Real-Time Virtual Machines: Theory and Practice

Luca Abeni
luca.abeni@santannapisa.it

ISTITUTO DI TECNOLOGIE DELLA COMUNICAZIONE, DELL'INFORMAZIONE E DELLA PERCEZIONE
Scuola Superiore Sant'Anna

Real-Time Applications

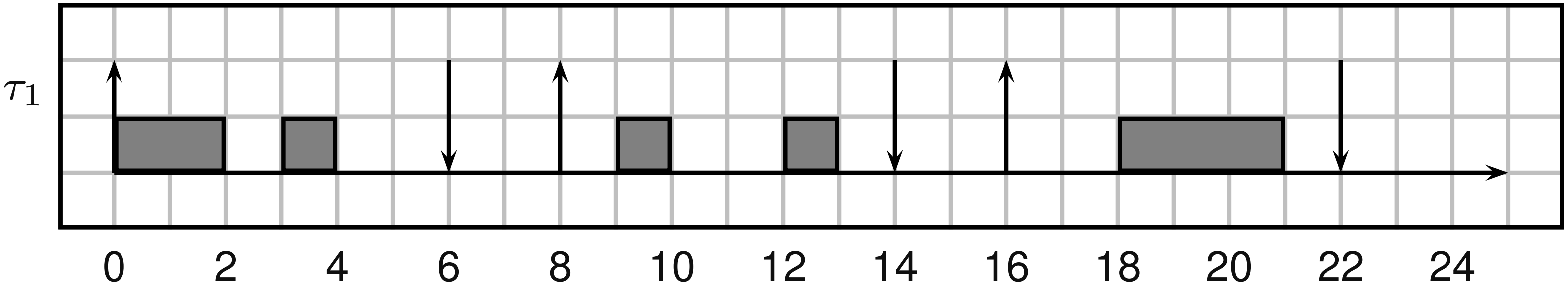
- Real-Time Application: The time when a result is produced matters
  - A correct result produced *too late* is equivalent to a wrong result (or to no result)
- What does "*too late*" mean, here?
  - Applications characterised by temporal constraints that have to be respected!
- Examples:
  - Control applications, autonomous driving, ...
  - But also infotainment, gaming, telecommunications, ...!!!

Temporal Constraints

- Temporal constraints are modelled through *deadlines*
  - Finish some activity before a time (deadline)
  - Generate some data before a deadline
  - Terminate some process/thread before a deadline
  - ...
- What happens if a constraint is not respected?
  - Simple: the application fails!

Real-Time Task

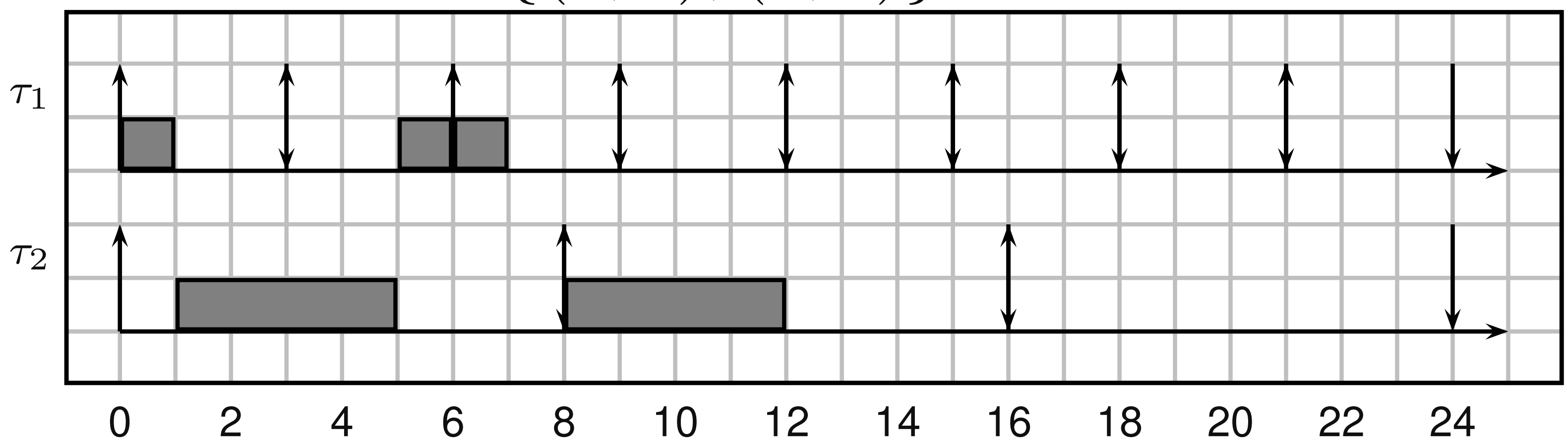
Task (process or thread): sequence of actions characterized by deadlines and maximum execution time. Example: periodic task with period 8 and maximum execution time 3.



