increases in the two-node case. Each node may host multiples of the Web or the DB tier as shown in Figures 3(c) and 3(d). In our experiments, we increase the number of RUBiS instances from one to two and then to four, and compare OpenVZ with Xen in application-level performance and system resource consumption. For each scenario, we provide a detailed analysis of the corresponding Oprofile statistics and point to plausible causes for the observed virtualization overheads.

5 Experimental Results

This section reports the results of our experimental evaluation using the RUBiS benchmark. In particular, we compare two different configuration options, *single-node* vs. *two-node*, for placing the two tiers of a single RUBiS application on physical servers. All experiments are done on the base, OpenVZ, and Xen systems, and a three-way comparison of the results is presented.

5.1 Single-node

In this configuration, both the Web and the DB tiers are hosted on a single node, as shown in Figure 3(a). In both the Xen and the OpenVZ systems, the two tiers run in two separate virtual containers. To evaluate the performance of each system, we scale up the workload by increasing the number of concurrent threads in the RUBiS client from 500 to 800. Each workload is continuously run for 15 minutes, and both application-level and system-level metrics are collected.



(a) Throughput (b) Average response time

Figure 4: Single-node - application performance

5.1.1 Performance evaluation

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

Figure 5 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 goes up linearly with the number of threads in the workload. The database consumption remains very low at about 1-4% of total CPU capacity, due to the Web-intensive nature of the browsing mix workload. A bigger difference can be seen in the Web tier consumption from the three systems. For each workload, the Web tier consumption in Xen is roughly twice the consumption experienced by the base system, while the OpenVZ consumption stays very close to the base case. As the workload increases, the slope of increase is higher in the case of Xen compared to the other two systems. This indicates higher CPU overhead in Xen than in OpenVZ, and should be related to the higher response times we observe from the application.

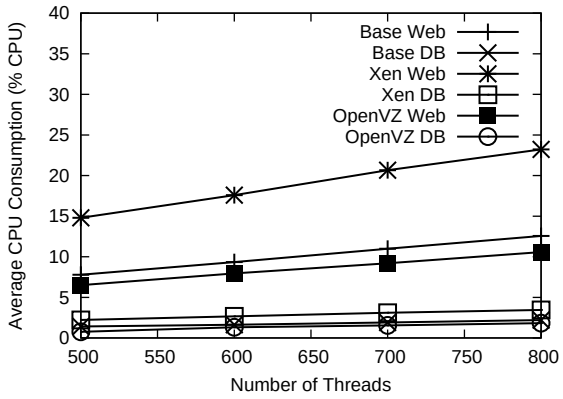


Figure 5: Single-node - average CPU consumptions

5.1.2 Oprofile analysis

Figures 6 shows the aggregate values of the selected hardware counters for the three systems when running 800 threads. For OpenVZ, each of the counter values is for the whole system including the shared kernel and all the virtual containers. For Xen, Oprofile provides us with a counter value for each of the domains, including Dom0. The DomU value in the figure is the sum of the values from the Web and DB domains. All counter values are normalized with respect to the base case. While all the counter values for OpenVZ are less than twice the corresponding values for the base case, the total number of L2 cache misses in Xen (Dom0 + DomU) is more than eleven times the base number. We therefore speculate that L2 cache misses are the main cause of the observed response time differences between Xen and OpenVZ.

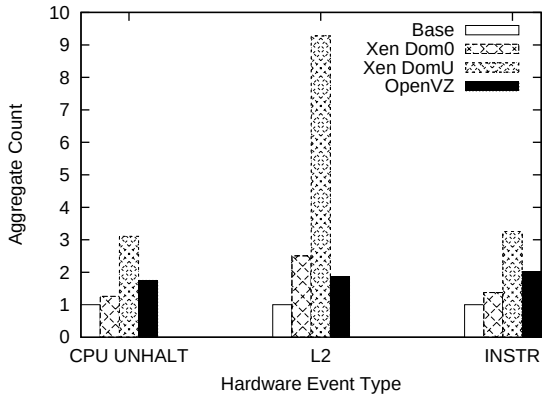


Figure 6: Single-node - Oprofile analysis

Symbol Name	OpenVZ	Base
do_anonymous_page	31.84	25.24
__copy_to_user_ll	11.77	9.67
__copy_from_user_ll	7.23	0.73

Table 1: Base vs OpenVZ - % of L2 cache misses

This is consistent with the higher CPU overhead in Xen from Figure 5, because more L2 cache misses causes more memory access overhead and puts higher pressure on the processor.

