This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE Globecom 2010 proceedings.

XenTune: Detecting Xen Scheduling Bottlenecks for Media Applications

Min Lee*, A. S. Krishnakumar†, P. Krishnan†, Navjot Singh†, Shalini Yajnik†
*Georgia Institute of Technology, Atlanta, GA 30332
†Avaya Labs, Basking Ridge, NJ 07920
minlee@gatech.edu, {ask, pk, singh, shalini}@avaya.com

***Abstract*— Virtualization provides enormous economic and ecological benefits by enabling server consolidation and supporting low-cost green data centers. A key component of the hypervisor in a virtualized system is the *scheduler*. A typical scheduler provides parameters to influence the hypervisor's behavior. Specifically, an application's performance on the popular open-source Xen virtualization platform can be influenced by tuning its scheduler behavior using the *weight*, *cap*, and *processor pinning* variables. However, determining which parameters to tune and how to do that is non-trivial. In this paper, we introduce XenTune, a *monitoring tool* for the credit scheduler in Xen that helps in understanding application behavior in scheduler terms and assists in determining scheduler parameters. We demonstrate how the tool is used with application domains – specifically a media application domain. We demonstrate experimental results using a real work-load that shows the considerable benefits of correct scheduler parameter choices. XenTune had helped in the design of a recently proposed scheduler *S* and in this paper we show how scheduler *S* interacts with the media application to optimize its performance.**

***Index Terms* — Xen, virtualization, schedulers, media applications, performance tuning.**

I. INTRODUCTION

Recent advances in virtualization technology [1][2][3][4][5] have made it more suitable for general purpose adoption. In data centers, server consolidation to reduce cost, space and power has been a driving force behind the success of the technology. Different market segments are now seriously evaluating the use of virtualization to host their infrastructure and solutions. Enterprise IP telephony applications (especially media-based applications) pose unique challenges to virtualization technology due to their soft real-time performance requirements. Virtualization technology has improved in terms of performance in the last few years but still faces significant performance challenges when it comes to real-time applications. Therefore, it is important to determine the performance bottlenecks of hosting such applications in virtualized environments. Xen [3] is a popular open-source platform for virtualization, and is the basis for our study here.

Even though significant amount of effort has been dedicated to optimizing the performance of virtual environments [6][7][8], there are still some areas that remain as bottlenecks and pose challenges to the adoption of the technology in certain domains, e.g. enterprise telephony. The key factors affecting the performance of such applications are scheduling and network I/O performance. Research efforts such as [9][10][11] study the impact of virtual machine (VM) scheduling on I/O virtualization performance and highlight the need for improved scheduling techniques that help I/O bound VMs while maintaining fairness. Understanding how real-time applications like media servers, that otherwise work adequately in a non-virtualized environment, behave in a virtualized deployment is critical to the adoption of virtualization for enterprise IP telephony. Our work presents insights into the behavior of such applications and provides results that will be useful in measuring how application performance is impacted by scheduler decisions, especially for media-based applications.

The motivation for our work was the poor performance of IP telephony media servers under Xen. Our analysis pointed to the Xen credit scheduler being the root cause of the performance bottleneck. Our main contribution in this paper is a *monitoring tool* called XenTune that assists a user in understanding application behavior in scheduler terms. Specifically, the default Xen scheduler has per-processor queues for domains/VMs, algorithms to update the VM status in a queue and move the VMs between queues eventually scheduling them on a processor. (This is explained in more detail in Section II.) The scheduler accepts certain user-specified parameters (like VM *weights* and *caps*, processor pinning, etc.) to help guide its behavior. However, determining these parameters is mostly left to the user with knowledge of the VM and the scheduler, and is a complicated task – leading most users to leave the parameters at default settings, or use a hit-and-trial method to set them. Our monitoring tool uses hooks provided by the *xentrace* [12][13] utility to collect relevant statistics about how a VM is being treated by the Xen scheduler. These statistics vividly demonstrate what scheduler parameters can be manipulated to achieve better performance. As we will see, in the case of a media application, the tool predicts the limitation in changing certain parameters and, in effect, certain limits of the credit scheduler itself. This has led us to design a new scheduler ***S*** [14] that introduces the notion of *laxity* and provides significantly better performance for near real-time VMs without impacting the quality of non real-time VMs. We believe that a monitoring tool that allows such analyses to be conducted, and as a part of the analysis suggests scheduler parameter modifications will be very useful to the end-user. In this paper we describe the components of the monitoring tool, its usage, and our results with a media application VM.

