and cache footprint, due to the unnecessary high frequent context switch. The further analysis reveals that the high frequency is a result of a) inefficient processing of interrupts, and b) over aggressive hypervisor scheduling.
3) Optimizations to improve the efficiency of interrupt processing and to limit the over aggressive hypervisor scheduling, are proposed. Dynamically-allocable tasklets is used to eliminate the unnecessary involvement of hypervisor scheduling to idle VM context, and adaptive hypervisor scheduling rate control is adopted to limit the context switch if the scheduling frequency exceeds certain threshold.

The results show the improved memory and cache efficiency with reduction of the overall CPI, resulting in the improvement of server consolidation capability by 15% in SPECvirt_sc2010. In the meantime, our optimization achieves an up to 50% acceleration of service response, which greatly improves the QoS of Xen virtualization solution.

The rest of this paper is organized as follows. Section 2 presents the background of server consolidation including a brief introduction of Xen VMM and the consolidation workload we use. Section 3 introduces the experimental configuration. In Section 4, we present the detailed overhead breakdown, followed by the illustration of two optimizations to address the high frequency context switch issue in Section 5. The performance benefits of our optimization are shown in Section 6. Section 7 describes related work, and Section 8 concludes this paper.

2. Background

2.1 Server Consolidation

Multiple virtual machines are allowed to run on the same physical hardware in system virtualization, so as to increase system utilization and thus reduce cost. Server consolidation reduces the number and variety of components in the environment. This may not be limited to servers but also to other physical elements such as tapes, disks, network devices and connections, operating systems, and peripherals involved in the server consolidation. It becomes easy to move and change systems, applications, and peripherals with fewer hardware and software standards to manage. Although the physical hardware is shared across the virtual machines, the virtual machines are completely isolated from each other and each virtual machine runs a separate operating system instance with its own applications.

Performance characterization and analysis for server consolidation [8] [9] [11] [32] is important for deployment with fair sharing of resources, providing feedback to IT administrators and platform architects, projecting and optimizing future platform performance and so on. In this paper, SPECvirt_sc2010[1] is adopted for performance analysis of server consolidation.

2.2 Xen Virtual Machine Monitor

A virtual machine monitor [12] [13] [14] [15] allows multiple operating systems to share a single machine safely. It isolates operating systems and controls accesses to hardware resources. We use open source VMM, Xen, which is widely adopted as a representative VMM in both academia and industry, for our study. The organization of Xen [6] is depicted in Figure 1. Xen consists of two elements: hypervisor and the driver domain. The hypervisor provides an abstraction layer between the guest operating systems and the actual hardware. The driver domain, a privileged VM, called domain0, manages other guest VMs, called domainU. One of the major functions of the driver domain is to conduct real I/O operations to a bare device on behalf of domainU to implement a reliable I/O architecture [16] [17] [18] [19]. With hardware assistance, Single Root I/O Virtualization and Sharing (SR-IOV) [23][24], which enables efficient sharing of a single I/O device among multiple VMs, provides a foundation for efficiently utilization of I/O resources as reaching near-native I/O performance. Although this paper is based on Xen, the problem addressed here is not limited to this specific hypervisor. We have observed the same issue in KVM.

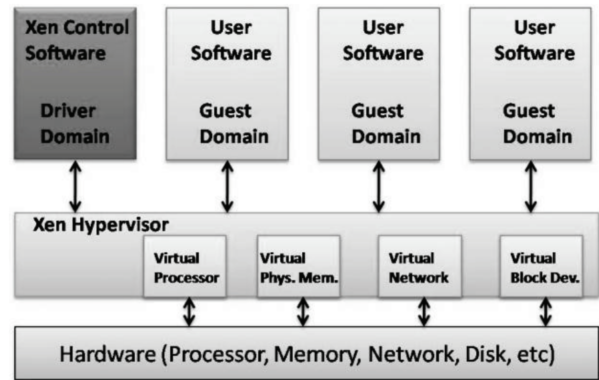

Figure 1. The Xen virtual machine environment

2.3 SPECvirt_sc2010 Benchmark

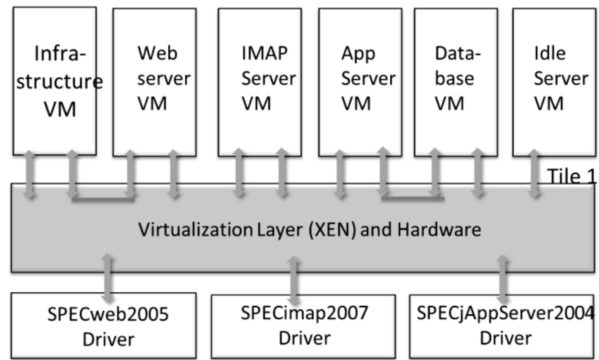

Figure 2. SPECvirt_sc2010 block diagram

SPECvirt_sc2010 [7] is SPEC's first benchmark addressing performance evaluation of datacenter servers used in virtualized server consolidation. The benchmark utilizes several SPEC workloads, to complete certain amount of transactions, representing applications that are common targets of virtualization and server consolidation as shown in Figure 2. Each of these standard workloads is modified to match a typical server consolidation scenario of CPU resource requirements, memory, disk I/O, and network

[1] The benchmark runs discussed here are for our research and non-compliant with the SPEC run-rules. The data presented here are only to illustrate the points discussed in this paper and cannot be compared with any other SPECvirt_sc2010 results

utilization. These workloads are modified versions of SPECweb2005, SPECjAppServer2004, and SPECmail2008. Scaling is achieved by running additional sets of virtual machines, called "tiles", until overall throughput reaches a peak. One tile consists of 6 different hardware-assistant VMs (HVM). All workloads must continue to meet required quality of service (QoS) criteria. The compliant results are necessary to meet all the Qos requirements. Performance metrics (throughputs) are obtained by calculating the arithmetic mean of the 3 normalized values per tile and summing up the scores for all tiles.

3. Experiment Configuration

The server under test is the latest Intel Xeon 5680 server with two 3.33GHz processors. Each processor has 6 cores, 12 threads with Hyper-Threading technology [20] on and 12MB shared L3 cache. Xen 4.1.0 is selected as the virtual machine monitor. We allocate enough storage and network devices to make sure that there are no hardware bottlenecks. One LSI HBA is used to connect to an external disk enclosure. Each tile is assigned 4x64GB Intel solid status disk as storage. Due to the public availability of SR-IOV network card & iSCSI [33] in data centers and the performance advantage of hardware-assisted virtualization solution, we use SR-IOV & iSCSI [33] in our experiment environment. As depicted in Figure 3, all the disk storages are placed at the remote iSCSI target machine, directly linked through 10 GB network card to the server under test. All VMs, running under server side, select host's SR-IOV virtual functions as their networks. Each guest accesses its VFs directly. In this way, IO requests from disks are exchanged for network bandwidth, walking through the software layer with the assistance of SR-IOV and iSCSI.