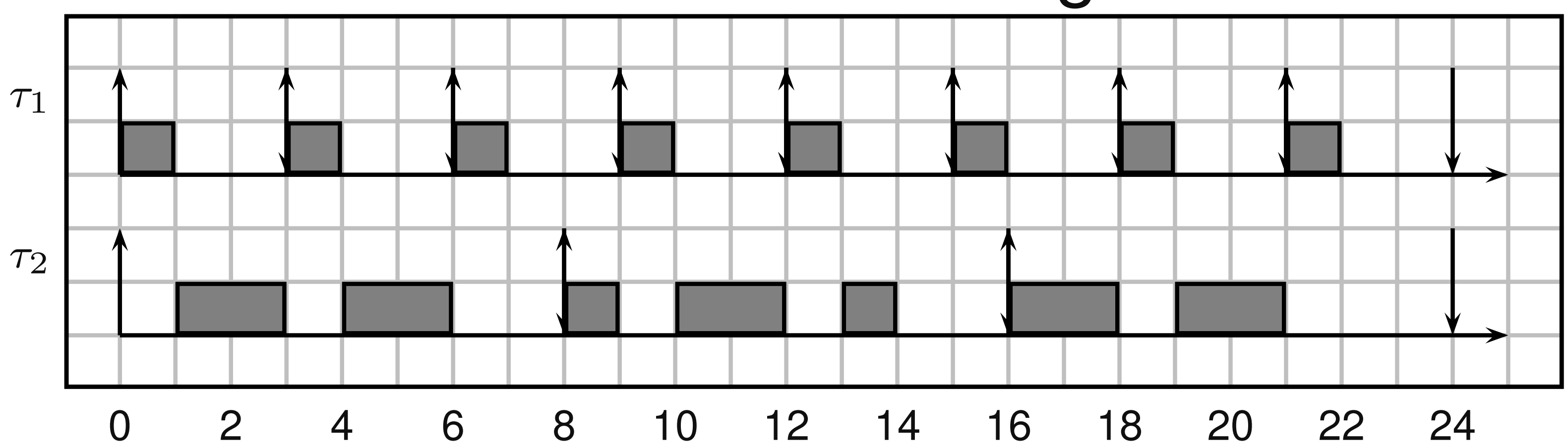
Note: while the first and the third activations execute for 3 time units the second one executes for only 2 time units.

RT Scheduling: Why?

- Taskset $\mathcal{T} = \{(1,3),(4,8)\}$: not schedulable by FCFS



- $\mathcal{T}$ is schedulable with other algorithms



The Scheduling Problem

- A real-time task τ_i is properly served if all jobs respect their deadline...
- ...Appropriate scheduling is important!
  - The CPU scheduler must somehow know the temporal constaints of the tasks...
  - ...To schedule them so that such temporal constraints are respected
- How to schedule real-time tasks? (scheduling algorithm)
- Is it possible to respect all the deadlines?
- Do commonly used OSs provide appropriate scheduling algorithms?

Fixed Priorities

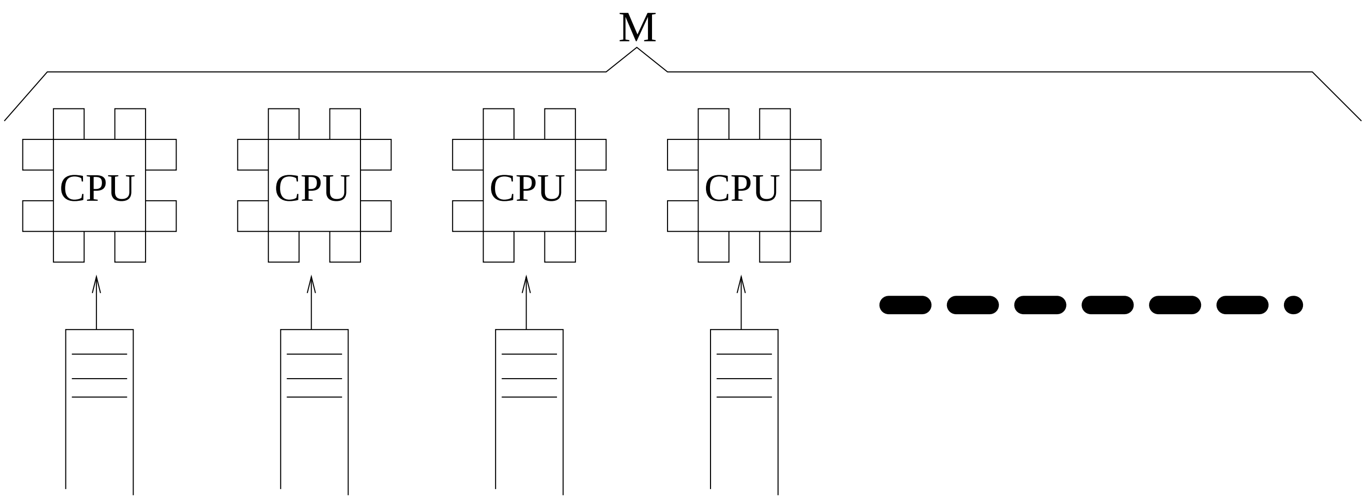
- Fixed-priority schedulers are often used for real-time tasks...
- ...Is this ok in general, or only in some cases?
  - Given a set of real-time tasks $\Gamma = \{\tau_i\}$, can a fixed priority scheduler allow to respect all the deadlines?
  - Is it possible to know in advance if some deadline will be missed?
  - How to assign the priorities?
- Example: if fixed priorities are enough, the `SCHED_FIFO` and `SCHED_RR` policies can be used!

Schedulability Analysis

- Given a set of tasks, is it possible to know in advance if deadlines will be missed?
  - A task is *schedulable* if it will not miss any deadline
    - Depends on the task, on the scheduler, on the other tasks, ...
- Schedulability test: mathematical test to check if τ will miss any deadline
  - Possible idea: compute the amount of time needed by all the tasks to respect all their deadlines in an interval (t_0, t_1) and compare with $t_1 - t_0$