Table 1 shows the percentage of L2 cache misses for OpenVZ and the base system 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_ll` and `__copy_from_user_ll` copy data back and forth from the user-mode to kernel-mode pages. The data indicates that cache misses in OpenVZ result mainly from page re-mapping due to switching between virtual containers. The increase in the number of L2 cache misses causes the instructions to stall and reduces the total number of instructions executed.

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. Table 2 shows the highest cache miss functions identified using Oprofile for the Web domain in the Xen system.

The function `hypervisor_callback` is called

Symbol name	L2 cache misses (%)
hypervisor_callback	32.00
evtchn_do_upcall	44.54
__copy_to_user_ll	3.57
__do_IRQ	2.45

Table 2: Xen Web tier - % of L2 cache misses

Symbol name	Number of ret instr(%)
hypervisor_callback	39.61
evtchn_do_upcall	7.53
__do_IRQ	10.9l

Table 3: Xen kernel - % of retired instructions

Binary	OpenVZ	Base
libphp5.so	0.85	0.63
vmlinux	3.13	2.85
httpd	4.56	2.40
mysqld	4.88	2.30

Table 4: Base vs. OpenVZ - % cache misses/instruction for binaries

when an event occurs that needs hypervisor attention. These events include various activities including cache misses, 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 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.

Looking at the number of retired instructions for the Xen kernel (Table 3) for different functions, we see that in addition to overheads in hypervisor callback and upcall event handling, 10% of the instructions are executed to handle interrupts, which also is a major source of overhead. The higher percentage is due to increase in the interrupts caused by switching between the two domains.

We also looked at the hardware counters for a particular binary (like `httpd`, `mysqld`). As we compare OpenVZ to the base system, we do not see a clear difference between the two in either the percentage of cache misses or the percentage of instructions executed for each binary. However, there is larger difference in the percentage of cache misses per instruction. Table 4 shows this ratio for particular binaries that were running in OpenVZ and the base system. The dynamic library `libphp5.so` is responsible for executing the PHP scripts on the apache side. We can see that the percentage of cache misses per instruction is higher in OpenVZ for all four binaries. This indicates some degree of overhead in OpenVZ, which contributes to small increase in response times (Figure 4(b)) and slightly higher CPU consumption (Figure 5) compared to the base case. Unfortunately, we cannot directly compare the numbers obtained from particular domains in Xen to the numbers in the Table 4 as they do not include the associated work done in Dom0.