Section II discusses the Xen credit scheduler in more detail. In Section III, we describe the target media application. The main components of XenTune are detailed in Section IV. Section V uses XenTune to evaluate the behavior of the media

* This work was done when Min Lee was visiting Avaya Labs.

978-1-4244-5637-6/10/$26.00 ©2010 IEEE

application. In Section VI, we study the interaction of the media application with the new scheduler that we built with the benefit of insights from the monitoring tool. We conclude with Section VII and discuss some future work.

II. Scheduling in Xen

A. Xen

Server consolidation to reduce cost, space and power has been a driving force behind the success of virtualization. Virtualization allows multiple servers to run on the same physical hardware without interfering with each other. A thin layer called hypervisor or Virtual Machine Monitor (VMM) runs on top of the hardware and provides virtual hardware interfaces to the VMs.

In case of Xen, the hypervisor runs at the highest privilege level and controls the hardware. Virtual machine instances are also called *domains* in Xen. A privileged domain called Dom0 and other non-privileged guest domains called DomU run above the hypervisor like an application runs on an OS. Dom0 is a management domain that is privileged by Xen to directly access the hardware and it manages the initiation/termination of other domains.

The hypervisor virtualizes physical resources such as CPUs and memory for the guest domains. Most of the non-privileged instructions can be executed by the guest domains natively without the intervention of the hypervisor. However, privileged instructions will generate a trap into the hypervisor. The hypervisor validates the request and allows it to continue. The guest domain can also use *hypercalls* to invoke functions in the hypervisor. For this, the guest OS needs to be ported to use the functionality and this porting is called *para-virtualization*. The performance of applications in the guest domain is dependent on the methods used for I/O and CPU scheduling. We briefly describe these below.

As the I/O devices are shared across all the guest domains, the hypervisor controls access to them. Xen provides a delegation approach for I/O via a split device driver model where each I/O device driver called the *backend* driver runs in Dom0. The DomU has a *frontend* driver that communicates with the backend driver via shared memory. The split I/O model requires implementation in Dom0 and DomU. The I/O processing requires Dom0 and the guest domain to be scheduled on the CPU. I/O intensive applications are sensitive to the scheduler used. We outline below the default scheduler – credit scheduler – used by Xen.

B. Credit Scheduler

The Xen credit scheduler is designed to ensure that each virtual machine and/or a virtual CPU (vcpu) gets a fair share of the physical CPU resource. In the credit scheduler, each domain is assigned a parameter called the weight, and CPU resources (or *credits*) are distributed to the virtual CPUs of the domains in proportion to their weight at the end of each accounting period (default 30ms). At the end of the accounting period the vcpu priority (or task priority) is recalculated based on the credits. When a task exhausts its credits, its priority is set to *over*. When it still has credits, it is set to *under*. To accommodate low latency, the scheduler supports a *boost* priority, where *blocked tasks waiting for an I/O event* are boosted upon receiving an event/interrupt [9]. A task gets boosted for a very short time (less than one tick, typically, 10ms) after which it reverts to normal priority. This simple optimization of task boosting is good in the delegation model for an I/O-bound domain such as Dom0 that handles I/O for all domains, wakes up a lot and finishes its work within a very short time. The scheduler is invoked (tickled) when a task enters the boost state, and it will pre-empt the current running task (unless it is running in boost priority) and run the boosted task immediately [9]. The priorities are implemented as different regions of the same run-queue, with one run-queue per physical processor. The tasks are ordered by priority – *boost*, *under* and *over*. When some task enters the run-queue, it is simply inserted into the end of the corresponding region of the queue, and the scheduler picks up the task from the head of the queue to execute.