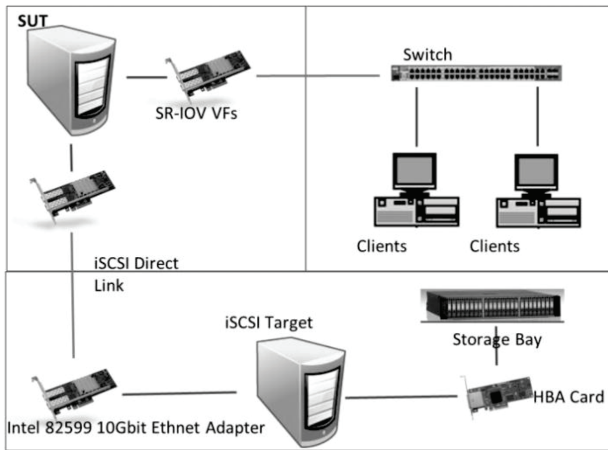

Figure 3. SR-IOV & iSCSI solution

4. Characterization and Analysis

In this Section, we adopt well-designed profiling methodologies to thoroughly investigate hypervisor's overhead and show the challenges met under server consolidation. Accordingly, several possible solutions are presented as the initial steps in section 5 to conquer such challenges. Although this work is based on the Xen hypervisor, we believe the analysis methods and challenges raised are equally applicable to other hypervisors.

4.1 Scalability Challenges

To provide insights into behaviors of server consolidation, we present the load scalability of SPECvirt_sc2010 on the latest 2-way commercial server. As shown in Figure 4, when the system load is low, the overall system CPU utilization scales up linearly with the system throughput. 2-tile load provides 99% increase in CPU utilization over 1-tile and 4-tile achieves 111% CPU utilization increase over 2-tile. These numbers indicate relatively good CPU utilization scalability. However, 8-tile consumes 219% more CPU resources than 4-tile, indicating a sharp increase of CPU utilization. The increase ratio becomes even greater at peak performance implying underlying high overhead of scalability as the system load is overwhelming. As mentioned in Subsection 2.3, SPECvirt_sc2010 workload cares not only about the throughputs, but also about QoS. As illustrated in Figure 4, the total response time (geometric mean result of three sub-workload's response time) keeps relatively low until close to the peak throughput from 0.13ms to 0.32ms (146% increase), an exponential-like increase from 8-tile to 9-tile configuration. The dramatic increase of the total response time near the peak performance implies potential bottlenecks existing with high system load, as the hypervisor is unable to function efficiently under such condition.

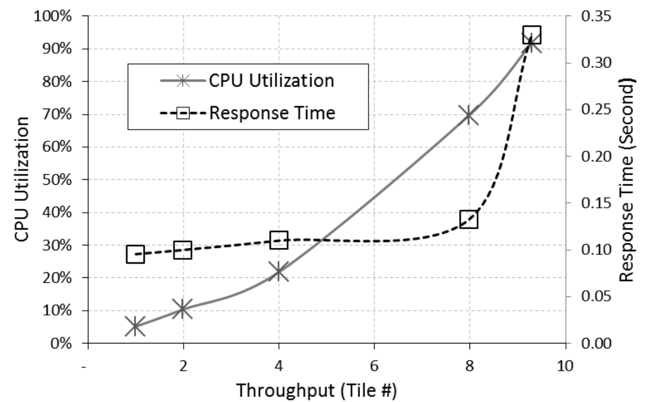

Figure 4. CPU utilization & response time trend with throughput

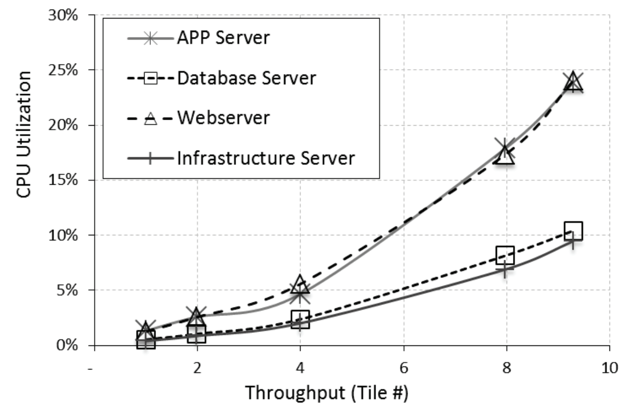

Figure 5. CPU utilization for each VM with throughput

The breakdown of CPU utilization scalability for each VM is shown in Figure 5 (Idle server is not illustrated here due to low

CPU utilization number). VMs of webserver and JAVA application server scale worse than those of database server and infrastructure server when system load becomes heavy. Interestingly, both webserver and java application server are assigned two virtual CPUs for a single VM to satisfy CPU requirements, while the rest VMs are assigned only one virtual CPU due to less consumption. One possible reason is that more number of virtual CPUs brings the extra overhead such as the inter-communication between different virtual CPUs, which causes the worse scalability of related VMs. It also implies the efficiency of current hypervisor that manages multiple virtual CPUs drops, as the number of VM increases.

Turning our attention to the cause of the high CPU utilization consumption with heavy system stress shown in Figure 4, the overall system CPU utilization is divided into four different parts – guest user, guest kernel, hypervisor and domain0. Because these four parts are running in different privilege levels – the guest part (HVM guest) runs in non-root mode and hypervisor and domain0 (PV guest) run in root mode – we can easily breakdown them by programing the hardware performance counter separately. Guest parts are the valuable work while hypervisor and domain0 are so called virtualization overhead. As illustrated in Figure 6, all components, except domain0, increase sharply as tile number increases. At the peak throughput, hypervisor part occupies more than 25% of the total CPU cycles – much higher than expected. It is equally to say that the hypervisor needs to occupy 6 cores of the total 24 logical cores at the peak performance, which is identified to be a big overhead.

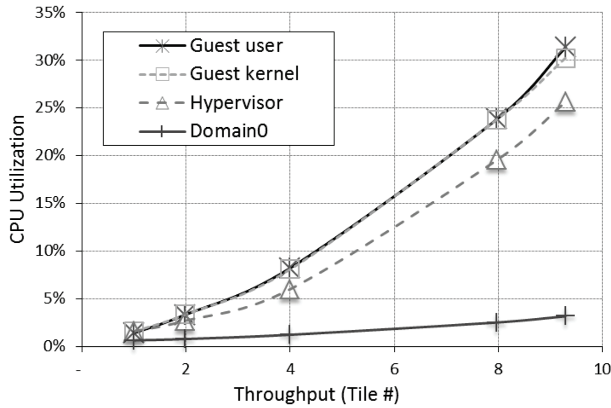

Figure 6. System CPU utilization breakdown

Equation 1 shows cycles per throughput are decided by the multiplicative of Cycles per instruction (CPI) and Path Length (PL). CPI refers to the number of clock cycles that happens when an instruction is being executed, used to describe a processor's performance from one aspect. PL refers to instructions required per transaction, deemed as a measurement of the software's performance on particular computer hardware. As displayed in Figure 7, the trend of PL keeps relatively flat when system load is low, but increases slightly (5.9% increase from 4-tile to 8-tile), when reaching the peak throughput, implying less efficient software execution when system stress is heavy. Whereas CPI increases sharply as throughput grows (88% increase from 1-tile to 9-tile). Obviously, the dramatic increase of CPI is the major contributor for non-linear CPU utilization increase of both guest and hypervisor parts.