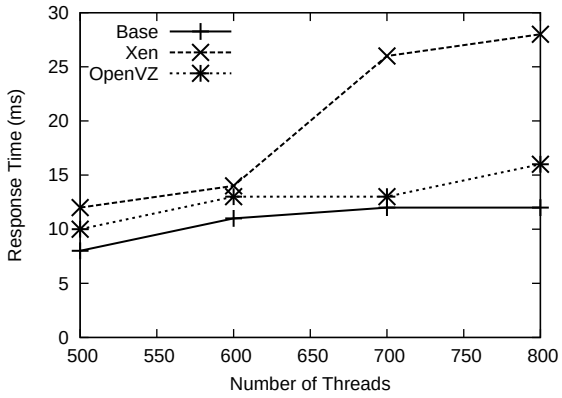


Figure 7: Two-node - average response time

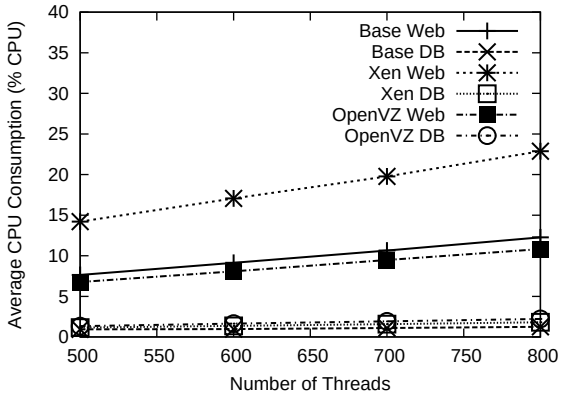


Figure 8: Two-node - average CPU consumption

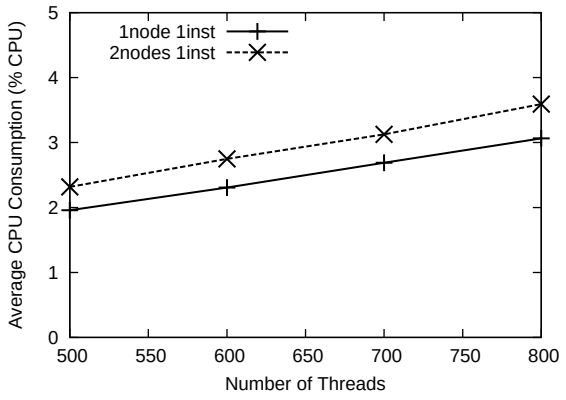


Figure 9: Single-node vs. two-node - Xen Dom0 CPU consumption

5.2 Two-node

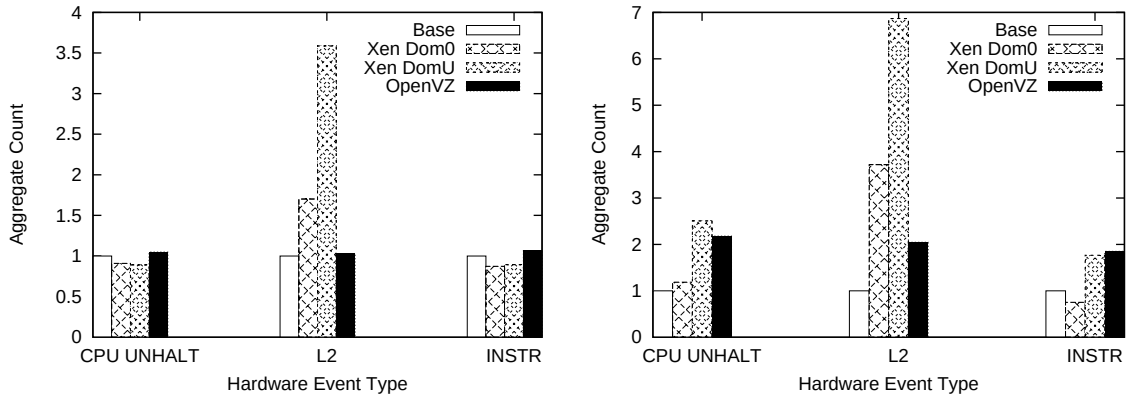
In this configuration, we run the Web and the DB tiers of a single RUBiS application on two different nodes, as shown in Figure 3(b). The objective of the experiment is to compare this way of hosting a multi-tiered application to the single-node case. Note that in the case of Xen and OpenVZ, there is only one container hosting each of the application components (Web tier or DB tier).

5.2.1 Performance evaluation

Figure 7 shows the average response time as a function of workload for the three systems. The throughput graph is not shown because it is similar to the single-node case, where the throughput goes up linearly with the workload and there is little difference between the virtualized systems and the base system. We see a small overhead in OpenVZ compared to the base case with the maximum difference in the mean response time below 5 ms. In contrast, as the worklaod increases from 500 to 800 threads, the response time from the Xen system goes from 13 ms to 28 ms with an increase of 115%. However, this increase in response time is significantly lower than that experienced by using Xen in the single-node case (28 ms vs. 130 ms for single-node with 800 threads).

Figures 8 shows the average CPU consumption as a function of workload for both the Web and the DB tiers. The trend here is similar to the single-node case, where the DB tier consumption remains low and the Web tier consumption goes up linearly as the the number of threads increases. The slope of increase in the Web tier consumption is higher for Xen compared to OpenVZ and the base system. More specifically, 100 more concurrent threads consume roughly 3% more CPU capacity for Xen and only 1% more for the other two systems.