The credit scheduler also performs load balancing between cores in a symmetric multiprocessing (SMP) environment. When a physical CPU becomes idle or has no boosted or under priority tasks, it checks its peer CPUs' queues to see if there is a strictly higher priority task. If so, it steals that higher priority task.

C. Parameters to Tune Scheduler Behavior

Xen provides some parameters to the users/applications to control its scheduling behavior. One of the main parameters for controlling the scheduler is processor pinning. Pinning involves explicitly assigning CPU resources to vcpus. In order to reserve a particular physical CPU for a vcpu, the vcpu is pinned to the physical CPU and all the other vcpus are pinned away from this physical CPU. In addition to pinning, the default credit scheduler allows additional ways of tuning the scheduler – weights and caps. Weights allow the user to specify the proportion of the CPU to be assigned to a given virtual machine. A default weight of 256 is given to each virtual machine. A cap value optionally fixes the maximum amount of CPU a virtual machine will be able to consume, even though the host system may have idle CPU cycles.

III. Media Applications and Performance

A media application is responsible for processing incoming media streams and either playing them as they arrive or transcoding them for retransmission. Packet losses and/or delays in packet processing can adversely affect the quality of a media stream. Media streams can originate from any live source e.g. IP phone, media server, audio from a radio station, video camera feed etc. Our emphasis is on the performance of a media server running on a Xen-based platform that hosts a virtualized enterprise IP telephony system with multiple virtual machines, including virtual machines handling call signaling, media processing and cpu-intensive tasks. The media server used in this work takes a real-time protocol (RTP) stream from the caller, uses a standard jitter buffer, and re-encodes the stream for the callee. We exercise the system at 4 calls per second using the G.711 codec with a call hold time of 30s for a maximum of 240 streams (120 callers and 120

callees) incoming into the media server. These parameters were chosen to load the media server close to its capacity. We sample 1 in 4 calls and measure the call quality with the ITU-T Perceptual Evaluation of Speech Quality (PESQ) metric [15]. The PESQ value is based on comparing the original waveform from the caller against the final one received at the callee. It ranges from a minimum of 0 (bad) to maximum of 4.5 (best). A value of 4 and above is considered to be toll-quality voice. We then plot a graph of PESQ values.

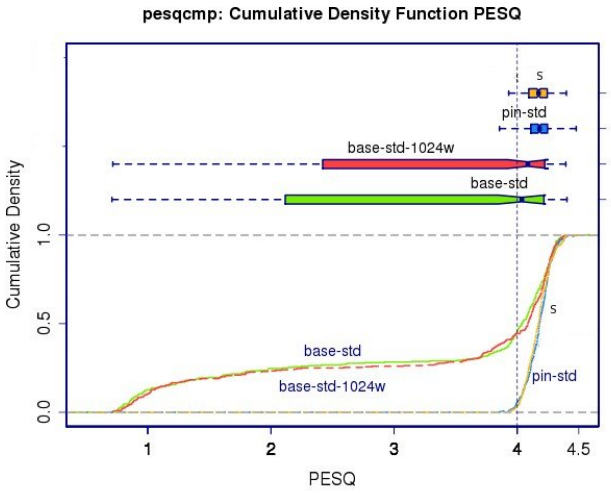

Figure 1: PESQ voice quality metric for media application

Figure 1 shows PESQ graph for a set of configurations – (a) default credit scheduler (*base-std*), (b) credit scheduler with an increased weight of 1024 for media domain (*base-std-1024w*), (c) credit scheduler with media domain pinned to a CPU (*pin-std*), and (d) the laxity-based scheduler ***S*** (*s*) [14]. The graph in the figure shows PESQ values in two forms. The upper half of the graph shows the PESQ values as boxplots for the four configurations. The lower half shows the cumulative density functions of the PESQ values. From the graph, we see that the *base-std* configuration shows almost 50% streams with poor voice quality. Giving higher weight to the media domain (*base-std-1024w*) has negligible impact on the voice quality. In case of pinned configuration (*pin-std*), the media server is given a dedicated physical CPU and other domains are moved away from that physical CPU. This configuration provides all the needed resources for the media processing and shows all the streams with good voice quality. The performance of scheduler ***S*** is comparable to the pinned configuration. To understand the issues involved, we explore the workings of the scheduler using our xentrace-based monitoring tool described in the next section.