$$Cycles/Throughput = CPI \ X \ PL \quad (1)$$

Among various aspects which may impact the value of CPI, data from hardware performance counter points out that the increase of cache miss rate such as LLC and TLB, depicted in Figure 8, plays the vital role to the increase of overall CPI. Cache miss rate of LLC increases by 2.67x and that of TLB increases by 0.38x from 1-tile to 9-tile. High cache miss rate was caused by larger memory and cache footprint under server consolidation as VM number goes up. In other words, as the number of consolidated VMs grows, the memory footprint consumed by both VMs and hypervisor becomes greatly increased so that current hardware resource like hierarchical cache and TLB cannot process efficiently, resulting in slow instruction execution. Besides the hardware approach of enlarging the cache size, there are software ways to reduce the footprint, such as to merge the same page together as KSM [35]. Also a well-tuned scheduler to achieve better context switch frequency – satisfy latency/throughput at the same time – is another software approach.

Without effective cache usage on multicore platform, such caches can cause thrashing that severely degrades system performance. Some cache-aware schedulers have been discussed and developed recently in the native system [34]. However, in virtualization environment, it becomes more complex as the intrusion of isolation of various VMs running to share the same hardware resource. A well-designed scheduler in virtualization should not only efficiently isolate various VMs to make them access the underlying hardware resource fairly, but also treat the hardware as a whole to maximize the throughput. For example, as one of the practical implementations, KSM allows to share equal anonymous memory across different processes and in turn also across different KVM virtual machines. The object of KSM is to increase memory density, or conversely, to reduce the memory size with the same number of running VMs

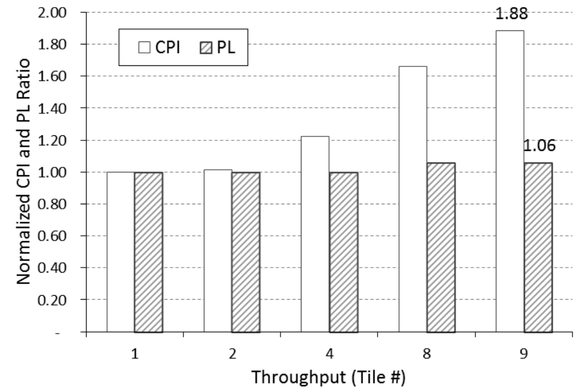

Figure 7. CPI and Path Length trend with throughput

To resolve the feckless usage of cache in virtualization, controlling context switches frequency by eliminate unnecessary ones is another important approach to be worthy considered. Unnecessary context switches result in unnecessary cache flush, thus causing extra overhead. Low frequency of context switch among VMs will produce small overhead of hypervisor and assure cache hot thus reduce cache footprint. But it may result in the QoS issue, especially for latency sensitive workloads. High frequency of

context switch will otherwise satisfy the QoS requirement to some extent, but impair the cache performance thus consume high CPU utilization. So how to select the right frequency is the key issue scheduler should solve. In section 5, we will show in detail that current scheduler in virtualization lacks considerations to eliminate unnecessary context switch and impairs the overall performance. Meanwhile, as the first step to conquer these challenges, a scheduler with fine control of context switch frequency is implemented to showcase the benefits from both the CPI and PL sides.

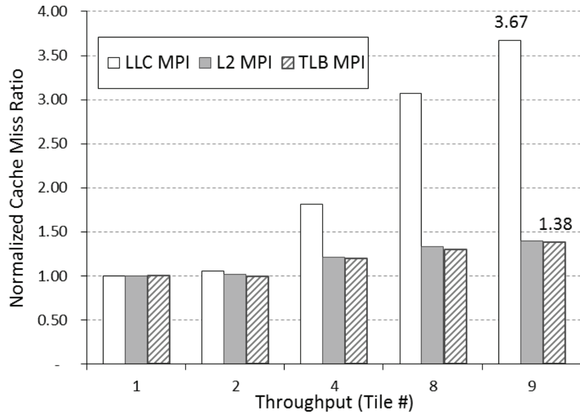

Figure 8. Cache miss rate trend with throughput

4.2 Hypervisor Overhead Breakdown

Scalability issue, as observed in Figure 4, is primarily caused by the sharp increase of CPI, which can be mitigated by either hardware or software ways. We mainly focus on software solutions, so that it's worthwhile to dig into the behaviors of hypervisor to find out the clues.

To further investigate the hypervisor's overhead, we need to firstly clarify the terminology – VMExit. When the running VMs are HVM guests, a VMExit marks the point at which a transition is made between the VM currently running and the hypervisor that executes the sensitive instructions on behalf of guest system. In other words, when a VMExit event happens, related physical CPU migrates from non-root mode to root mode, henceforth, hypervisor performs its system management functions according to the VMExit reasons. With the assistance of tracing tools – XenTrace – we breakdown the hypervisor's overhead (25% of the overall system CPU cycles as shown in Figure 6) according to such categories (56 VMExit reasons in total). Thus it helps to detect which VMExit event causes the big overhead in hypervisor.

Figure 9 depicts hypervisor's detailed CPU cycles breakdown according to the VMExit number at peak performance. The VMExit event of 'External Interrupt' consumes the largest portion, more than 12% CPU cycles. The following major parts are 'APIC Access', 'HLT' and 'IO instruction' which consume 4.78%, 3.73% and 2.94% respectively. The rest 52 VMExit only contribute less than 2% CPU utilization. Regarding such distribution, the major overheads of hypervisor are centralized in a small amount of VMExit events. In the following paragraphs, we will take a look into the reasons of top three VMExit events and propose possible ways to reduce the overhead.

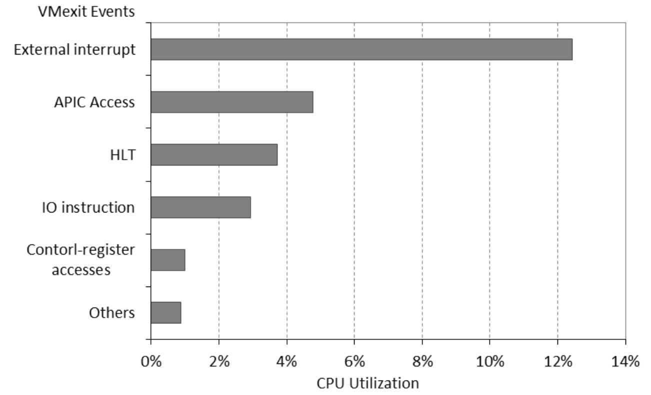

Figure 9. Hypervisor CPU utilization breakdown according to VMExit number

a) 'External Interrupt' VMExit Event

'External Interrupt' VMExit event responds to the handling process of system interrupt requests (IRQs) including handler's top and bottom halves. In our experiment, the software emulated network and disk are passed through by SR-IOV NIC and iSCSI disk solutions, which produce lots of network IRQs. It is readily to understand that the 'External Interrupt' VMExit event composes a large portion of the hypervisor's overhead.