Figure 9 shows the Dom0 CPU consumptions for both the single-node and two-node configurations for Xen. In the two-node case, we show the sum of the Dom0 consumptions from both nodes. In both cases, the Dom0 CPU consumption remains low (below 4% for all the workloads tested), and it goes up linearly as the workload increases. If the fixed CPU overhead of running Dom0 were high, we would have expected the combined consumption in the two-node case to be roughly twice that from the single-node case, since the former has two Dom0's. But the figure suggests it is not the case. The difference between the two cases is within 0.5% of total CPU capacity for all the worklaods. This implies that Dom0 CPU consumption is mostly workload dependent, and there is very little fixed cost.



(a) Web tier (b) DB tier

Figure 10: Two-node - Oprofile analysis

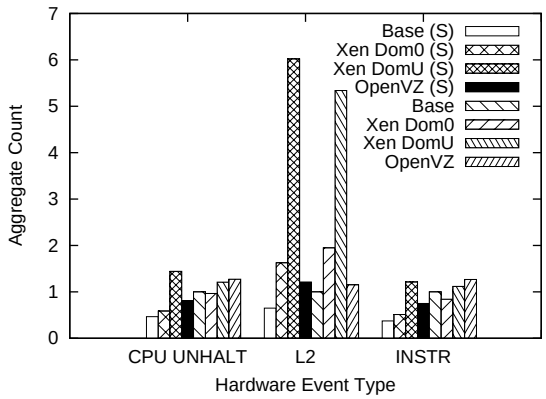


Figure 11: Single-node vs. two-node - Oprofile analysis

Symbol Name	Single	Two
do_anonymous_page	25.25	24.33
__copy_to_user_ll	9.68	12.53
__copy_from_user_ll	0.73	1.11

Table 5: Single-node vs. two-node - base system % of L2 cache misses

Symbol Name	Single	Two
do_anonymous_page	31.84	33.10
__copy_to_user_ll	11.77	15.03
__copy_from_user_ll	7.23	7.18

Table 6: Single-node vs. two-node - OpenVZ % of L2 cache misses

Symbol Name	Single	Two
hypervisor_callback	32.00	43.00
evtchn_do_upcall	44.53	36.31
__do_IRQ	2.50	2.73

Table 7: Single-node vs. two-node - Xen Web tier % of L2 cache misses

Symbol Name	Single	Two
hypervisor_callback	6.10	4.15
evtchn_do_upcall	44.21	42.42
__do_IRQ	1.73	1.21

Table 8: Single-node vs. two-node - Xen Web tier % of L2 cache misses/instruction

5.2.2 Oprofile analysis

We now analyze the sources of overhead in the two-node case using Oprofile. Figures 10(a) and 10(b) show the values of the hardware counters for the two nodes hosting the Web tier and the DB tier, respectively, for a workload of 800 threads. For both tiers, we see that the total number of L2 cache misses in Xen (Dom0 + DomU) is between five to ten times those from the base case, and the OpenVZ number is only twice that of the base case.

Figure 11 provides a comparison of these values between the single-node and two-node cases. Each counter value shown for the two-node case is the sum of the two values from the two nodes. All values are normalized with respect to the base system in the two-node setup. We observe that relative L2 cache misses are higher for the single-node case as compared to the two-node case for both OpenVZ and Xen. For example, for the total number of L2 cahce misses, the ratio between the Xen number (Dom0 + Domu) and the base number is 11 in the single-node case vs. 7.5 in the two-node case. This is expected due to extra overhead caused by running two virtual containers on the same node.

Table 5 shows the number of L2 cache misses in the base kernel for both the single-node and two-node configurations. For the two-node case, we add the aggregate counters for each kernel function in both nodes and normalize them with respect to the total number of cache misses from the two nodes. The same comparison is

shown in Table 6 for the OpenVZ system. The percentage of L2 cache misses for different kernel functions stay almost the same between the single-node and two-node cases (within 4% of each other), except for the `__copy_to_user_ll` function where we see the two-node value being 28% higher. Comparing Table 6 to Table 5, we see that all the values are higher in OpenVZ than in the base case indicating higher page re-mapping overhead in the OpenVZ system.