IV. XENTUNE: MONITORING TOOL

We built a xentrace-based monitoring tool, called XenTune, which interacts with the Xen scheduler via the xentrace hooks. Xentrace will be discussed in more detail in Section IV.A. We added custom events to xentrace to get additional scheduler metrics discussed later in this section.

Figure 2 shows the architecture of the monitoring tool. The Xentrace Events module receives the events from xentrace and provides them to the Metrics Analysis Engine. The Metrics Analysis Engine processes and evaluates the various scheduler metrics collected. The metric data can then be either displayed using the graphical user interface (UI) or provided to the Scheduler Parameter Tuning module. The Scheduler Parameter Tuning module calculates the optimal parameters for the scheduler to get the best performance from the VMs. The Scheduler Parameter Tuning module feeds this information back to the Xen Scheduler.

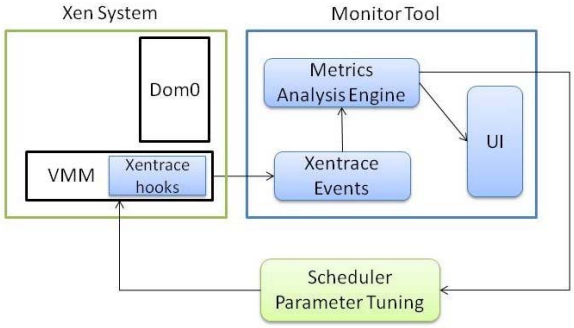

Figure 2: XenTune monitoring architecture

In this work, we focus on describing the Xentrace Events module and the Metrics Analysis Engine. The user-interface and the automated Scheduler Parameter Tuning modules are being currently implemented and will be discussed in a more detailed paper later.

A. *Xentrace*

Xentrace is a light-weight event logging tool, packaged as part of a Xen deployment. It is used to capture trace data from the Xen hypervisor. It can be used as a means to identify the location of time spent in the hypervisor, track hypervisor calls and OS interactions as well as debug hypervisor crashes. It provides a default set of events that can be monitored by the user. For each event instance, xentrace dumps the following data in binary format:

CPU TSC EVENT D1 D2 D3 D4 D5

where CPU is the processor number, TSC is the record's timestamp (the value of the CPU cycle counter), EVENT is the event ID and D1 through D5 are the trace data specific to the event.

The event classes provided by xentrace include, among others, scheduler, Dom0 and memory. Since this work focuses on the scheduling aspects, the scheduler events are of most interest here. The scheduler event class provides a set of generic events that are applicable across all types of schedulers. Such events include VM state changes (sleep, wait, block, yield, shutdown, addition and removal) and timer events. Each event can have up to five data items associated with it.

B. *Custom Xentrace Scheduler Events*

In particular, the scheduler event class in xentrace provides an event called TRC_SCHED_SWITCH_INFNEXT, which is invoked when a domain is just about to be scheduled. The event produces data regarding the time that the domain had to wait before it got scheduled and the time for which it will get scheduled. In addition, TRC_SCHED_SWITCH_INFPREV is

another event that provides data on when a domain stops running and the time for which it ran. However, this information is not sufficient for our purpose. In order to determine the effect of weights and pinning on the application, in addition to the time spent waiting in the run queue, we need to identify the priority (under, over or boost) of the vcpu while it was waiting in the run queue and the time spent in the run queue with each priority level. We added our own events to the credit scheduler to extract this information. For efficiency reasons, we collect this information while the scheduler is running and only dump it out to xentrace every 10 seconds.