IRQs could happen at any time in the following cases: a) system 'Halt' period; b) virtual CPU running state or c) 'Hypervisor' running state. Only in case b) will system generate the 'External Interrupt' VMExit event and transition from guest to hypervisor context to handle it. In other cases, system enters hypervisor directly without such event happening. Therefore, as system load is heavy, interrupts are more probably to interrupt current virtual CPU's running state. One issue emerges here for external interrupt handling process. If the current running virtual CPU, disrupted by the external interrupt, is the target virtual CPU to which external interrupt delivers, a virtual IRQ will be injected directly when this virtual CPU enters from hypervisor to its running state. However, if the target VM is running on a different physical CPU, an inter-processor interrupt (IPI) will be imperatively sent to the target physical CPU to interrupt it and inject virtual IRQ. In addition, if the target virtual CPU is not running, a scheduling request should be raised to wake it up. Such extra procedures, caused by IPIs and scheduling requests, will possibly aggrandize and thus impair the running streamline of VMs and cause performance turbulence, especially under the condition with large scales of VMs. From the tracing data at peak performance, lots of IPIs and scheduling events, happening during the handling process of 'External Interrupt' VMExit, demonstrate such problem. Sending interrupts to the wrong core or waking up target virtual CPU frequently are the major causes of the high overhead in processing 'External Interrupt' VMExit.

In [30], it proposed an interrupt migration method to avoid most of IPIs by marking target core status at virtual CPU migration. Interrupts are deliver to the right core with this information except for the condition that the target VM migrates to another core, in which an IPI is needed to be sent to the original core to clear the old vector. Consequently, if the frequency of virtual CPU migration is high with lots of VMs consolidating on one system, the mitigation degree will be relatively small. In such situation, the scheduler should comply with the handling process of external interrupts to balance both efficiency and fairness. As a practical

example, we enhance current Xen schedule's capability in section 5 to overcome the scalability issue caused by excessive scheduling during external interrupt handling process.

b) 'APIC Access' VMExit Event

Accesses to the APIC-access page inside the guest VMs are considered as virtualized APIC accesses, causing VMExit event during which register access operations, such as EOI and ICR, are emulated by hypervisor. The procedure of software emulation consumes lots of CPU cycles as shown in Figure 9. In [30], it is observed that, EOI access accounts for most percentage of all kinds of 'APIC Access' emulations under SR-IOV condition. By passing instruction fetch/emulation stage, it shortens the code path of vEOI emulation greatly, resulting in decrease of total CPU usage. However, the overhead of 'APIC Access' VMExit is still high when the number of IRQs is large for consolidation workloads.

c) 'HLT' VMExit Event

'HLT' VMExit event occurs when the current VM is idle and executes HALT instruction. Hypervisor marks the current VM's state as 'blocked' and enters scheduling procedure. If there are no available VMs to run at this time, system will enter idle state. The CPU cycles consumed in 'HLT' VMExit as shown in Figure 9 were calculated by eliminating the idle state. It is the real overhead consumed by hypervisor to process scheduling. For CPU intensive workloads, rare number of 'HLT' VMExit events is triggered due to long running time slice. Whereas for IO intensive and client-server type workloads, such as SPECvirt_sc2010, large number of 'HLT' VMExit events is produced due to intermittent running-state exchanges of virtual CPUs. Apparently, scheduling work is considered as the vital factor to reduce the overhead in this process. For example, if the pattern of tasks running inside VMs can be detected, then a deliberate scheduling scheme can be carried out to maximize the efficiency of cache usage.

The above analysis of hypervisor's behaviors demonstrates that scheduler is one of the vital factor in the constitution of virtualization overhead. Current scheduling scheme in Xen shows bottlenecks on the massive advanced system with heavier load (more VMs and heavier stress) as shown in Figure 4. In the following section 5, a close analysis related to scheduling challenges will be presented with specified identification. Then two optimizations are presented to showcase the performance benefits.

5. Optimization Methods

In this section, we analyze the performance of Xen's credit scheduler and propose two solutions, dynamically-allocable tasklets and context-switch rate controller, to optimize the scalability performance. The result shows the improvement of server consolidation capability by 15% in SPECvirt_sc2010. In the meantime, our optimization achieves an up to 50% acceleration of service response.

5.1 Challenges in Scheduler

In order to exactly identify the software scalability bottlenecks, code level profiling tools are necessary. OProfile is a system-wide profiler for Linux systems, capable of profiling all running code at low overhead. It is based on statistical profiling – continually sampling currently executing code at every fixed hardware events. Large set of sample approximating to the real hardware event distribution are employed. Xenoprofile [14] extends to Xen and Oprofile to fully profile across multiple domains and cover code in user processes, kernel and hypervisor. We strengthen Xenoprofile's functionalities by enabling call-graph to obtain the full picture of callees constitution for specified functions. As shown in Figure 11, for each function entry in the left column, the right side functions are called by this entry. We also see a special entry with a '[self]' marker. This records the normal samples for the function itself, but the percentage becomes relative to all callees. This allows comparing time spent on the function itself with that on functions it calls. In order to reduce the profiling overhead, we choose 4 as the call-graph depth.

As shown in Figure 10, cycles of top four functions increase as system load and throughput grow. It makes sense for 'vmx_asm_do_vmentry' to consume the biggest part of CPU resource due to the primary path walked when the system transits from VM to hypervisor. The 'Schedule' function consumes the second largest part. Moreover, the following functions, such as 'do_softirq' and 'context_switch, are all related to the scheduling process. For example, the majority amount of 'softirq' is raised to trigger the scheduling process as listed in Figure 11.

