Virtual Machines and Real-Time

Luca Abeni

luca.abeni@santannapisa.it

May 31, 2021

Real-Time in VMs???

- Running real-time applications on an RTOS is not a problem...
- ...But, can real-time applications run in virtual machines?
  - Real-Time in Virtual Machines??? But... Why?
- Component-Based Development
  - Complex applications: sets of smaller components
  - Both functional and temporal interfaces
- Security (isolate real-time applications in a VM)
- Easy deployment; Time-sensitive clouds

Real-Time in VMs

- Real-Time applications running in a VM?
  - As for OSs, two different aspects

Real-Time in VMs

- Real-Time applications running in a VM?
  - As for OSs, two different aspects
    - Resource allocation/management (scheduling)
  - CPU allocation/scheduling: lot of work in literature

Real-Time in VMs

- Real-Time applications running in a VM?
  - As for OSs, two different aspects
    - Latency (host and guest)
  - Latencies not investigated too much (yet!)

Real-Time in VMs

- Real-Time applications running in a VM?
  - As for OSs, two different aspects
    - Resource allocation/management (scheduling)
    - Latency (host and guest)
  - CPU allocation/scheduling: lot of work in literature
  - Latencies not investigated too much (yet!)
- Virtualization: full hw or OS-level
  - Containers: real-time performance of the host kernel
  - Hw virtualization: hypervisors (example: KVM or Xen) can introduce latencies!

Latency

- Latency: measure of the difference between the theoretical and actual schedule
  - Task τ expects to be scheduled at time t ...
  - ... but is actually scheduled at time t'
  - $\Rightarrow$ Latency $L = t' - t$
- The latency L can be accounted for in schedulability analysis
  - Similar to what is done for shared resources, etc...
  - Strange “shared resource”: the OS kernel (or the hypervisor)