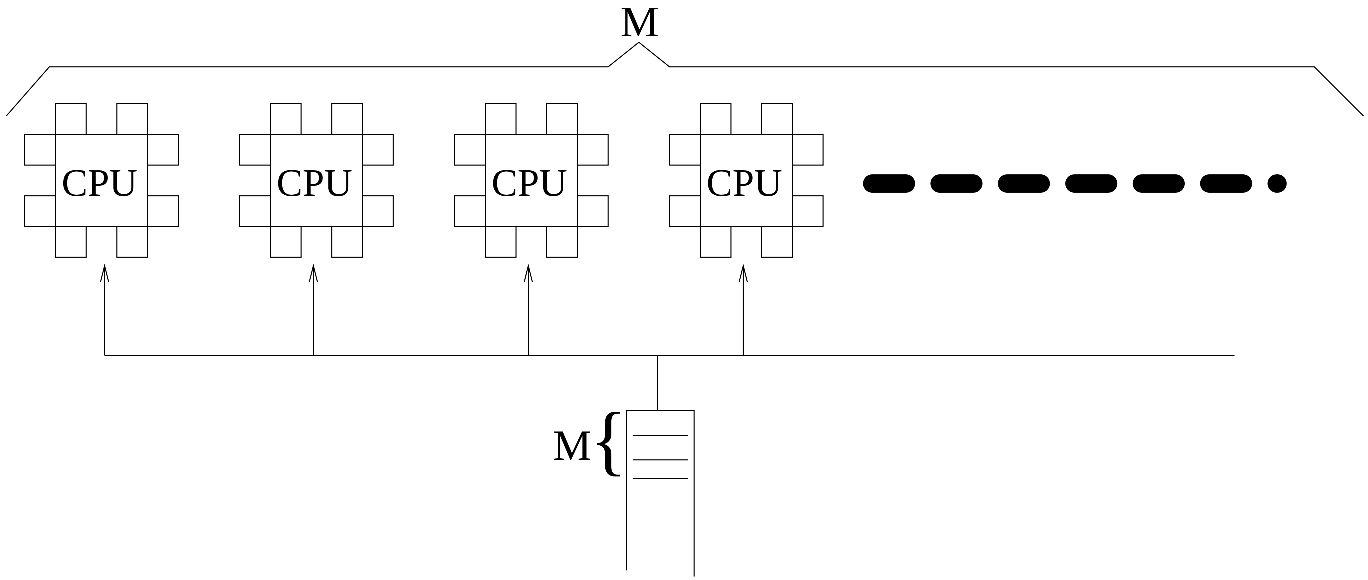
What About Multiple Cores?

- How to schedule tasks on multiple CPUs / cores?
  - First idea: partitioned scheduling
- Statically assign tasks to CPU coress
- Reduce the problem of scheduling on M cores to M instances of uniprocessor scheduling



Or...

- One single task queue, shared by M CPU coress
  - The first M ready tasks are selected
  - What happens using fixed priorities?
  - Tasks are not bound to specific CPUs
  - Tasks can often migrate between different CPUs
- Problem: UP schedulers do not work well!



Example: Fixed Priorities in Linux

- `SCHED_FIFO` and `SCHED_RR` use fixed priorities
  - They can be used for real-time tasks, to implement priority assignments from literature
  - Real-time tasks have priority over non real-time (`SCHED_OTHER`) tasks
- Difference between the two policies: visible when more tasks have the same priority
- The administrator can use `sched_setscheduler()` (or similar) to schedule real-time tasks
  - Might be a little bit impractical, but can be used
- However...