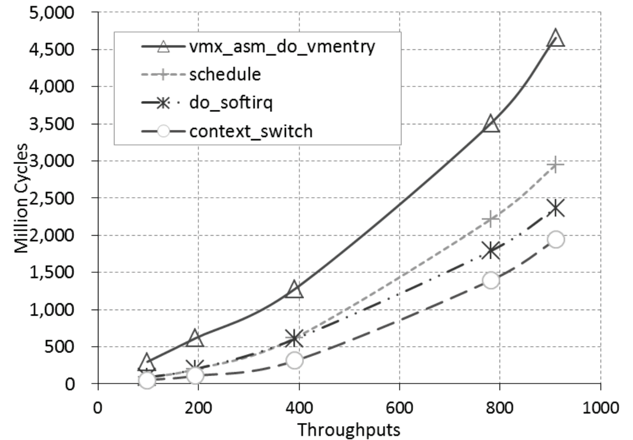

Figure 10. Top functions call-graph trend with throughput

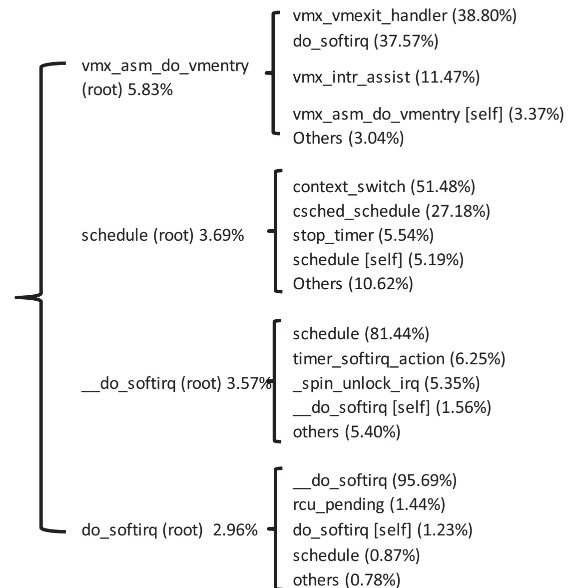

Figure 11. Top hot functions call-graph distribution

By tracing each scheduling process from the beginning to the end, cycles utilized by the context switch procedure are obtained. Table 1, cycles consumed by the context switch process occupy more than 27% of total hypervisor cycles at peak performance. From the frequency side, data in Table 1, collected by counting based tools, shows the procedure of scheduling (Sched: scheduler) is triggered 29 thousand times per second per physical core, and 26 thousand of which cause context switches (Sched: context switches). Context switch here refers to the process of de-schedule the current running virtual CPU and schedule in the next virtual CPU. One question is that is it reasonable to trigger so many context switches – approximately 26 thousand per second for one physical thread at peak performance? Stated differently, the average running tile slice for a virtual CPU once scheduled in is far less than tenth of a microsecond.

Table 1. Scheduler events number & CPU utilization at peak

Scheduler events	Number/second/thread
Sched: scheduler	29,819
Sched: context switches	26,551
Hypervisor	CPU Utilization
Total	26.23%
Context switches	7.28%

From the cause number distribution of context switch shown in Figure 12, most of the context switches take place in the 'External Interrupt' VMExit events, constituting 65% of the total number. Subsequently, 21% context switches occur in the 'HLT' VMExit events. Since 'HLT' VMExit occurs when the current VM is idle and executes HALT instruction, hypervisor primarily activates the scheduling process. However, the high frequency of context switches in the 'External Interrupt' VMExit events is unreasonable. From the code path of handling pass-through 'IRQ' (produced by SR-IOV devices) in Table 2, hypervisor raises the 'SCHEDULE_SOFTIRQ' to trigger a scheduling operation during 'External Interrupt' handler. Thus, system switches to the idle virtual CPU's context to process such tasklets. Basically, the idle virtual CPU shares the same context as the original one so that the overhead should be small. However, if the number of external interrupts is large enough, such kind of transition overhead cannot be ignored anymore.

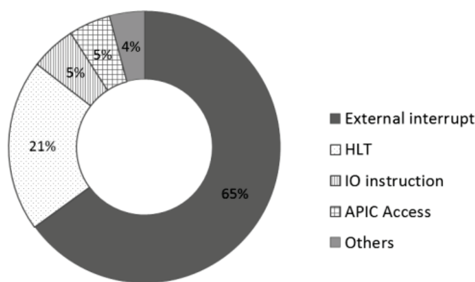

Figure 12. The Cause of Context switch

On the other hand, in virtualization environment, all external interrupts are handled by hypervisor that physical IRQs designated to some VMs are injected as virtual IRQs. Correspondingly, hypervisor kicks related VCPUs by raising scheduling requests combined with IPIs in some conditions, as discussed in section 4.2. Therefore, if the number of IRQ is very huge, the scheduling happens more frequently which explains the scheduling distribution in Figure 12. For current scheduler in Xen, there is no scheme to control the scheduling rate to eliminate the unnecessary scheduling requests so as to cause high frequency of context switch.

Table 2. IRQ pass-through delivery code path

```
Hvm_do_IRQ_dpci
-->|tasklet_schedule(&dpci->dirq_tasklet)
-->|-->|tasklet_schedule_on_cpu(t, smp_processor_id());
-->|-->|-->|tasklet_enqueue(t)
-->|-->|-->|-->cpu_raise_softirq(cpu, SCHEDULE_SOFTIRQ);
```

Therefore, scheduler in Xen meets bottlenecks, which originate from the following two aspects: a) inefficient interrupt handling process mechanism and b) lack of context switch frequency control. In next paragraphs, as the first step to conquer the scheduling challenges for virtualization under server consolidation, two optimizations are proposed, aiming to reduce the context switch frequency thus improve the overall throughput.

5.2 Dynamically-allocable tasklets

Considering the first bottleneck, since not all tasklet users need to run in virtual CPU context (more specifically, the idle virtual CPU context), the best fix is to make it per-tasklet configurable. Based on what we observed, the method of dynamically-allocable tasklets is devised. In this new mechanism, tasklets are dynamically-allocable tasks, running in either virtual CPU context or in softirq context on at most one CPU at a time. Softirq versus virtual CPU context execution is specified during per-tasklet initialization. This method avoids all tasklets running in virtual CPU context at the same time. Consequently, a number of unnecessary context switches is expected to be reduced.

By adopting the dynamically-allocable tasklets method, the number of context switches at peak performance is reduced by 54%, which is a significant improvement to the efficiency of interrupt processing, as shown in Table 3. However, the scheduling number is still high, more than 13k per second per thread, which means the average running time slice for one virtual CPU is still below 0.1 million second. A context switch rate controller is further proposed to solve this issue in the next paragraph.

Table 3. Context switch number comparison at peak performance

Scheduler events (#/send/thread)	Original	Configurable tasklet	Reduction(%)
Sched: scheduler	29819	13668	54.16%
Sched: context switches	26551	12006	54.78%

5.3 Context-Switch Rate Controller (CSRC)

Three different CPU schedulers were introduced in recent years – Borrowed Virtual Time (BVT) [28], Simple Earliest Deadline First (SEDF) [10] and Credit Scheduler [26] [29] – all of which allow the user to specify CPU allocation via CPU share (weights). The credit scheduler, Xen's latest scheduler, is a proportional share scheduler with a load balancing feature for SMP systems. The virtue of the credit scheduler is the simplicity of the operation with reasonable fairness guarantee and performance.

Figure 13 displays the flow of Xen's credit scheduler. There are two major parts in this flow: 1) pick up the next running virtual CPU and 2) do context switch when selecting a new different running virtual CPU. Since credit scheduler sorts run queue according to priority instead of credit, the current running virtual CPU is added to the tail of the run-queue with the same priority. Therefore, if the run-queue with the same priority as current virtual CPU is not empty, the current virtual CPU will be scheduled out and the head virtual CPU in the run queue will be scheduled in. It explains the data in Table 3 why the number of context switch (12006) is very close to the number of scheduler triggered (13668). At peak throughput, the numbers of virtual CPU and physical CPU are in the high ratio 60:24, so it is possible that there are runnable candidate virtual CPUs in the run-queue as the scheduling process is triggered. In this condition, the current virtual CPU is more probably to be scheduled out, following the context switch process.