As discussed earlier, a run queue associated with a physical CPU is logically treated as three queues – boost, under and over. Each part of the run-queue has its own characteristics:

- Boost run-queue: A vcpu gets placed in the run-queue with a boost priority when it gets woken up due to some event, e.g. a packet arrival. I/O processing domains such as Dom0 usually get put in a boost state when a packet arrives on the network interface, whereas a CPU-bound job which does not block will not usually get into a boost state. Since boost is the highest priority a vcpu can have, the scheduling latency while the vcpu is in the boost state is usually low.
- Under run-queue: A vcpu gets placed in this run-queue when it has some work to do and has enough credits to get scheduled.
- Over run-queue: A vcpu get placed in the run-queue with over priority when it has work to do, but no credit to get scheduled. A long wait time in this run-queue usually means that the vcpu is short on CPU resource.

Our main aim during monitoring is to determine the workings of the scheduler and the interaction of the scheduler with the application workload. In order to do this, we collect the following metrics for each priority level (boost, under and over) using custom events in xentrace:

- *Sched_count*: Number of times a vcpu entered the run-queue with a given priority level
- *Sched_latency*: Average amount of time a vcpu waited in the run-queue with a given priority before being picked up by the scheduler
- *Sched_timeslice*: Average amount of time a vcpu got scheduled on a CPU by the scheduler when picked up from the run-queue with a given priority
- *Sched_totaltime*: Total scheduled time with a given priority. This is the product of the sched_count and the sched_timeslice.

Using the above scheduling metrics, we characterize the behavior of the application workload and determine the values for the scheduler parameters suitable to meet the needs of the application workload.

V. TRACING THE MEDIA APPLICATION

In this section, we trace our media application using XenTune and determine the effect of the tunable scheduler parameters on the performance of the media application. The media domain runs a media processing application, like an IP telephony gateway, a media server or a conferencing server. Such applications are highly I/O intensive and, at the same time, they also need CPU cycles to process media packets.

A. Operating with default scheduler

Figure 3 shows the graphs of the four scheduler metrics – sched_count, sched_latency, sched_timeslice, and sched_totaltime for the media application domain. The first graph (sched_count) gives the number of times the vcpu for the domain entered the run-queue with a given priority. The second graph (sched_latency) shows the average amount of time it waited in the run-queue with the given priority. The third graph (sched_timeslice) shows the average amount of time it ran on the CPU after the scheduler picked it up from the run-queue. The last graph shows the total time spent on the CPU with a given priority.

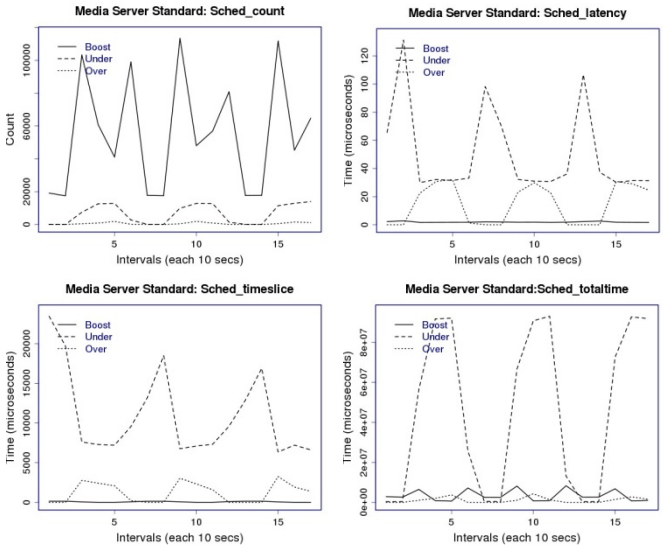

Figure 3: Scheduler metrics for media application with default scheduler parameters – (a) sched_count, (b) sched_latency, (c) sched_timeslice, (d) sched_totaltime