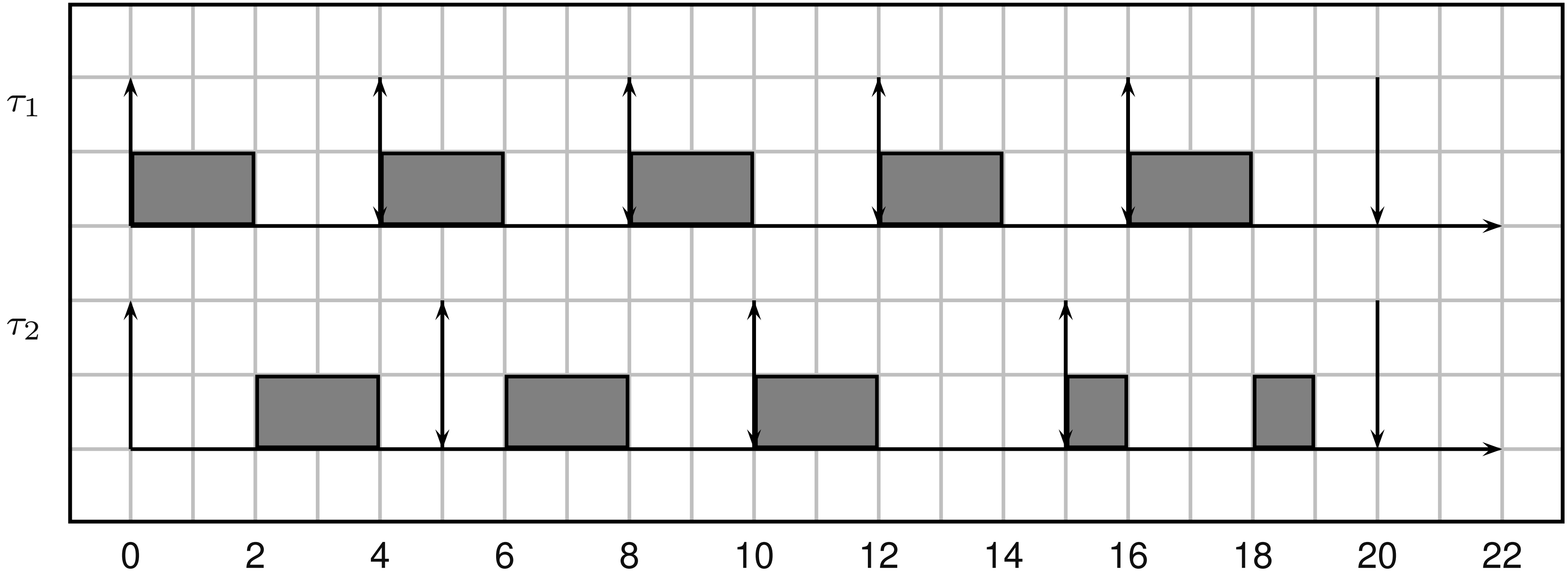
Periodic Task Example

- Consider a periodic task

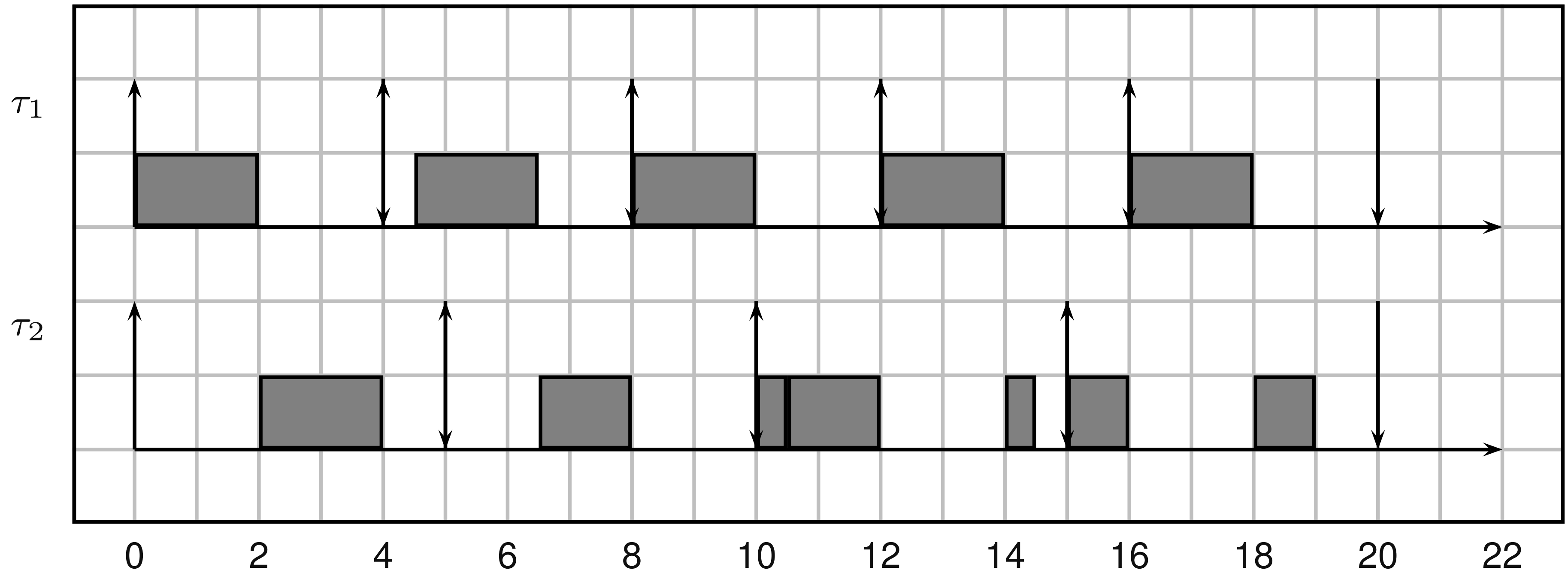
```
/* ... */
while(1) {
  /* Job body */
  clock_nanosleep(CLOCK_REALTIME,
                  TIMER_ABSTIME, &r, NULL);
  timespec_add_us(&r, period);
}
```

- The task expects to be executed at time $r = r_0 + jP$...
- ...But is sometimes delayed to $r_0 + jP + \delta$
  - Why? Because it executes on a real system, not in a simulator!!!

Theoretical Schedule



Actual Schedule



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

Theory vs Real Schedule

- The delay δ in scheduling a task is due to *kernel latency*
- Kernel latency can be modelled as a blocking time
- In real world, high priority tasks often suffer from blocking times coming from the OS (more precisely, from the kernel)
  - Why?
  - How?
  - What can we do?
- To answer the previous questions, we need to at the OS internals...

The OS Kernel

- Kernel: "core" of the OS, managing the hardware $\rightarrow$ can be non-preemptable
- System executing kernel code for a long time (long syscalls or heavy interrupt load) $\rightarrow$ long latencies!!!
- Possible Solutions:
  1. Insert explicit rescheduling points into the kernel
  2. Make the kernel preemptable (spinlocks protect critical sections) $\rightarrow$ sections holding a spinlock for too much time still create problems
  3. 1 + 2 $\rightarrow$ preemptable kernel + lock breaking
- Or, rewrite the kernel from scratch...

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 modelled as a so-called "blocking time" $\Rightarrow$ real-time theory knows how to handle it!
  - Analysis already developed for shared resources
  - Can be easily adapted to account for kernel latency

Effects of the Latency

- Upper bound for L? If not known, no schedulability tests!!!
  - 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
  - Large blocking time experienced by *all tasks*!
  - The worst-case latency L^{max} cannot be too high
- Real-Time OS (RTOS): OS provinding a low upper bound for the kernel latency!

Latency in Linux

- Tool (`cyclictest`) to measure the latency
- 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, scheduling real-time tasks in Linux is not an issue anymore...

Virtualization

- Virtualization: creation of a *virtual instance* of a computing system
  - Computer (PC, server, embedded board, ...)
  - Operating System
  - Storage device / other
- Separate / independent from the physical system(s) hosting it
- This mainly requires two activities:
  1. Pooling: consolidating possibly distributed resources into a single logical entity
  2. Isolation: giving the virtualized application a "virtual" private copy of the resources

Resource Pooling

- Set of multiple, possibly distributed, resources
- Single "virtual resource", that can be used to transparently access them
  - Pool of physical servers hosting VMs in a cloud; accessed by starting a VM $\Leftarrow$ load balancing
  - Pool of storage devices (disks, databases, ...) accessed as a single virtual storage device $\leftarrow$ I do not know where data are really stored...
  - ...
- Used for automatically distributing the load, for building powerful machines based on less powerful ones, for making computation independent on data placement, ...

Resource Isolation

- The usage of virtual resources must be controlled by the virtualization software
  - Example: applications running in a VM should not be able to access resources outside of the VM...
  - ...Nor to directly access physical resources!
  - ...
- Virtual resources should not even be distinguishible from physical ones
  - Example: applications running in a VM should have the impression to run on a physical machine...

Different Kinds of Isolation...