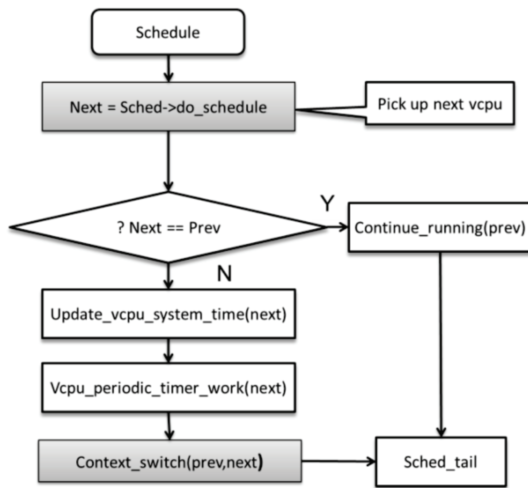

Figure 13. Flow of credit scheduler

Considering excessive number of hypervisor scheduling in Xen's current scheduler, we carry out one proposal to reduce the scheduling's overhead by controlling the frequency of context switch, called Context-Switch Rate Controller (CSRC). The basic philosophy is to reduce the context switch number, if possible, when the rate of VCPU scheduling exceeds the threshold. Two criterions are presented here: 1) to skip the current scheduling process, if the frequency of context switch is bigger than the threshold during last period (10 million second) under the condition that the last running virtual CPU is still runnable (not blocked). The frequency number, counted during the last 10 million second period is applied to judge whether the scheduling frequency is too high. 2) if the last running virtual CPU runs less than some time slice (1million second for example) and is still runnable, to skip this scheduling process, aiming to elongate the running slice for specified virtual CPUs whose running time slice is too short.

Taking the flow in Figure 14 for example, the current running virtual CPU is vcpu1 and the system is under hypervisor's environment. When the scheduler is triggered at some point, it will first judge whether the frequency of scheduling is too high. If yes, it will check whether vcpu1 is still runnable. If it is still yes, vcpu1 will return directly to skip the following scheduling process. If the scheduling frequency is below the threshold, it will check whether vpu1's last running slice is less than 1 million second, if so and vcpu1 is still runnable, it will return to continue its running stage directly.

After applying the CSRC method, the number of context switch at peak performance (with CSRC method, system obtains higher performance than previous result) reduces slightly as shown in Table 4, but the number of context switch reduces almost by 20%, demonstrating CSRC method effectively reduces the frequency of context switch. Detailed performance benefits, brought by the proposed methods, are to be exhibited in section 6.

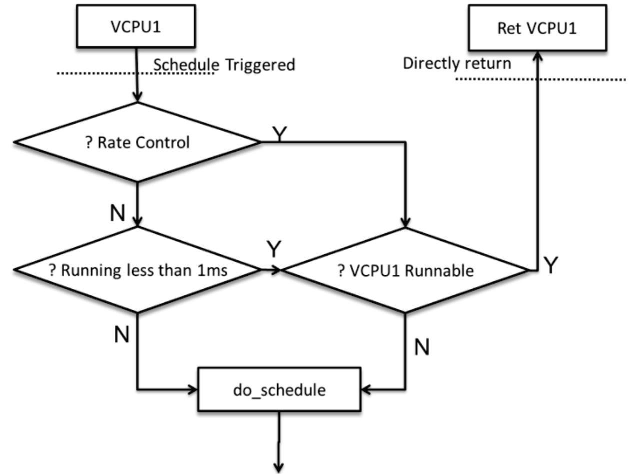

Figure 14. Proposals of SRC scheme

Table 4. Context switch number comparison at peak performance

Scheduler events (#/s/thread)	Configurable tasklet	CSRC & Configurable tasklet	Reduction(%)
Sched: scheduler	13668	13616	0.38%
Sched: context switches	12006	9685	19.33%

6. Evaluation

As depicted in Figure 16, by adopting the method of dynamically-allocable tasklet, the CPU utilization decreases from 92% to 89% comparing to the original result at peak performance (9-tile). After supplying CSRC method, the system CPU utilization decreases even more, about 81% at the same throughput comparing to the original peak results. Furthermore, due to lower CPU utilization, more loads can be added to achieve higher throughput with CSRC. When considering the compliant results – achieving QoS requirements – the peak performance (throughput per CPU utilization) with CSRC is improved by 15% comparing to the original result, as shown in Figure 15. There is no obvious performance change when system load is light. SPECvirt_sc2010 workload takes both throughput and QoS into consideration. In Figure 17, it reveals that proposed methods of dynamically-allocable tasklets and CSRC shorten the response time significantly when close to peak performance, explaining that such methods achieve higher throughput with qualified QoS. It is also observed that, when system load is light, the response time of proposed methods is slightly higher than original results due to relative less frequent scheduling when applying CSRC method. However it has little influence on the overall performance.

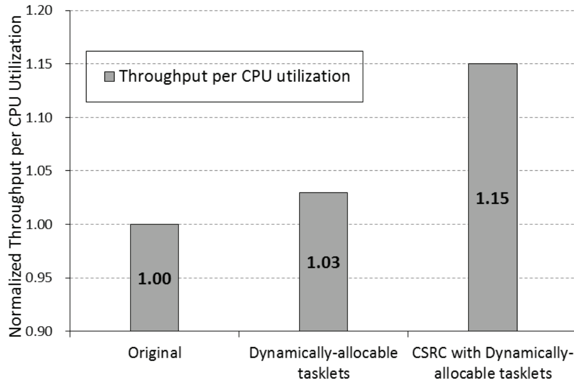

Figure 15. Peak throughput per CPU utilization comparison

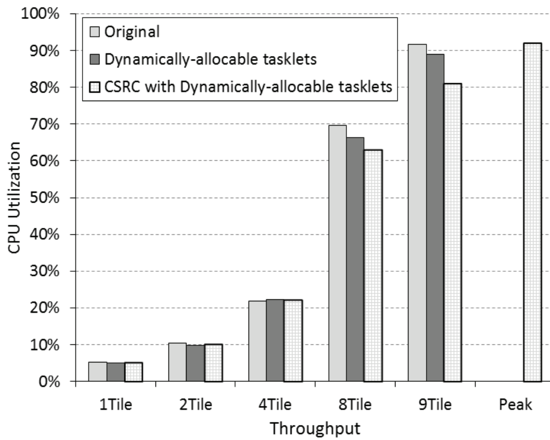

Figure 16. Proposals of dynamically-allocable tasklets and CSRC improve reduce CPU utilization and improve overall throughputs