Figure 3(a) shows that the media application gets into the run-queue with a boost priority a large number of times. This is to be expected as the media application is an I/O intensive application. It sleeps while waiting for packets to arrive and when a packet comes in, the hypervisor puts the domain in boost state in the run-queue. However, as seen from the graph (c) sched_timeslice from the boost priority is very low. This implies that the vcpu gets scheduled for a very short period of time and may get de-scheduled immediately after picking up the packet. Graph (d) shows that most of the scheduled time spent on the CPU is from the under priority state, which means that the vcpu had enough credits/weights to do its processing job. However, the sched_timeslice from under state (graph (c)) is not very large (10-20 msecs) which implies that the vcpu gets preempted by scheduler within 10-20 msecs of being scheduled and does not get enough CPU time for full

packet processing. In contrast, let us take an example of a computationally-intensive domain which spends most of its time doing CPU operations. Figure 4 below shows the graphs for such a computational domain. As seen from the figures, the computational domain enters the run-queue with over priority most of the time, unlike the media server case in Figure 3(d). The sched_count and the sched_totaltime from over state are much larger than the corresponding values from the under and boost state, indicating that the domain lacks CPU resources and could benefit from more credits.

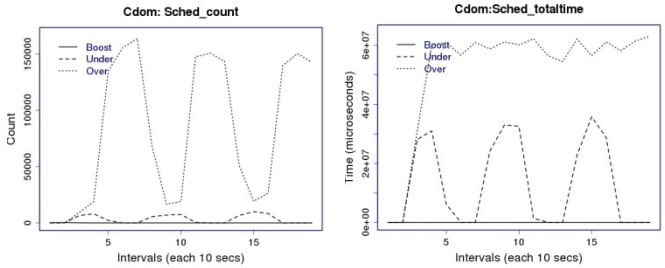

Figure 4: Scheduler metrics for computational domain

This foregoing discussion indicates that *increasing the weight for the media domain should not significantly improve its performance*. In the next section, we use our tool to verify this prediction.

B. Using weights to control the scheduler

The graphs in Figure 5 show the scheduler metrics for the case when the media domain is given a higher credit/weight.

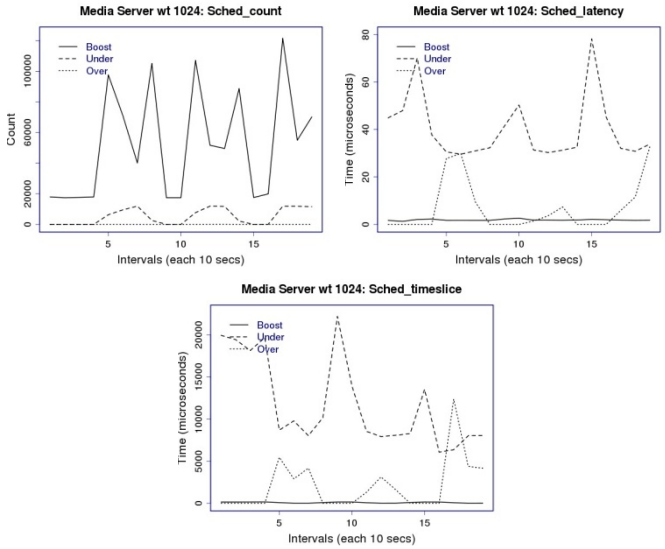

Figure 5: Scheduler metrics for media application with higher weights to the media domain

As predicted earlier, increasing the weight of the domain does not improve the performance. The increase in weight reduces the sched_latency and does give the domain more CPU resources to process a packet. However, as seen from the graphs, the sched_timeslice from the under state is still in the range of 10-20 msecs. Hence, domain gets scheduled for a very short time before being preempted by the scheduler due to other domains waiting for their share of the CPU. This leads to insufficient "timely" CPU for packet processing and hence no improvement in PESQ performance as shown in Figure 1. This analysis suggests that dedicating CPU resources to the VM should help – which can be done through pinning.