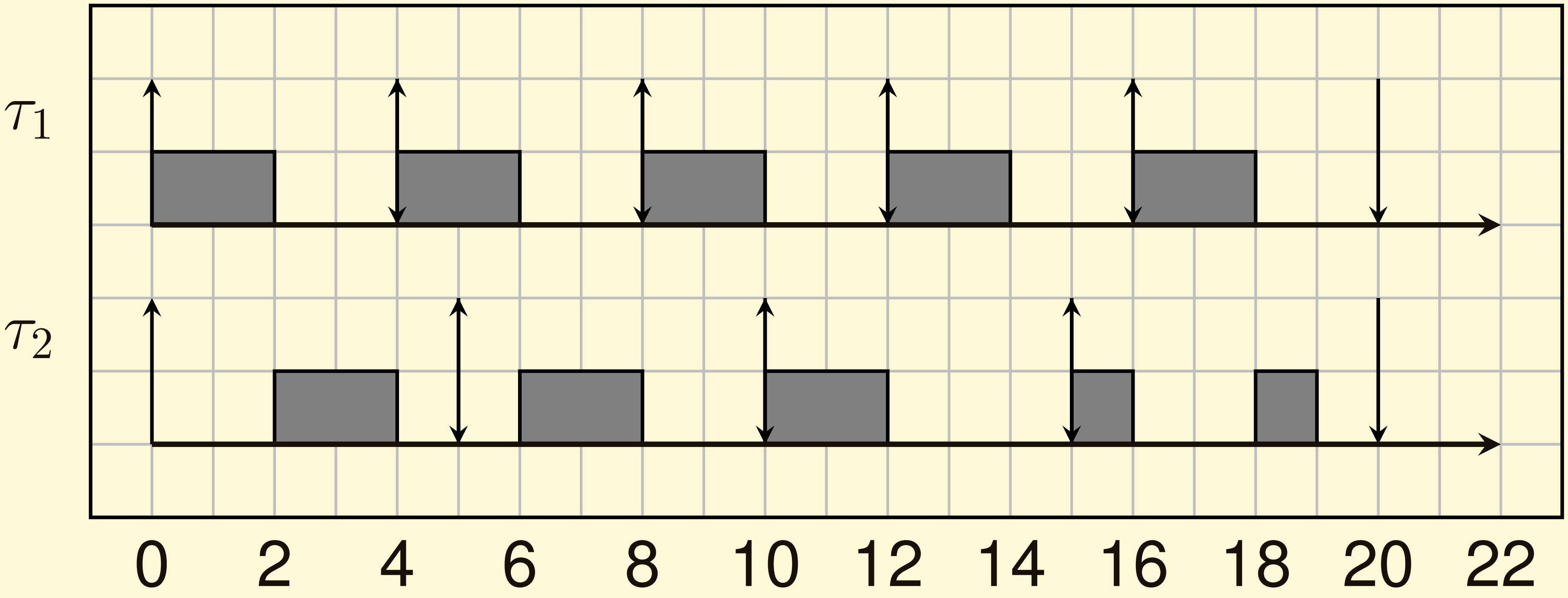
Example: Periodic Task

- Consider a periodic task

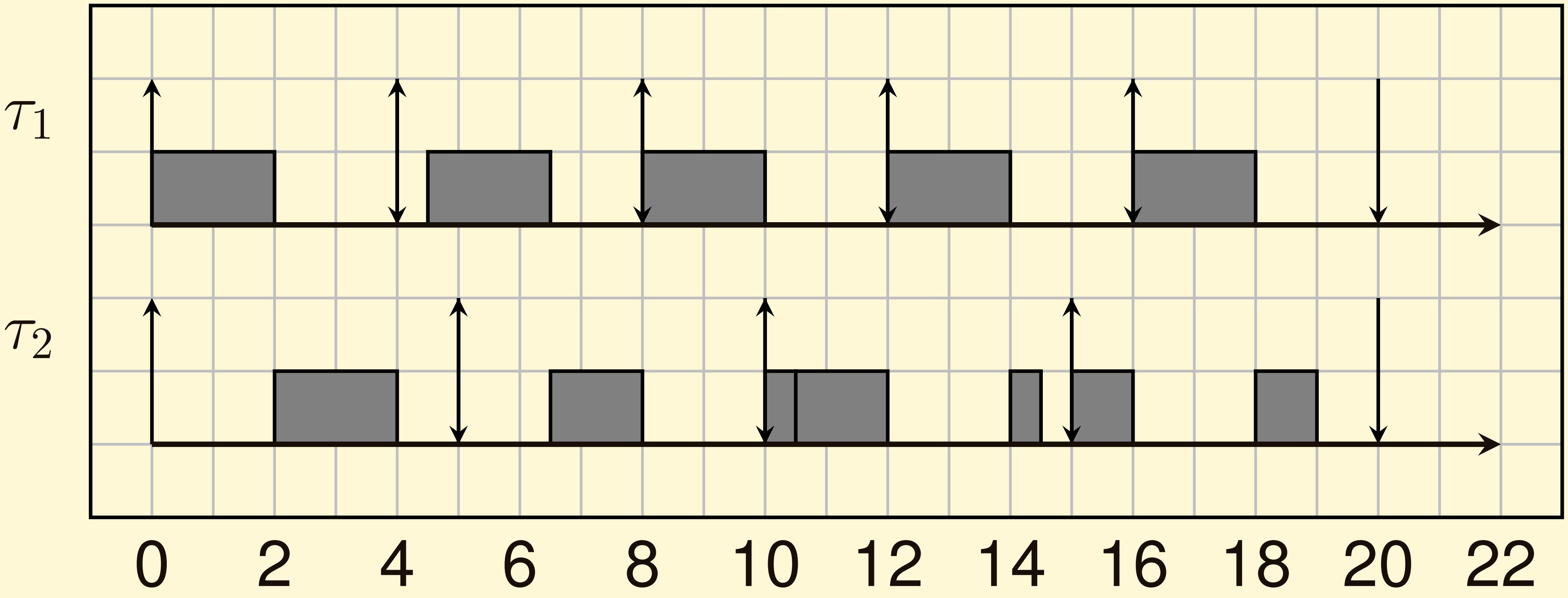
```
/* ... */
while(1) {
  /* Job body */
  clock_nanosleep(CLOCK_REALTIME,
                  TIMER_ABSTIME, &r, NULL);
  timespec_add_us(&r, period);
}
```

- The task expects to be executed at time `r` $(= r_0 + jT)$...
- ...But is sometimes delayed to $r_0 + jT + \delta$

Theoretical Schedule



Actual Schedule



- What happens if the 2^{nd} job of τ_1 arrives a little bit later???
  - The 2^{nd} job of τ_2 misses a deadline!!!