Table 7 shows a similar comparison for Xen-specific kernel calls in the Xen system Web tier. If we add the first two rows, we see that the total number of cache misses for the functions `hypervisor_call_back` and `evtchn_do_upcall` is very similar in the single-node and two-node cases. The values for the `_do_IRQ` call are similar too. However, the difference is larger in the number of cache misses per instruction, which is shown in Table 8. We see the sum of the first two rows being 10% higher in the single-node case, and the percentage of L2 cache misses/instruction for interrupt handling being 43% higher. The reason is that more instructions are executed in the two-node case because of less stalling due to cache misses.

6 Scalability Evaluation

In this section, we investigate the scale of consolidation that can be achieved by different virtualization technologies. We increase the number of RUBiS instances from one to two then to four in the two-node configuration, and compare the scalability of Xen and OpenVZ with respect to application performance and resource consumption.

6.1 Response time

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

Figure 12(a) shows the average response time as a function of workload when running two instances of RUBiS on two nodes. For either Xen or OpenVZ, there are two curves corresponding to the two instances I and II. The response time remains relatively constant for OpenVZ but goes up about 500% (15 ms to roughly 90 ms) as the workload increases from 500 to 800 threads.

Figure 12(b) shows the same metric for the case of running four instances of RUBiS, and we see an even greater increase in the average response time in the Xen case. As the workload increases from 500 to 800 threads, the average response time experienced by each application instance goes from below 20 ms up to between 140 and 200 ms, an increase of more than 600%, yet the average response time in the OpenVZ case stays near or below 30 ms in all cases.

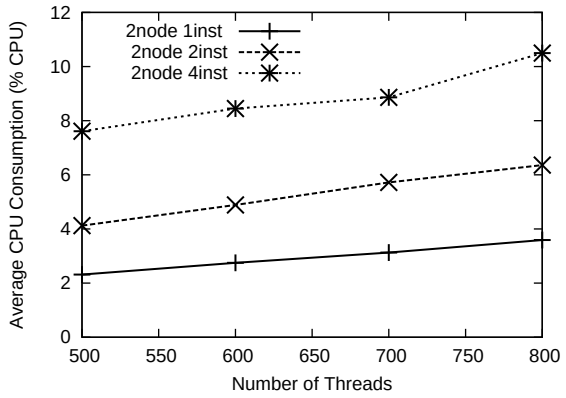


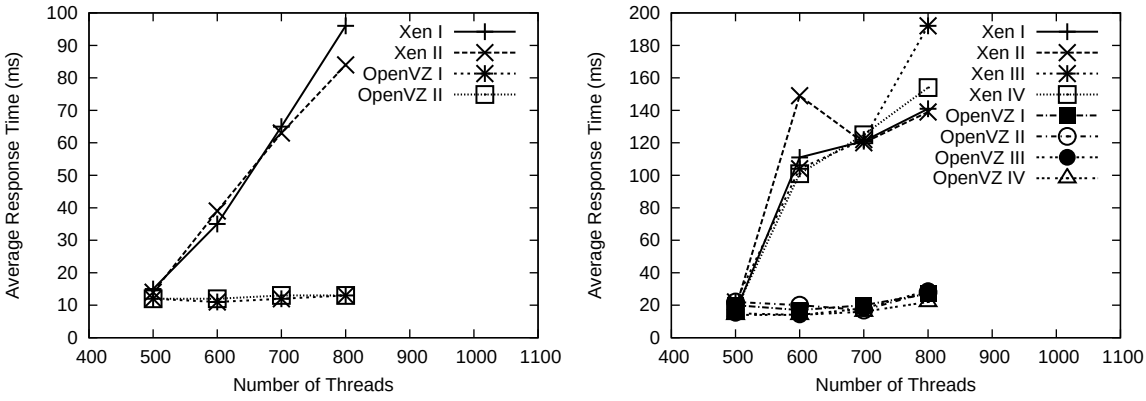
Figure 14: two-node multiple instances - Xen Dom0 CPU consumption

In Figure 13(a), we compare the four consolidation configurations for a workload of 800 threads per application, and show the mean response time averaged across all the application instances in each configuration. In the Xen system, the average response time per application in the two-node case 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 respect to application-level performance.