- Resource isolation can be used for different reasons
  - Security $\leftarrow$ reduce the impact of compromised subsystems
  - Application sandboxing $\leftarrow$ execute non-trusted software
  - Performance guarantees $\leftarrow$ isolate the performance of a component from interference of other components
  - ...
- Different kinds of requirements

Virtualized Resources

- Virtual Machine: efficient, isolated duplicate of a physical machine
  - Why focusing on *physical* machines?
  - What about abstract machines?
- Software stack: hierarchy of abstract machines
  - ...
  - Abstract machine: language runtime
  - Abstract machine: OS (hardware + system library calls)
  - Abstract machine: OS kernel (hardware + syscalls)
  - Physical machine (hardware)

Hardware Virtualization

- Can be full hardware virtualization or paravirtualization
  - Paravirtualization requires modifications to guest OS (kernel)
- Can be based on trap and emulate
- Can use special CPU features (hardware assisted virtualization)
- In any case, the hardware (whole machine) is virtualized!
  - Guests can provide their own OS kernel
  - Guests can execute at various privilege levels

OS-Level Virtualization

- The OS kernel (or the whole OS) is virtualized
  - Guests can provide the user-space part of the OS (system libraries + binaries, boot scripts, ...) or just an application...
  - ...But continue to use the host OS kernel!
- One single OS kernel (the host kernel) in the system
  - The kernel virtualizes all (or part) of its services
- OS kernel virtualization: container-based virtualization
- Example of OS virtualization: wine

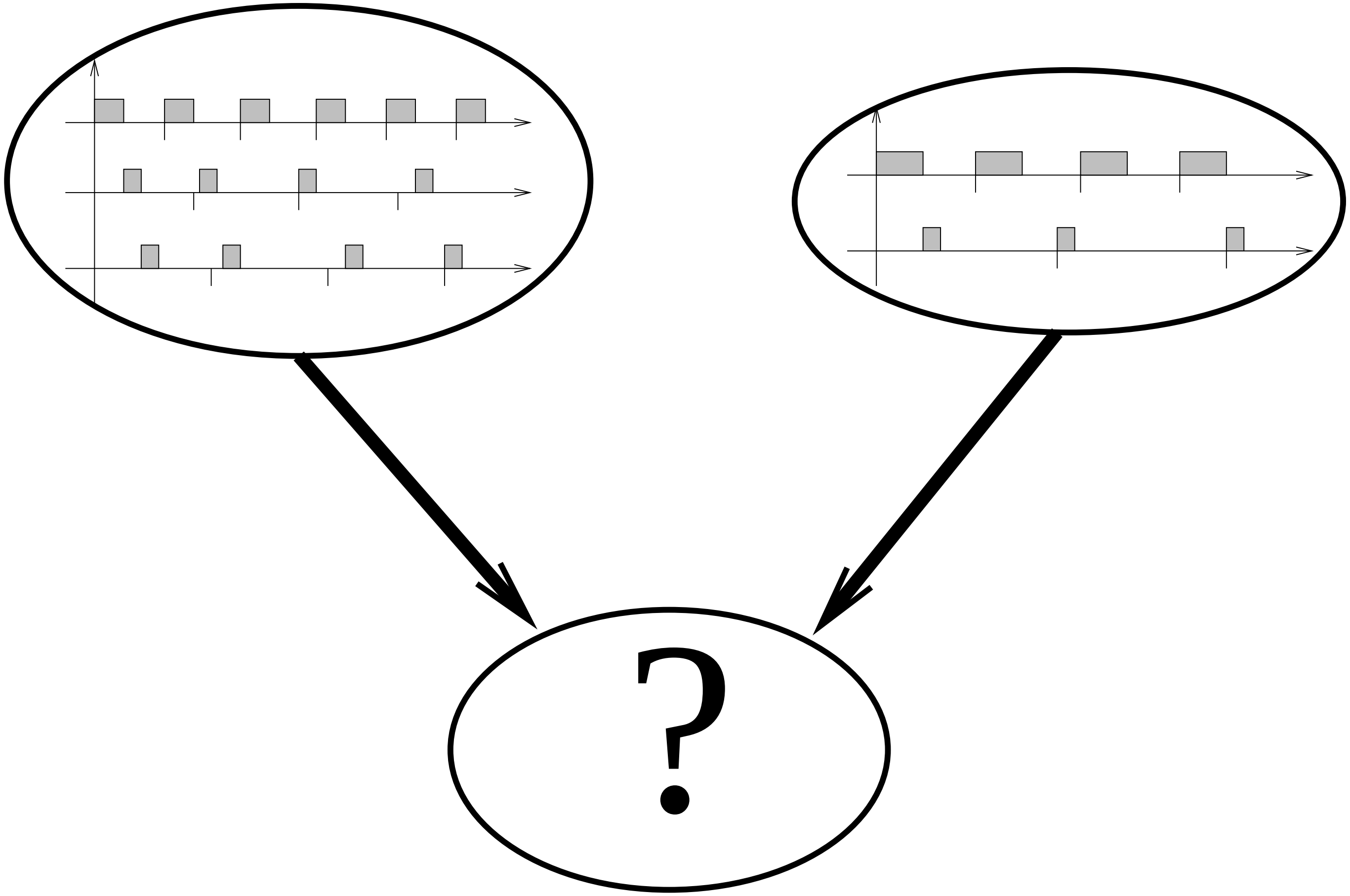
Virtualization Levels and Technologies

- Virtualization level: hw, kernel, OS, language, ...
  - Defines the abstractions provided through virtualization
- Virtualization technology: how is the VM implemented?
  - “Full VM” for hw virtualization
    - Hypervisor $\rightarrow$ software component used to virtualize the CPU
  - Kernel mechanisms for implementing application or OS containers (OS-level virtualization)
- But... Hypervisors can do OS-level virtualization too!

Real-Time and Virtualization???

- Serving real-time applications in (for example) Linux-based systems is not a problem today...
- ...Can real-time applications run in Virtual Machines?
  - Real-Time in Virtual Machines??? But... Why?
- Component-Based Development
- Security
  - Isolate real-time applications in a VM
- Easy deployment
- Real-Time Cloud Computing
- ...