C. Using pinning to control resource allocation

Next, we see if pinning the vcpu of a domain to a physical CPU and pinning all the other domains away from this physical CPU solves the resource problem.

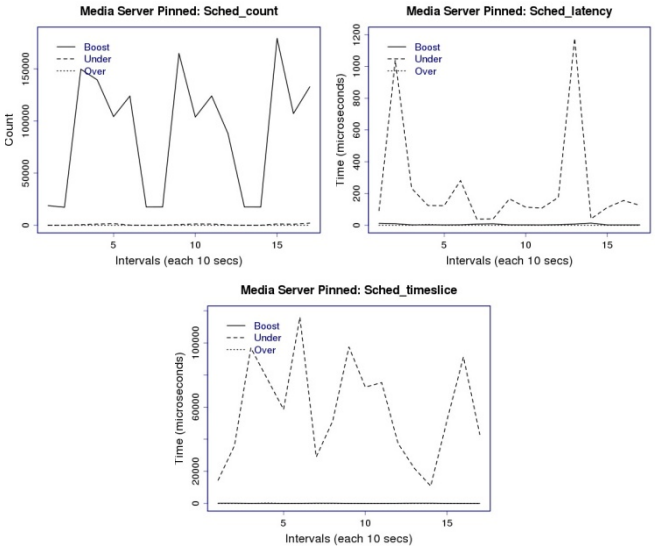

Figure 6: Scheduler metrics for the pinned configuration

Figure 6 shows the scheduler metric graphs for the pinned configuration. As seen from the graphs, reserving resource through pinning does help in improving sched_timeslice metric for the domain. In the pinned configuration, there are no other domains competing for the CPU. As seen in the graph above, each time the media server domain gets scheduled, it gets about 60-100 msec of CPU to do its job. Whenever a packet comes in, the domain is either already scheduled and immediately picks up the packets or gets scheduled and processes the packets. The larger amount of allocated CPU time slice leads to timely packet processing and an improved PESQ score (as shown in Figure 1).

VI. USING AN ENHANCED SCHEDULER

The XenTune experiments in Section V indicate that real-time media processing domains: (a) need CPU for processing packets, (b) need CPU in a "timely" manner for handling the delivered packet. The experiments also show that the default credit scheduler (even with increased weights) is insufficient for meeting the timeliness requirements of the media applications. Pinning helps in securing timely CPU resources, however, it is non-work conserving (i.e. the CPU may not be fully utilized) and hence not beneficial for the system as a whole. We used the knowledge gathered through XenTune to develop a new scheduler ***S*** that targeted the requirements of such real-time media processing domains [14]. This scheduler uses a notion of laxity to determine the real-timeness of a domain and places the domain in the run-queue based on its laxity value. In effect, it moves the media server ahead in the

run-queue. Details of the scheduling algorithm and its performance are discussed in [14]. As we see from Figure 1, the media server under scheduler ***S*** has similar performance to the pinned case. One would expect that moving the media server ahead in the queue effectively gives it a "pinning" effect. In this section, we study the interaction of scheduler ***S*** with the media application using our monitoring tool. *Interestingly, we see that the scheduler **S** does well for a different reason than the pinned case*.