What is also interesting is that the single-node configuration (one-node one-inst in Figure 13(a)) results in worse application performance and worse scalability than the two-node case using Xen. For example, if a maximum average response time of 160 ms is desired, we can use the two-node option to host four instances of RUBiS with 800 threads each, but would require four separate nodes for the same number of instances and comparable performance if the single-node option is used.

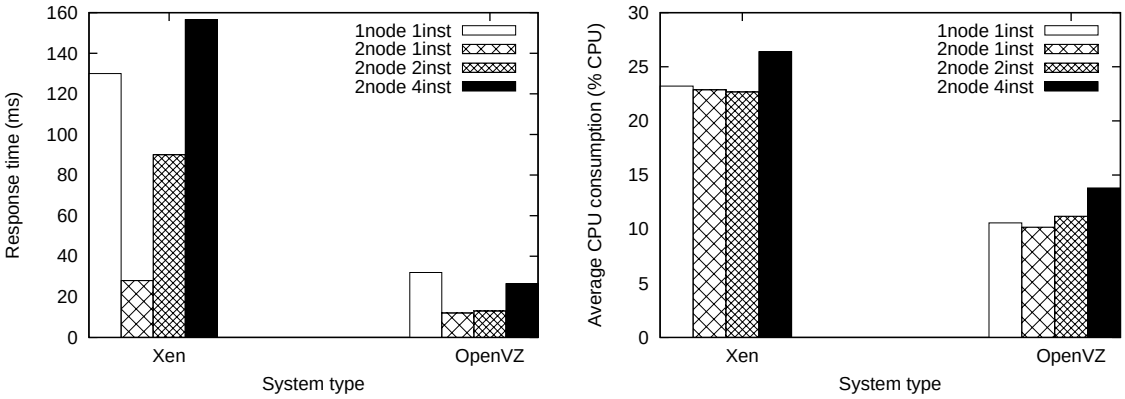
6.2 CPU consumption

In Figure 13(b), we compare the four consolidation configurations in terms of the average Web tier CPU consumption seen by all the application instances with a workload of 800 threads each. 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 fit at least two more instances of the RUBiS application.