Combining Real-Time VMs



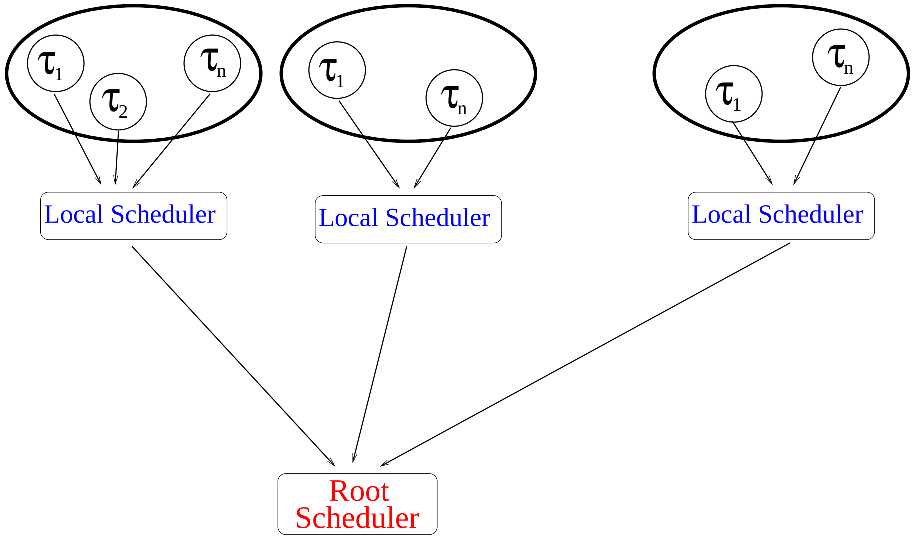
- Real-time analysis for each application / in each VM...
- What about the resulting system?

Modelling RT VMs

- Example: VM $\mathcal{C}^i$ containing n^i real-time tasks
- How to model/analyze it?
  - Real-time theory only shows how to schedule single tasks...
    - Here, we schedule VMs!
  - We need to somehow "summarise" the requirements of a VM containing multiple tasks!
- So, 2 main issues:
  1. Describe the temporal requirements of a VM in a simple way
  2. Schedule the VMs, and somehow "combine" their temporal guarantees

Schedulers Hierarchy

- Scheduling hierarchy
  - Root/Host scheduler (schedules VMss)
  - Local/Guest scheduler inside each VM
- Compositional Scheduling Framework (CSF): allows to compose real-time guarantees



Implementable through Virtual Machines!

- Hardware virtualization
- Containers

Root Scheduler Requirements

- First requirement: analyse the schedulability of a VM independently from other VMs
  - The root scheduler must provide some kind of temporal protection between VMs
- Various possibilities
  - Resource Reservations / server-based approach
  - Static time partitioning
  - ...

The root scheduler must guarantee that each VM receives a minimum amount of resources in a time interval

Schedulability Analysis

- (Over?)Simplifying things a little bit...
- ...Suppose to know the amount of time needed by a VM to respect its temporal constraints and the amount of time provided by the global scheduler
- A VM is "schedulable" if

$$\text{demanded time} \leq \text{supplied time}$$

  - "demanded time": amount of time (in a time interval) needed by a VM
  - "supplied time": amount of time (in a time interval) given by the global scheduler to a VM
- Of course the devil is in the details

Supplied Time

- Description of the root scheduler temporal behaviour
- More formally:
  - Depends on the time interval t we are considering
  - Depends on the root scheduler $\mathcal{S}$
- Minimum amount of time given by $\mathcal{S}$ to a VM in a time interval of size s
- Problem: fixed-priority scheduling does not guarantee a minimum time to the VM!!!
  - It can be shown that *reservation-based scheduling* of VMs can work
- Reserve a runtime Q every period P for a VM

Summing Up...

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

Summing Up...

- 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

Summing Up...

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

Summing Up...

- 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
  - Open-source hypervisors: KVM and Xen
  - Real-Time containers

Hardware Virtualization and Latencies

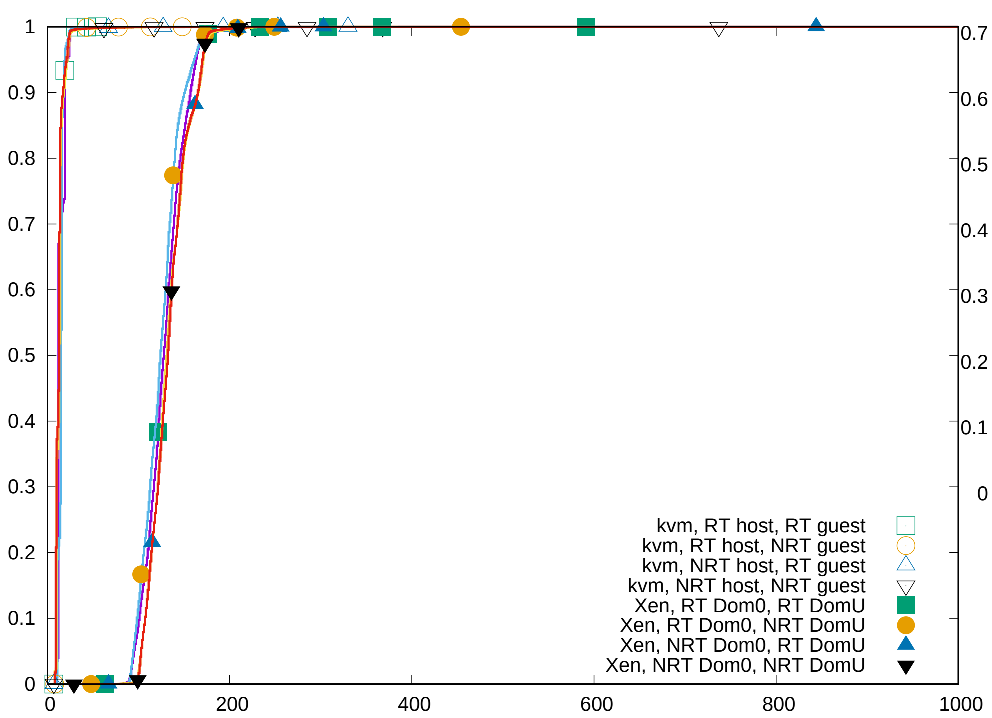
- Hypervisor: software component responsible for executing multiple OSs on the same physical node
  - Can introduce latencies too!
- Different kinds of hypervisors:
  - Xen: bare-metal hypervisor (*below* the Linux kernel)
    - Common idea: the hypervisor is small/simple, so it causes small latencies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