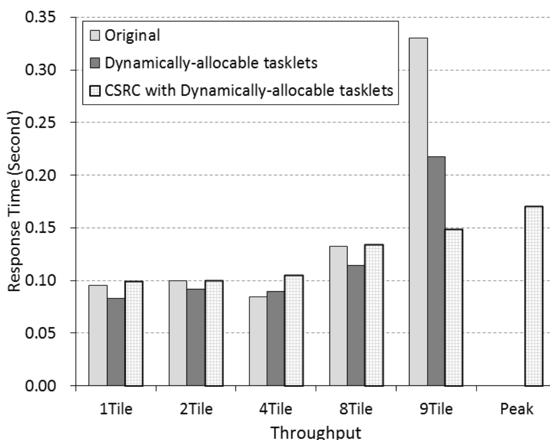

Figure 17. Proposals of dynamically-allocable tasklets and CSRC shorten the response time significantly when close to peak performance

As displayed in Figure 18, PL of the optimized CSRC method at peak performance (9-tile) reduces by 4%, implying more efficient software execution – less number of context switches reduces the instructions required for each function. From the CPI side shown in Figure 19, CPI of optimized CSRC method obtains 6% reduction than original result at 9-tile. Less number of context switches reduces the cache miss ratio – 6% reduction for LLC and 15% for TLB, resulting in lower CPI. All above prove that CSRC benefits from both hardware and software aspects.

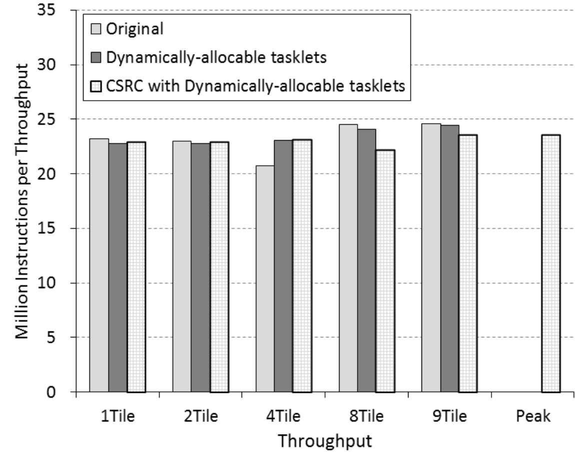

Figure 18. Path length trend comparisons

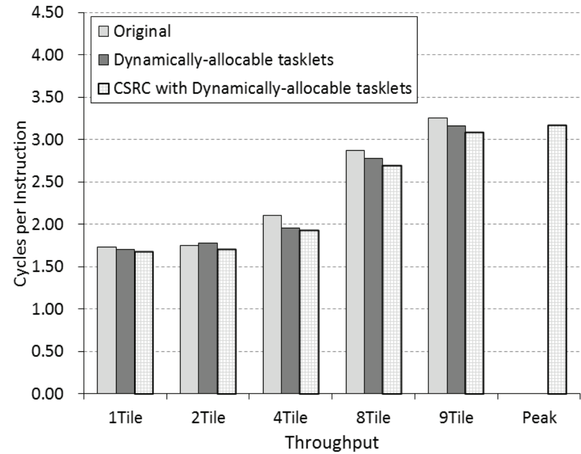

Figure 19. CPI trend comparisons

Finally, from the CPU utilization breakdown of different privilege levels as shown in Figure 20, both guest and hypervisor parts obtain lower CPU utilization when applying CSRC methods. CPU utilization at the same throughput (9-tile) reduces significantly from 92% to 81% by controlling the context switch rate. The hypervisor part contributes 6% and the guest part contributes the other 5%. As discussed above, CPU utilization reduction of guest part is primarily caused by the lower CPI and hypervisor part is caused by both PL and CPI.

To sum up, integrated with the two methods proposed in this section, the overall system throughput of SPECvirt_sc2010 is improved by 15%. The optimized methods obtains lower CPU utilization by reducing both CPI and PL, benefited from less number of context switches. The results show that, current frequency of context switch is so high that impair the server consoli-

dation's performance, which can be partially solved by fine-grained scheduling rate controller. There are still more considerations for scheduler in virtualization, such as to reduce memory and cache footprint, to be aware of running tasks in running VMs, to fairly and efficiently allocate and utilize the underlying hardware resources and so on.

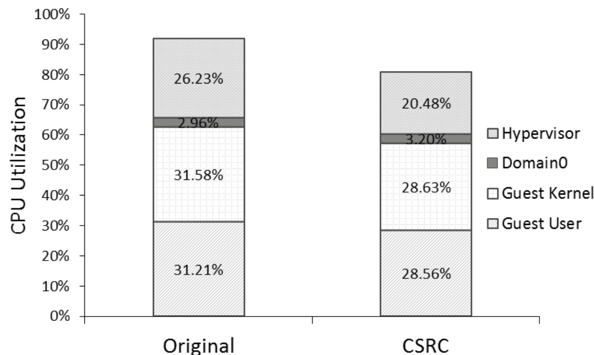

Figure 20. CPU utilization breakdown comparisons at same throughput (9-tile)

7. Related Work

Analyzing and improving virtual consolidation performance is a popular topic in the research community. There are many studies performed to improve virtualization's performance [29] [36] [31] [27]. In software, Liao et al. [29] [36] proposed some scheduling optimizations to improve I/O performance. Besides software optimizations, hardware-assisted solutions, such as VMDq [31], self-virtualized devices [27] and SR-IOV [24] are proposed in virtualization to achieve high-performance.

Studies for better hypervisor resource scheduler [19] [25] [28] are covered in several previous works. Using Xen as hypervisor, Cherkasova et al. [25] studied the impact that three different schedulers have on the throughput of three I/O-intensive benchmarks. In addition to the Credit and SEDF schedulers, their study also addressed Xen's BVT [28] scheduler. They evaluated how a single instance of these applications was affected by the choice of scheduling policy. In effect, they evaluated how the scheduler divided the processing resources between the guest domain and the driver domain. Ongaro et al. [19] explored the relationship between domain scheduling in a hypervisor and I/O performance using multiple guest domains concurrently running different types of applications. In addition, their work examined a number of new and existing extensions to Xen's credit schedule targeted at improving I/O performance. Finally, their study showed that latency-sensitive applications perform best if they are not combined in the same domain with a computing intensive application, but instead are placed within their own domain.

Compared to previous research work leveraging simple workload and hardware, we conducted our measurements and analysis with the state of art hardware. The workload we used is also modified through complex industry benchmark, thus representative enough of real virtual consolidation environments. We believe our performance data and conclusion are more realistic and can be used as a reference for real IT product system design.

8. Conclusion and Future Work

In this paper, through analyzing performance scalability of a representative consolidation workload on the latest 2-way X86 multiple core architecture, we identify both challenges and opportunities for virtualization performance under server consolidation condition: i) large memory and cache footprint caused by high frequency context switch brings in notable bottlenecks, and ii) the current scheduler can be improved with a finer control algorithm. Regarding these, we propose two potential optimization opportunities: applying a dynamically-allocable tasklets method brings in great context switch number reduction and a context switch rate control prototype results in total 15% performance improvement and up to 50% acceleration of service response. However, there is still great headroom for improvement. In future work, a guest-level sensitive scheduler considering its internal workloads and with finer control on priorities is arguably better in allocating the hardware resources between virtual machines to achieve a better overall system level performance. Long term fairness between virtual machines should also be considered in the scheduler design.