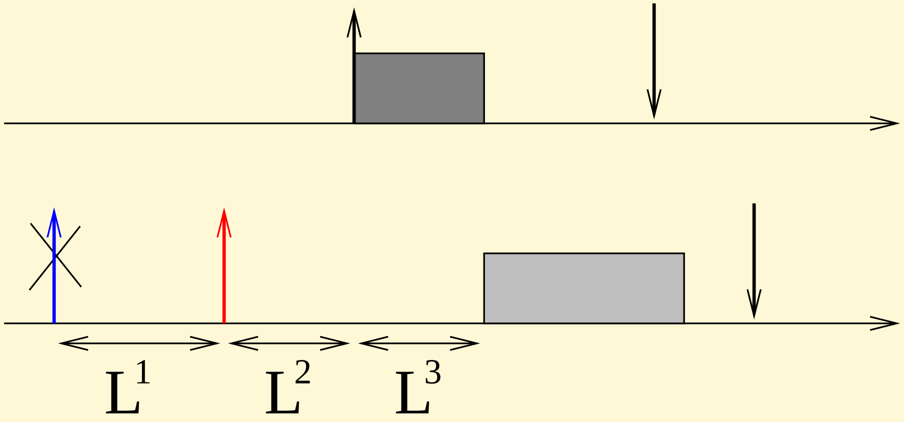
Effects of the Latency

- Upper bound for L? If not known, no schedulability analysis!!!
  - The latency must be *bounded*: $\exists L^{max} : L < L^{max}$
- If L^{max} is too high, only few task sets result to be schedulable
  - The worst-case latency L^{max} cannot be too high

Sources of Latency — 1

- Task: stream of jobs (activations) arriving at time r_j
- Task scheduled at time $t' > r_j \rightarrow$ Delay $t' - r_j$ caused by:

1. Job arrival (task activation) signaled at time $r_j + L^1$
2. Event served at time $r_j + L^1 + L^2$
3. Task actually scheduled at $r_{i,j} + L^1 + L^2 + I$



Sources of Latency — 2

- $L = L^1 + L^2 + I$
- I: interference from higher priority tasks
  - Not really a latency!!!
- L^2: *non-preemptable section latency* L^{np}
  - Due to non-preemptable sections in the kernel (or hypervisor!) or to deferred interrupt processing
- L^1: delayed interrupt generation
  - Generally small
  - Hardware (or virtualized) timer interrupt: *timer resolution latency* L^{timer}

Latency in Linux

- Tool (`cyclictest`) to measure the latency
  - Periodic task scheduled at the highest priority
  - Response time equal to execution time (almost 0)
- Vanilla kernel: depends on the configuration
  - Can be tens of milliseconds
- Preempt-RT patchset (`https://wiki.linuxfoundation.org/realtime`): reduce latency to less than 100 microseconds
  - Tens of microseconds on well-tuned systems!
- So, real-time on Linux is not an issue
  - Is this valid for hypervisors/VMs too?

What About VM Latencies?

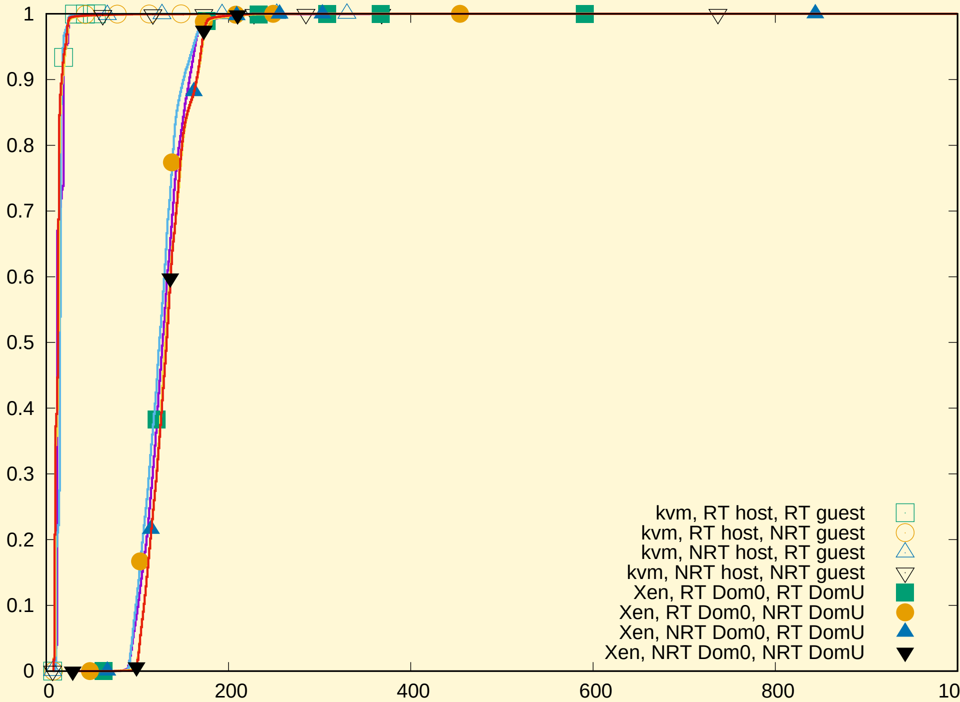
- Hypervisor: software component responsible for executing multiple OSs on the same physical node
  - Can introduce latencies too!
- Different kinds of hypervisors:
  - Xen: bare-metal hypervisor (*below* the Linux kernel)
    - Common idea: the hypervisor is small/simple, so it causes small latencies
  - KVM: hosted hypervisor (Linux kernel module)
    - Latencies reduced by using Preempt-RT
    - Linux developers already did lot of work!!!