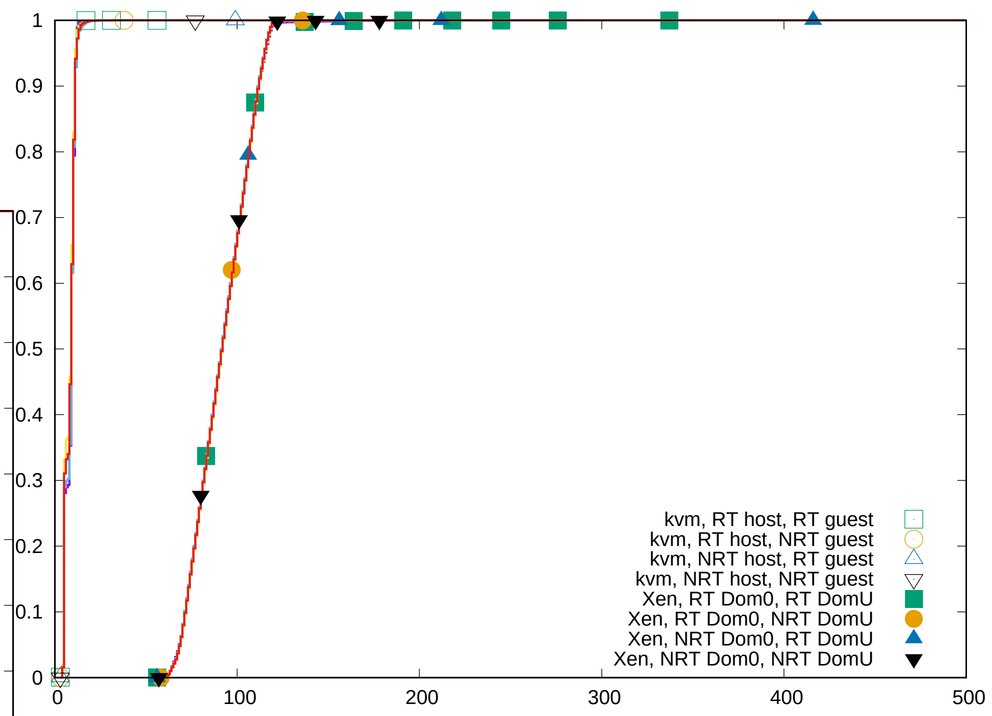
Hypervisor Latency

- 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





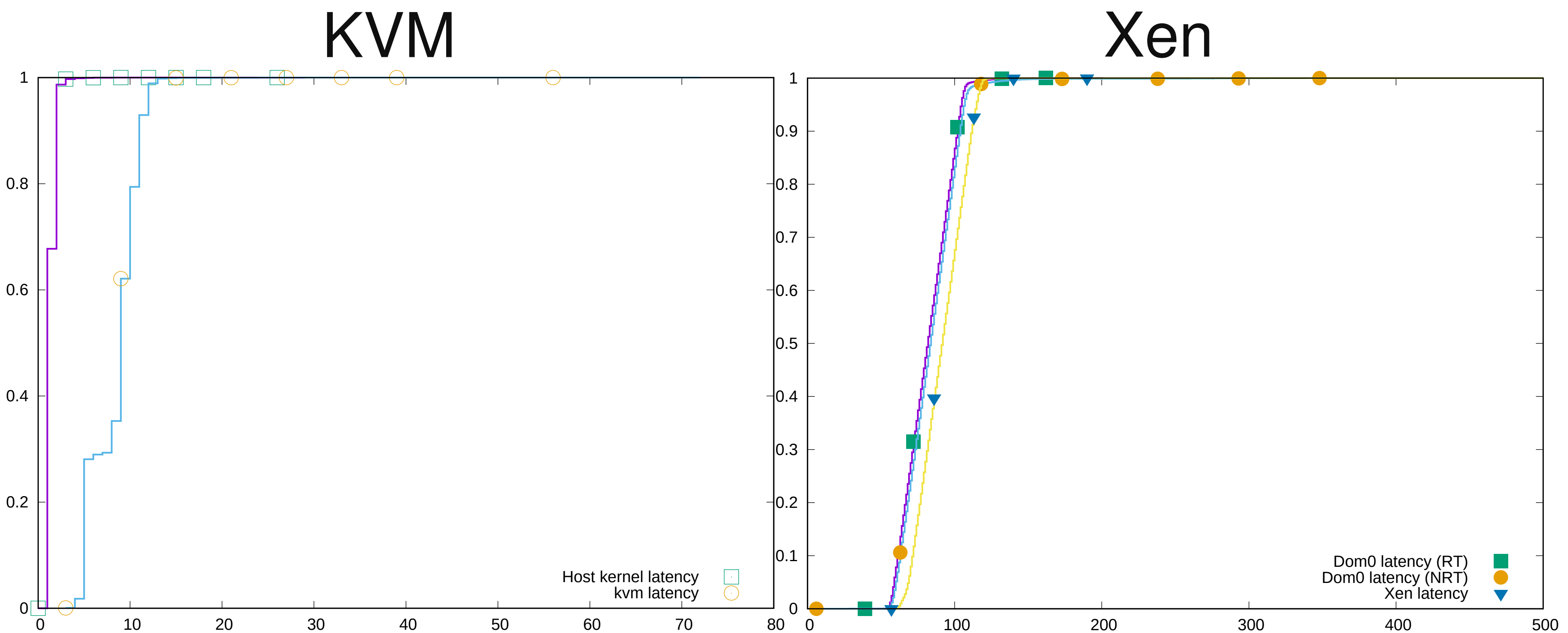
Intel Core i7

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

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)

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

Reducing the Xen Latencies

- 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 this "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?)