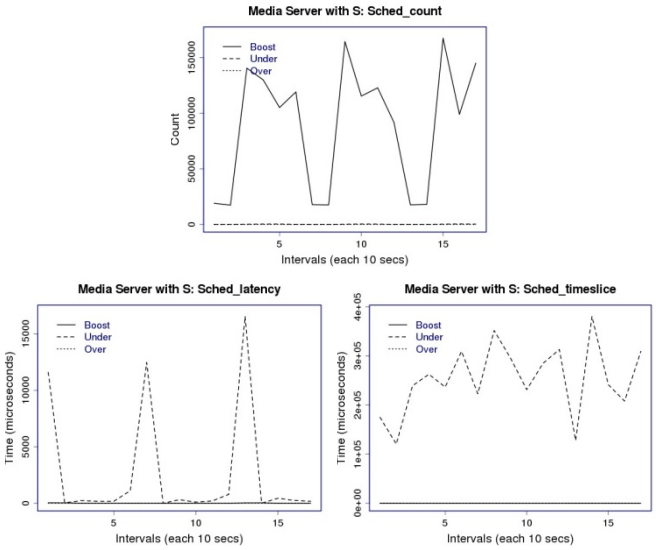

Figure 7: Scheduler metrics for the laxity-based scheduler ***S***

The results using XenTune with scheduler ***S*** are shown in Figure 7. With scheduler ***S*** the media domain shares the CPU with other domains, however due to the lower laxity value assigned to the domain, each time it enters the run-queue, it gets placed in the run-queue at a position where it can meet its laxity target. The sched_latency for ***S*** is higher than the pinned case because unlike the pinned case, the media domain shares the CPU with other domains. The side effect of the higher latency is that a number of packets collect while the media domain is waiting to be scheduled. However, the sched_timeslice metric for the domain each time it is scheduled from the under priority queue is as high as 200-300msecs. This allows the domain to pick up multiple packets each time it is scheduled and process them in a timely manner, thereby yielding a high PESQ score.

VII. CONCLUSIONS

This work presents XenTune, a monitoring tool that allows us to study the impact of scheduling parameters on various application domains. The tool shows that the default credit scheduler in Xen, even with increased weights, is insufficient for meeting the demands of media applications. It also points out the desired characteristics needed for a new scheduler to meet these requirements. The insights from the tool were useful in design of a new soft real-time scheduler. While the focus was on media applications, the tool is general enough to be applicable to other domains. Future work on the tool involves automating the scheduler parameter changes and adaptive tuning of the scheduler based on the running workload.

REFERENCES

[1] "Kernel-based Virtual Machine (KVM) for Linux." http://www.linux-kvm.org.

[2] VMware Inc., "VMware ESX and VMware ESXi." http://www.vmware.com/products/vi/esx.

[3] Xen 3.4." http://www.xen.org/files/ Xen_3_4_Datasheet.pdf."

[4] Intel, "Intel virtualization technology (Intel VT)." http://www.intel.com/technology/virtualization/.

[5] AMD, "AMD virtualization technology (AMD-V)." http://www.amd.com/virtualization.

[6] P. Apparao, R. Iyer, X. Zhang, D. Newell, and T. Adelmeyer, "Characterization & analysis of a server consolidation benchmark," in VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, pp. 21-30, 2008.

[7] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the Xen virtual machine environment," in VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments, pp. 13-23, 2005.

[8] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in Xen," in ATC '06: Proceedings of USENIX '06 Annual Technical Conference, pp. 2-2, 2006.

[9] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in virtual machine monitors," in VEE '08: Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, pp. 1-10, 2008.

[10] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam, "Xen and co.: communication-aware CPU scheduling for consolidated Xen-based hosting platforms," in VEE '07: Proceedings of the 3rd international conference on Virtual execution environments, pp. 126-136, 2007.

[11] G. Liao, D. Guo, L. Bhuyan, and S. R. King, "Software techniques to improve virtualized I/O performance on multi-core systems," in ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pp. 161-170, 2008.

[12] D. Gupta, R. Gardner, and L. Cherkasova, "XenMon: QoS Monitoring and Performance Profiling Tool", Technical Report HPL-2005-187, HP Labs, 2005.

[13] Xentrace "How to enable Xentrace Logging and Format with Xentrace_Format", http://support.citrix.com/article/CTX121583

[14] M. Lee, A.S. Krishnakumar, P. Krishnan, N. Singh, and S. Yajnik, "Supporting Soft Real-Time Tasks in the Xen Hypervisor", in VEE '10: Proceedings of the 6th international conference on Virtual execution environments, 2010.

[15] PESQ, http://www.itu.int/rec/T-REC-P.862/en, ITU-T Recommendation P.862, "Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs".