Supporting Soft Real-Time Tasks in the Xen Hypervisor

Min Lee\textsuperscript{1}, A. S. Krishnakumar\textsuperscript{2}, P. Krishnan\textsuperscript{2}, Navjot Singh\textsuperscript{2}, Shalini Yajnik\textsuperscript{2}

\textsuperscript{1}Georgia Institute of Technology  
Atlanta, GA, USA  
minlee@cc.gatech.edu

\textsuperscript{2}Avaya Labs  
Basking Ridge, NJ, USA  
{ask, pk, singh, shalini}@avaya.com

The results described in this paper are elaborated upon in:  

This document presents some of the key results of the research.

ABSTRACT

Virtualization technology enables server consolidation and has given an impetus to low-cost green data centers. However, current hypervisors do not provide adequate support for real-time applications, and this has limited the adoption of virtualization in some domains. Soft real-time applications, such as media-based ones, are impeded by components of virtualization including low-performance virtualization I/O, increased scheduling latency, and shared-cache contention. The virtual machine scheduler is central to all these issues. The goal in this paper is to adapt the virtual machine scheduler to be more soft-real-time friendly.

We improve two aspects of the VMM scheduler – managing scheduling latency as a first-class resource and managing shared caches. We use enterprise IP telephony as an illustrative soft real-time workload and design a scheduler S that incorporates the knowledge of soft real-time applications in all aspects of the scheduler to support responsiveness. For this we first define a laxity value that can be interpreted as the target scheduling latency that the workload desires. The load balancer is also designed to minimize the latency for real-time tasks. For cache management, we take cache-affinity into account for real time tasks and load-balance accordingly to prevent cache thrashing. We measured cache misses and demonstrated that cache management is essential for soft real time tasks. Although our scheduler S employs a different design philosophy, interestingly enough it can be implemented with simple modifications to the Xen hypervisor’s credit scheduler. Our experiments demonstrate that the Xen scheduler with our modifications can support soft real-time guests well, without penalizing non-real-time domains.

Keywords Virtualization, Xen, Enterprise telephony workloads, Server consolidation, Laxity

\* This work was done when Min Lee was an intern at Avaya Labs.
Overview

- Real time and Xen
- Default Credit Scheduler
- Experiments
  - Instrumentation
- Laxity-based scheduler
  - Laxity (A)
  - Boost with event (B)
  - Simple Load Balance (L)
  - Cache-aware Real-time load balancing (LL)
- Result
- Conclusion

Real time and Xen

- Near real-time applications pose challenges
  - low-performance virtualization I/O
  - scheduling latency
  - shared-cache contention

- Scheduler is central to all these issues
  - scheduling latency as a first-class resource
  - managing shared caches
Deployment of Generic Enterprise Telephony system.

- **Hardware**
- **Xen Hypervisor**
- **Local Area Network**
- **Backbone Network**
- **IP Endpoint**
- **Call-C and Sip-S** (Call setup/tear down)
- **Gateway/Media Server** (stream encode/decode)

**Default Credit Scheduler**

- Consumes credits to run
  - Over priority
  - Under priority

- Every 30ms, distribute credits based on weights
  - E.g. 20% CPU to VM 1, 40% CPU to VM 2
  - Proportional distribution
  - Default weight (256)
Default Credit Scheduler

- Boost up waking-up tasks

  CPU0 — VCPU4 — VCPU2 — VCPU7 — VCPU1 — VCPU0 — VCPU5

- New incoming (waking-up) task
  - Probably due to external event (packet arrival)
  - E.g. Domain 0

  VCPU3

- Only one time boost-up (<10ms)
Default Credit Scheduler

- Insufficient support for real-time tasks

- RT-tasks (Media-S) competes with other tasks
  - Treated as just CPU-bound task
  - Missing its deadlines

Credit Crisis

- Credit scheduler
  - Good for CPU task

- Boost priority
  - Boost waking-up task
  - Short time (<10ms)
  - Good for Dom0!

- Bad for Media task
  - Need both CPU & Low latency!
  - Competing with others
Credit Crisis

- Now on, Laxity-based scheduler is introduced.
  - Incremental (4 components)
- Experiments
  - Instrumentation
- Laxity-based scheduler
  - Laxity (A)
  - Boost with event (B)
  - Simple Load Balance (L)
  - Cache-aware Real-time load balancing (LL)

Experiments (Deployment)

- Enterprise Telephony system
  - Media server – ‘Media-S’
  - Signaling server – ‘Call-C, Sip-S’
  - Other VMs
    - Dom0
    - Cdom [Computational domain]
    - Two more domains [Licensing, management]
- Two flavors
  - Standard (Cdom is empty)
  - Cpuload (Cdom has 4 cpu-bound tasks)
- PESQ
  - Quality of voice
  - Major metric
Experiments (Setup)

- Dell 2950 server
  - 2 quad-core Xeon processors
  - 4 GB of RAM
- 2 cores from each socket
  - So private cache
- 4cps, sample one out of 4 calls
  - G.711, 20ms packetization
  - 30 sec hold time
  - Max 240 streams through Media-S
- PESQ
  - Compare the reference with stream from Media-S

Instrumentation

- Custom events for xentrace
- CPU utilization
- Worst/average wait time
  - Each priority
- Time slice length
  - Each priority
- Cache misses
  - By xenoprof
Reading the plots

- **PESQ**
  - Quality of voice
  - 0 (bad) ~ 4.5
  - 4.0 (toll quality)
- **Cumulative Density**
- **Boxplots**
  - Min/Max
  - 25%, 75% percentile
  - Median

Performance with default credit scheduler

- More weights (512, 1024)
  - didn’t improve much
- Only configuration it worked – pinning
- Media-S/Call-C requires significant CPU
  - Pinned Media-S/Call-C to CPU0,1 respectively
  - Pinned others to CPU2,3
  - Dom0 is floating
- Underutilizing CPU!
Laxity (A)

- Kind of priority?
  - Laxity!
  - Target scheduling latency or deadline
  - E.g. less than 5ms wait time in the run queue
  - Parameter specified by user

- Real-time task
  - Has laxity value

- Non-real-time task
  - Doesn’t have laxity value (or conceptually infinite laxity)

Implementation of laxity

- Where to insert real-time tasks to meet their deadline
  - In over priority, laxity value is ignored.
Prediction of wait time

- Each VCPU maintains an expected run time
  - Previous time slice length
- The amount of CPU time it utilized in its previous run
- Works reasonably well
  - Min/max
  - 25%,75% percentile
- More sophisticated formula can be used

Laxity Result

- Improved PESQ
Boost with event (B)

- Want to boost-up tasks receiving external event.
  - Packet arrival.

- But credit scheduler only boost up waking-up task!
  - Media-S in run queue doesn’t get boosted

- So, we boost it up in the run queue

Simple Load Balance (L)

Default Credit scheduler

(1) Over or idle task?
(2) Peek peer’s Q
(3) Steal higher priority task
Simple Load Balance (L)

Laxity-based scheduler (Simple Load Balance)

1. Under, over or idle task?
2. Peek peer’s Q
3. Steal higher priority task
4. If same priority
   a. If mine is real time task, don’t steal
   b. If peer’s is real time task, steal
   c. Both non-real-time, compare entrance time into queue
      - with some delay (2ms)

(1) This effectively distributes real-time tasks over CPUs
(2) Also prevents starvation
Simple Load Balance (L)

Laxity-based scheduler (Simple Load Balance)

(1) Kicks out non-real-time task if they waited for some time

Cache-aware Real-time load balancing (LL)

- Good, but...
  - Ping-ponging tasks trashes cache
- Bind RT-tasks to CPU
  - Fix RT-tasks to its initial CPU
  - Disable load-balancing for RT-tasks
- Unbalance of multiple RT-tasks
  - Need new load-balancer for RT-tasks
Cache-aware Real-time load balancing (LL)

- Don’t steal peer’s real-time tasks in L
  - Prefer hot cache (to low wait time)
- New x-sec-load balancer
  - Balance of real time task’s cpu utilization via bin-packing

![Diagram showing CPU utilization of real-time tasks on CPU0 and CPU1]

Cache-aware Real-time load balancing (LL)

- Improved PESQ

![Graph showing cumulative density function PESQ with different curves for base-sto, pinstd, and ALL-std]
Result (CPU utilization)

- We’re fully utilizing CPUs!

Result (Various laxity value)

- Lower laxity → more realtime → better quality

Various laxity values - standard configuration
Result (Various laxity value)

- Lower laxity → more realtime → better quality

Result (Boost with event)

- Some impact
Result (Load balance)

- Load balancing is essential for multicore

![PESQ improvement through load balancer in standard configuration](image)

Result (Cache misses)

- We should consider cache

![Cache misses for standard configuration](image)
Conclusion

- New soft real time aware scheduler for Xen
  - Better support for real time applications without penalizing non-real time tasks
  - Fully utilizing CPU resources
  - Timeliness requirement expressed by laxity

- Instrumentation to help design and measure performance
  - Can also be used for other purposes