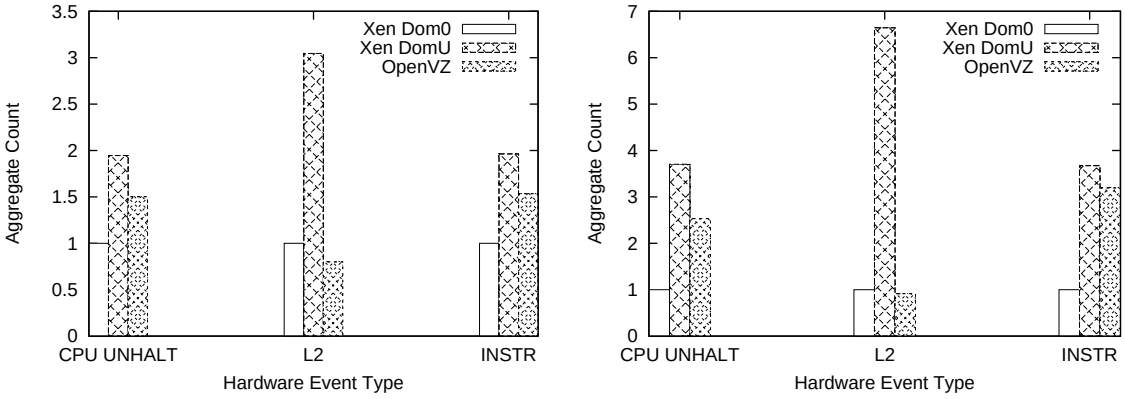
(a) Two instances (b) Four instances

Figure 12: two-node multiple instances - average response time



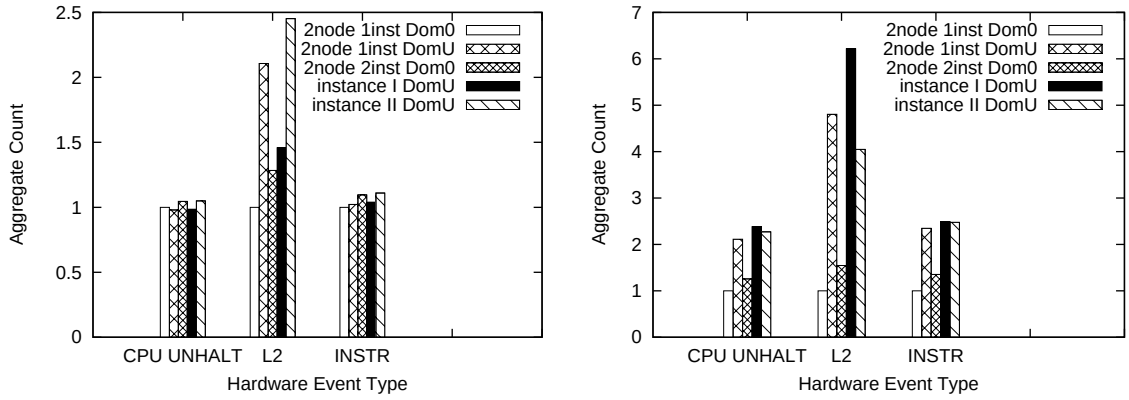
(a) Average response time (b) Web tier CPU consumptions

Figure 13: Comparison of all configurations (800 threads)



(a) Web tier (b) DB tier

Figure 15: two-node two instances - Oprofile analysis for Xen and OpenVZ



(a) Web tier (b) DB tier

Figure 16: Single instance vs. two instances - Oprofile analysis for Xen

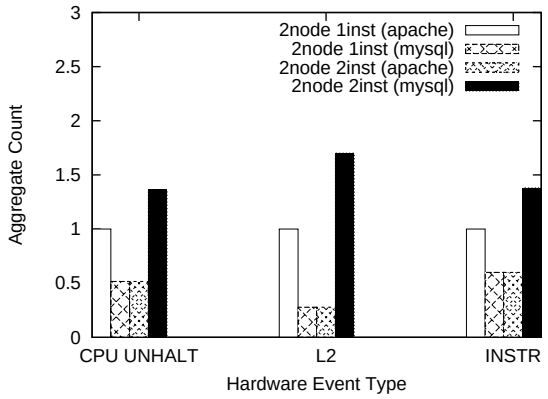


Figure 17: Single instance vs. and two instances - Oprofile analysis for OpenVZ

Figure 14 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 number of RUBiS instances hosted. For each scenario, the Dom0 consumption goes up approximately linearly as the workload increases from 500 to 800 threads. This is expected because Dom0 handles I/O operations on behalf of the domains causing its consumption to scale linearly with the workload. There is a slight increase in the slope as the number of RUBiS instances grows. Moreover, as the number of instances goes from one to four, the Dom0 consumption increases by a factor of 3 instead of 4, showing certain degree of multiplexing in Dom0.

To investigate how much more overhead is generated 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 domains is 25% and the average Dom0 consumption is 14%. The latter is higher than that obtained using the UP kernel.

6.3 Oprofile analysis

We also perform overhead analysis using Oprofile for the case of two instances, and compare it to the results from the one instance case. Figures 15(a) and 15(b) show the hardware counter values from the two nodes hosting the Web and the DB tiers for both Xen and OpenVZ. All values are normalized with respect to the Xen Dom0 values. As seen before, the L2 cache misses are considerably higher in Xen user domains on both nodes and are the main source of overhead.

We now compare the counter values with those from the two-node one-instance case. Figures 16(a) and 16(b) show the comparison of counter values from both nodes in the Xen system. We can see that the number of L2 cache misses for both tiers is 25% higher with two instances. It is also interesting to note that the number of instructions and percentage of execution time do not differ by much. We infer that the overhead is mainly due to the cache misses caused by memory remapping etc.

Figure 17 shows the corresponding data for OpenVZ. We observe similar patterns but the increase in the L2 cache misses is not as high as in Xen. Note that the L2 cache misses from both containers are included in the two instance case.