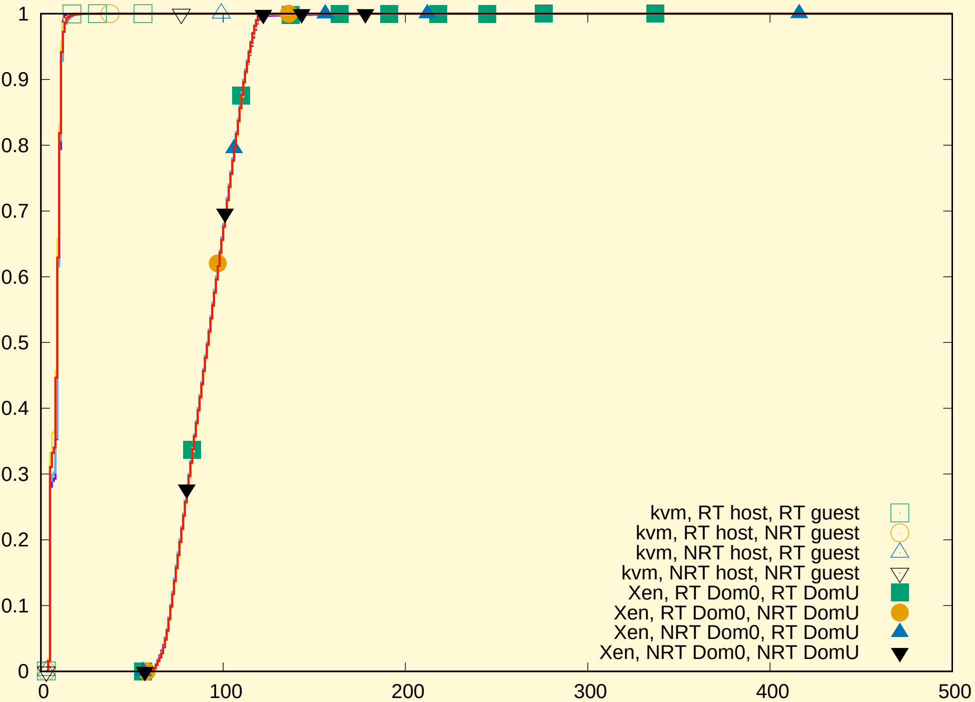
Hypervisor Latency

- Same strategy/tools used for measuring kernel latency
- Idea: run `cyclictest` in a VM
  - `cyclictest` process ran in the guest OS...
  - ...instead of host OS
- `cyclictest` period: $50 \mu s$
- "Kernel stress" to trigger high latencies
  - Non-real-time processes performing lot of syscalls or triggering lots of interrupts
  - Executed in the host OS (for KVM) or in Dom0 (for Xen)
- Experiments on multiple x86-based systems

Hypervisor Latencies

Intel Core Duo





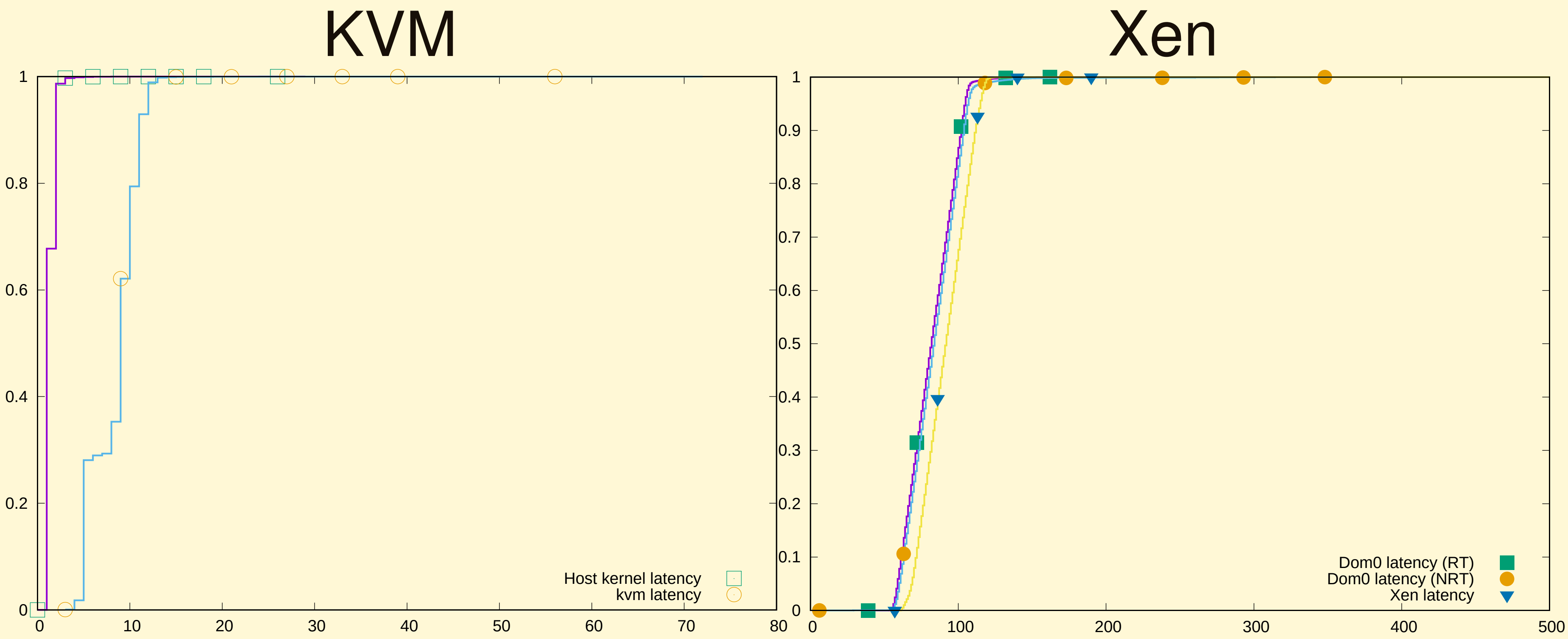
Intel Core i7

Worst Cases

Kernels	Core Duo		Core i7	
	Xen	KVM	Xen	KVM
NRT/NRT	$3216\mu s$	$851\mu s$	$785\mu s$	$275\mu s$
NRT/RT	$4152\mu s$	$463\mu s$	$1589\mu s$	$243\mu s$
RT/NRT	$3232\mu s$	$233\mu s$	$791\mu s$	$99\mu s$
RT/RT	$3956\mu s$	$71\mu s$	$1541\mu s$	$72\mu s$

- Preempt-RT helps a lot with KVM
  - Good worst-case values (less than $100\mu s$)
- Preempt-RT in the guest is dangerous for Xen
  - Worst-case values stay high

Hypervisor vs Kernel



- Worst Cases:
  - Host: $29\mu s$
  - Dom0: $201\mu s$ with Preempt-RT, $630\mu s$ with NRT

Investigating Xen Latencies

- KVM: usable for real-time workloads
- Xen: strange results
  - Larger latencies in general
  - Using Preempt-RT in the guest increases the latencies?
- Xen latencies are not due to the hypervisor’s scheduler
  - Repeating the experiments with the null scheduler did not decrease the experienced latencies

Impact of the Kernel Stress

- Experiments repeated without “Kernel Stress” on Dom0
  - This time, using Preempt-RT in the guest reduces latencies!
  - Strange result: Dom0 load *should not* affect the guest latencies...

Kernels	Core Duo		Core i7	
	Stress	No Stress	Stress	No Stress
NRT/NRT	$3216\mu s$	$3179\mu s$	$785\mu s$	$1607\mu s$
NRT/RT	$4152\mu s$	$1083\mu s$	$1589\mu s$	$787\mu s$
RT/NRT	$3232\mu s$	$3359\mu s$	$791\mu s$	$1523\mu s$
RT/RT	$3956\mu s$	$960\mu s$	$1541\mu s$	$795\mu s$

Virtualization Mechanisms

- Xen virtualization: PV, HVM, PVH, ...
  - PV: everything is para-virtualized
  - HVM: full hardware emulation (through qemu) for devices (some para-virtualized devices, too); use CPU virtualization extensions (Intel VT-x, etc...)
  - PVH: hardware virtualization for the CPU + para-virtualized devices (trade-off between the two)
- Dom0 kernel does not affect results; focus on guest kernel

Guest Kernel	PV	PVH	HVM
NRT	$661\mu s$	$1276\mu s$	$1187\mu s$
RT	$178\mu s$	$216\mu s$	$4470\mu s$

What’s up with HVM?

- HVM uses qemu as *Device Model* (DM)
  - Qemu instance running in Dom0
  - Used for boot and emulating some devices...
  - ...But somehow involved in the strange latencies!!!
- Scheduling all qemu threads with priority 99, the worst-case latencies are comparable with PV / PVH!!!
  - High HVM latencies due to the Kernel Stress workload preempting qemu...
- Summing up: for good real-time performance, use PV or PVH!

Cyclictest Period

- Most of the latencies larger than cyclictest period...
- Are hypervisor's timers able to respect that period?
  - Example of timer resolution latency...
- So, let's try a larger period!
  - $500\mu s$ and $1ms$ instead of $50\mu s$
  - Measure timer resolution latency $\rightarrow$ no kernel stress
- Results are much better!
  - $P = 500\mu s$: worst-case latency $112\mu s$ (HVM), $82\mu s$ (PVH) or $101\mu s$ (PV)
  - $P = 1000\mu s$: worst-case latency $129\mu s$ (HVM), $124\mu s$ (PVH) or $113\mu s$ (PV)

Further Analysis

- Xen latencies seem to be mainly due to timer resolution latency
  - Turned out to be an issue in the Linux code handling Xen's para-virtualized timers
    - Linux jargon: "clockevent device"
  - Does not activate a timer at less than $100\mu s$ from current time (`TIMER_SLOP`)
- After reducing the timer slop, average latency smaller than $50\mu s$ even for cyclictest with period $50\mu s$
  - Still larger than KVM latencies (probably due to non-preemptable sections?)

Final Results

- Xen with a properly configured `TIMER_SLOP`:
  - Timer resolution latency reduced to almost 0
  - Non-preemptable section latency dependent on the virtualization technology
  - Worst-case latencies higly dependent on the hardware
    - Example: some old CPUs need to (trap and) emulate `rdtsc` $\Rightarrow 15\mu s$ additional latency
- Xeon CPU: $28\mu s$ with PVH, $72\mu s$ for PV (KVM is $44\mu s$)
- Core 2 CPU: $88\mu s$ for PV, $182\mu s$ for PVH (KVM is $71\mu s$)

Reproducible Results

- Results can be reproduced on your test machine
  - You just need some manual installation of KVM, Xen, etc...

`http://retis.santannapisa.it/luca/VMLatencies`

- Scripts to reproduce the previous experiments
  - Number depends on the hw, but the obtained figures are consistent with the previous results
- The other figures can be easily ontained modifying scripts / configuration files

Summing Up

- Latencies experienced in a VM (`cyclictest`)
  - KVM: Preempt-RT allows to achieve low latencies → usable for real-time
  - Xen: high latencies, Preempt-RT does not help, strange impact of the Dom0 load
- Xen behaves better when PV or PVH is used
  - Part of the latencies due to the DM (qemu running in Dom0)?
- Xen experiences a large timer resolution latency
  - Fixable by modifying the guest kernel

Latencies and Scheduling

- Most of the industrial work on real-time virtualization focused on latency reduction
  - Example: real-time KVM industrial solution based on vCPU pinning — No scheduling!!!
- Scheduling VMs is still needed to share hardware resources...
  - Bounded latencies are needed to have precise and accurate vCPU scheduling...
  - ...But appropriate scheduling algorithms are still needed!!!
- Advanced scheduling algoritms are useless if latencies are not bounded, and bounded latencies are useless if appropriate scheduling is not used!