Scheduling the VMs

- Issue: how to guarantee a minimum amount of CPU time to each VM?
  - As said, `SCHED_FIFO` / `SCHED_RR` are not ok...
- `SCHED_DEADLINE` policy: schedule a thread / process for an amount of time Q every period P
  - Q: runtime
  - P: reservation period
- This is what we need!!! Allows to compute a supply function!
- Xen provides a similar scheduler (RTDS)

In Practice...

- Full hardware virtualization
  - KVM: use `SCHED_DEADLINE` to schedule the vCPU threads
  - Xen: use the RTDS scheduler
- OS-level virtualisation: containers (lxc, Docker, ...)
  - Extend `SCHED_DEADLINE` to schedule containers
    - More precisely, to schedule control groups
  - Implement the scheduling hierarchy in the kernel
    - Container-based real-time scheduling
- CPU allocation: interesting issues with multiple vCPUs

Example: KVM-Based VMs

- KVM: Linux driver for CPU virtualisation
  - User-space VMM (qemu, Firecracker, ...)
  - A thread per virtual CPU (vCPU thread)
- Use Preempt-RT for host and guest kernels
- Use `SCHED_DEADLINE` for the vCPU threads
  - CFS $\rightarrow$ (boring) algorithms to dimension Q and P
- Global guest scheduler $\leftarrow$ para-virtualised scheduling!!!
  - To always schedule on physical CPUs the highest priority guest tasks
- Use partitioned fixed priority scheduling in the guest

Example: Container-Based VMs

- Use Linux *control groups* and *namespaces*
  - One single scheduler for all the tasks
  - Can “see” the tasks inside containers
- Use Preempt-RT for the host kernel (there is no guest kernel)
- Patch `SCHED_DEADLINE` to schedule real-time control groups
  - Again, CSF allows to dimension Q and P
- No issues using a global guest scheduler!

Conclusions

- Deterministic scheduling of RT tasks in VMs is possible
  - But can be tricky; you need to know what to do
- RT support from Linux community $\rightarrow$ Preempt-RT
- Theory from RT research $\rightarrow$ `SCHED_DEADLINE`
- Lots of interesting problems, both theoretical and practical, being investigated at ReTiS Lab
  - Multi-core scheduling
  - Latencies in VMs
  - Scheduling VMs and scheduling *in* VMs
  - ...
- Results contributed to the open-source community

References from Literature

- Arvind Easwaran, Insik Shin, and Insup Lee. Optimal virtual cluster-based multiprocessor scheduling. *Real-Time Systems*, 43(1):25–59, September 2009. ISSN 1573-1383
- Linh T. X. Phan, Jaewoo Lee, Arvind Easwaran, Vinay Ramaswamy, Sanjian Chen, Insup Lee, and Oleg Sokolsky. CARTS: A tool for compositional analysis of real-time systems. *SIGBED Review*, 8(1):62–63, Mar 2011. ISSN 1551-3688
- Enrico Bini, Marko Bertogna, and Sanjoy Baruah. Virtual multiprocessor platforms: Specification and use. In *Proc. of 30th IEEE Real-Time Systems Symposium*, pages 437–446, 2009b
- Giuseppe Lipari and Enrico Bini. A framework for hierarchical scheduling on multiprocessors: from application requirements to run-time allocation. In *Proc. of 31st IEEE Real-Time Systems Symposium*, pages 249–258, December 2010
- E. Bini, G. Buttazzo, and M. Bertogna. The multi supply function abstraction for multiprocessors. In *2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 294–302, Aug 2009a. doi: 10.1109/RTCSA.2009.39
- I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embed. Comput. Syst.*, 7(3):30:1–30:39, May 2008. ISSN 1539-9087

References from SSSA

- Luca Abeni, Alessio Balsini, and Tommaso Cucinotta. Container-based real-time scheduling in the linux kernel. *SIGBED Rev.*, 16(3):33–38, November 2019a. doi: 10.1145/3373400.3373405. URL https://doi.org/10.1145/3373400.3373405
- Luca Abeni, Alessandro Biondi, and Enrico Bini. Hierarchical scheduling of real-time tasks over linux-based virtual machines. *Journal of Systems and Software*, 149:234 – 249, 2019b. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2018.12.008
- L. Abeni and D. Faggioli. An experimental analysis of the xen and kvm latencies. In *2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC)*, pages 18–26, 2019. doi: 10.1109/ISORC.2019.00014
- Luca Abeni and Tommaso Cucinotta. Adaptive partitioning of real-time tasks on multiple processors. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, SAC '20, page 572–579, New York, NY, USA, 2020a. Association for Computing Machinery. ISBN 9781450368667. doi: 10.1145/3341105.3373937. URL https://doi.org/10.1145/3341105.3373937
- Luca Abeni and Tommaso Cucinotta. EDF scheduling of real-time tasks on multiple cores: Adaptive partitioning vs. global scheduling. *SIGAPP Appl. Comput. Rev.*, 20(2):5–18, July 2020b. ISSN 1559-6915. doi: 10.1145/3412816.3412817. URL https://doi.org/10.1145/3412816.3412817
- Luca Abeni and Dario Faggioli. Using xen and kvm as real-time hypervisors. *Journal of Systems Architecture*, 106:101709, 2020. ISSN 1383-7621. doi: https://doi.org/10.1016/j.sysarc.2020.101709. URL https://www.sciencedirect.com/science/article/pii/S1383762120300035