Turning our attention to the cause of this overhead, let us look into the hardware counters for particular binaries in Xen. Table 9 shows the percentage of cache misses per instruction for the Web tier with one and two instances of RUBiS. The percentage of cache misses per instruction experienced in the kernel (`vmlinux`) is 30% lower with single instance (5.20%) than with two instances (7.44% on average).

Table 10 shows the comparison of the percentage of L2 cache misses for different kernel functions in Xen. We see that there is not much difference in the percentage of L2 cache misses for any function. However, the

Binary	One-inst	Two-inst I	Two-inst II
libphp5.so	0.80	0.73	2.10
vmlinux	5.20	7.35	7.53
httpd	3.01	3.13	4.80

Table 9: Web tier % L2 cache misses/instruction for binaries in Xen

Symbol name	One-inst	Two-inst I	Two-inst II
hypervisor_callback	43.00	37.09	38.91
evtchn_do_upcall	36.31	42.09	34.32
__copy_to_user_ll	2.84	3.11	2.61
__do_IRQ	2.73	2.33	2.62

Table 10: Web tier % of L2 cache misses in Xen kernel

percentage of L2 cache misses per instruction is different. We can conclude that the main reason of overhead comes from the higher L2 cache misses caused by page re-mapping, but the percentage of cache misses remains the same.

We see similar patterns in the OpenVZ case shown in Tables 11 and 12. Note that we cannot differentiate between the different containers in OpenVZ. It is interesting to note that the addition of another instance does not increase the overhead in OpenVZ that much.

7 Conclusions and Future Work

We 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 larger difference in application performance 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 115% and 600% in the two-node case depending on the number of applications.
- For all the cases tested, the virtualization overhead observed in OpenVZ is limited, 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. CPU consumption of all systems and all containers goes up linearly as the workload increases. The slope of increase in the case of Xen is higher than in OpenVZ and the base cases.
- The main cause of performance overhead in Xen is the number of L2 cache misses.
- In the Xen system, relative L2 cache misses are higher in the single-node case compared to the sum of cache misses in the two-node case. Between the base and OpenVZ systems, the difference is minor.
- In the Xen system, the percentage of L2 cache misses for a particular function in the kernel is similar in the single-node and two-node cases. But the percentage of misses per instruction is higher in the single-node case.
- The percentage increase in the number of L2 cache misses for single and multiple instances of RUBiS is higher in Xen than in OpenVZ. In other words, as the number of applications increases, OpenVZ scales with less overhead.
- In the Xen system in a two-node setup, the percentage increase in response time from 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 in the two-node case, the Xen system becomes overloaded when hosting four instances of RUBiS, while the OpenVZ system should be able to host at least six without being overloaded.
- Hosting multiple tiers of a single application on the same node is not an efficient solution compared to

Binary	One-inst	Two-inst I + II
libphp5.so	0.60	0.78
vmlinux	2.40	2.86
httpd	2.94	4.73

Table 11: Web tier % L2 cache misses/instruction for binaries in OpenVZ

Symbol name	One-inst	Two-inst I + II
do_anonymous_page	33.10	33.31
__copy_to_user_ll	10.59	11.78
__copy_from_user_ll	7.18	7.73

Table 12: Web tier % of L2 cache misses in OpenVZ kernel

the case of hosting them on different nodes as far as response time and CPU consumption are concerned.

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 quantitative analysis to understand the differences in performance overheads.

More work 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.

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/windowsserversystem/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://en.wikipedia.org/wiki/OpenVZ.

[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] P. Padala, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, K. Salem, and K. G. Shin. Adaptive control of virutalized resources in utility computing environments. In *Proceedings of the EuroSys 2007*.

[8] C.A. Waldspurger. Memory resource management in VMware ESX server. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[9] 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.

[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] Oprofile: A system profiler for Linux, http://oprofile.sourceforge.net/.

[12] 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.

[13] 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.

[14] 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 (to appear), http://www.cs.princeton.edu/ mef/research/vserver/paper.pdf*.

[15] 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.

[16] D. Gupta, R. Gardner, and L. Cherkasova. XenMon: QoS monitoring and performance profiling tool. In *HP Labs Technical Report HPL-2005-187*, October 2005.

[17] 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.

[18] XenSource, http://www.xensource.com/.