Acknowledgements

We would like to thank Keir Fraser for his contributions to compose the dynamically-allocable tasklets algorithm for this study.

References

[1] Intel Corporation. Terascale computing. http://www.intel.com/research/platform/terascale/index.htm.

[2] Intel Corporation. Intel Develops Tera-Scale Research Chips. http://www.intel.com/pressroom/archive/releases/20060926corpb.htm.

[3] VMware. http://www.vmware.com/.

[4] Microsoft Virtual Server, http://www.microsoft.com/hyper-v-server/

[5] Kernel Based Virtual Machine http://www.linux-kvm.org/

[6] 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 Symposium on Operating Systems Principles (SOSP), Oct. 2003

[7] SPECvirt_sc2010 http://www.spec.org/virt_sc2010/

[8] P. ApparaoNewell, Towards Modeling & Analysis of Consolidated CMP Servers, Workshop on the Design, Analysis, and Simulation of Chip Multi-Processors (dasCMP), 2007

[9] Jeffrey P. Casazza, Redefining server performance characterization for virtualization benchmarking, Intel Technology journal August 2006

[10] I. M. Leslie, D. Mcauley, R. Black, T. Roscoe, P. T. Barham, D. Evers, R. Fairbairns, and E. Hyden. The Design and Implementation of an Operating System to Support Distributed Multimedia Applications. IEEE Journal of Selected Areas in Communications, 1996

[11] VMmark http://www.vmware.com/products/vmmark/

[12] 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

[13] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. War_eld, and M. Williamson. Safe hardware access with the Xen virtual machine monitor. In 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS), Oct 2004

[14] 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

[15] Selvamuthukumar Senthilvelan and Murugappan Senthilvelan. Study of content-based sharing on the xen virtual machine monitor http://www.cs.wisc.edu/~remzi/Classes/736/Spring2005/Projects/Muru-Selva/cs736-report.pdf.

[16] Wiegert, J., et al.: Challenges for Scalable Networking in a Virtualized Server. In: 16th International Conference on Computer Communications and Networks

[17] 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

[18] L. Cherkasova and R. Gardner. Measuring CPU overhead for I/O processing in the Xen virtual machine monitor. In USENIX Annual Technical Conference, Apr. 2005

[19] Diego Ongaro , Alan L. Cox , Scott Rixner, Scheduling I/O in virtual machine monitors, Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, March 05-07, 2008, Seattle, WA, USA

[20] Susan J. Eggers , Joel S. Emer , Henry M. Levy , Jack L. Lo , Rebecca L. Stamm , Dean M. Tullsen, Simultaneous Multithreading: A Platform for Next-Generation Processors, IEEE Micro, v.17 n.5, p.12-19, September 1997

[21] Xudong Zheng, Jiangang Duan, Shameem F Akhter, Zhidong Yu, Hui Lv, A Consolidation Workload Characterization Study on Modern Platform, CMG'09

[22] UHLIG, R., NEIGER, G., RODGERS, D., SANTONI, A. L., MARTINS, F. C. M., ANDERSON, A. V., BENNETT, S. M., KAGI, A., LEUNG, F. H., AND SIMTH, L. 2005. Intel Virtualization Technology. IEEE Computer. 38, 5, 48–56

[23] Yaozu Dong, Zhao Yu, Greg Rose: SR-IOV Networking in Xen: Architecture, Design and Implementation. Workshop on I/O Virtualization 2008

[24] DONG, Y., YANG, X., LI, X., LI, J., TIAN, K., AND GUAN, H. 2010. High Performance Network Virtualization with SR-IOV. In Proceeding of the 16th IEEE International Symposium on High-Performance Computer Architecture (HPCA'10). IEEE, Bangalore, India, 271-280

[25] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the Three CPU Schedulers in Xen. ACM SIGMETRICS Performance Evaluation Review, 35(2):42–51, 2007

[26] Credit Scheduler.

http://wiki.xensource.com/xenwiki/CreditScheduler

[27] RAJ, H., AND SCHWAN, K. 2007. High performance and scalable I/O virtualization via self-virtualized devices. In Proceeding of the 16th international symposium on high performance distributed computing (HPDC'07). ACM, Monterrey, CA, 179-188

[28] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose scheduler. In Proceedings of the 17th ACM SOSP, 1999.

[29] GUO, D., LIAO, G., AND BHUYAN, L. N. 2009. Performance characterization and cache-aware core scheduling in a virtualized multi-core server under 10GbE. In Proceeding of 2009 IEEE International Symposium on Workload Characterization (IISWC'09). IEEE, Austin, TX, 168-177

[30] Yaozu Dong, Xiaowei Yang, Xiaoyong Li, Jianhui Li, Kun Tian, Haibing Guan. High performance network virtualization with SR-IOV. HPCA'2010. pp.1~10

[31] SANTOS, J. R., TURNER, Y. JANAKIRAMAN, G., AND PRATT, I. 2008. Bridging the gap between software and hardware techniques for I/O virtualization, In Proceeding of the USENIX Annual Technical Conference (USENIX'08). USENIX, Boston, MA, 29-42.

[32] Hui Lv, Xudong Zheng, Zhiteng Huang, Jiangang Duan, Tackling the Challenges of Server Consolidation on Multi-Core Systems. IISWC'2010

[33] Open-iscsi project http://www.open-iscsi.org/

[34] J. Calandrino and J. Anderson. On the design and implementation of a cache-aware multicore real-time scheduler. In Proceedings of the 21st Euromicro Conference on Real-Time Systems, July 2009.

[35] Arcangeli, Andrea ; Eidus, Izik ; Wright, Chris: Increasing memory density by using KSM. http://www.kernel.org/doc/ols/2009/#19-28. Version: 2009

[36] LIAO, G., BHUYAN, L. N., WU, W., YU, H., AND KING, S. R. 2010. A new TCB cache to efficiently manage TCP sessions for Web servers. In Proceeding of the 6th ACM/IEEE Symposium on Architecture for Networking and Communication Systems (ANCS'10). ACM, San Diego, CA, 1-10.

[37] Jun Nakajima, Qian Lin, Sheng Yang, Min Zhu, Shang Gao, Mingyuan Xia, Peijie Yu, Yaozu Dong, Zhengwei Qi, Kai Chen, Haibing Guan: Optimizing virtual machines using hybrid virtualization. SAC 2011: 573-578

[38] A. J. Bernstein, Program Analysis for Parallel Processing, IEEE Trans. on Electronic Computers". EC-15, pp. 757–62, 1966

[39] X. Zhang, A. E. Eichenberger, Y. Luo, K. O'Brien , K. O'Brien, Exploiting Parallelism with Dependence-Aware Scheduling, In Proceedings of the 18th International Conference on Parallel Architecture and Compilation Techniques (PACT'09) (Raleigh, NC, USA, Sept. 2009), ACM, pp. 193–202