Multi-Processor Real-Time Virtual Machines

Luca Abeni

luca.abeni@santannapisa.it

January 20, 2022

Multi-CPU VMs

- What about multiple CPUs?
  - Much more complex problem...
  - How to schedule the VMs on multiple CPUs?
  - Which local scheduler for multi-CPU VMs?
- How to model multi-CPU VMs?
  - Simplest (but pessimistic) solution: a supply function per CPU
- How to perform the schedulability analysis?
  - Depends on the (local and/or root) scheduler
- Multi-processor scheduling strategies: global vs partitioned

Multi-CPU Schedulers

- Root scheduler model:
  - Multi Supply Function, Multi-Processor Resource model (MPR), Parallel Supply Function (PSF), ...
  - MSF: Pessimistic, because the worst cases often cannot happen simultaneously
  - MPR: Again, pessimistic (does not specify how the runtime is distributed between cores)
  - PSF: much less pessimistic, but difficult to use
- MPR is strictly tied to global scheduling in the guest
- PSF is much more generic...
- What about MSF? Depends on the local scheduler

MSF And the Guest Scheduler

- MSF supports both global scheduling and partitioned scheduling in the guest
  - Global EDF (or Global FP) analysis...
  - Compute a (pessimistic) workload and compare it with the multi supply function
- Partitioned scheduling in the guest is also possible
  - Consider the tasks assinged to a virtual CPU...
  - ...Compute their dbf (or workload)...
  - ...And compare it with the sbf of the virtual CPU!!!
- This is all cool, but... What does “global scheduling” or “partitioned scheduling” mean?
- Let’s see... Multi-processor real-time scheduling in less than 10 slides!

Multiprocessor Scheduling

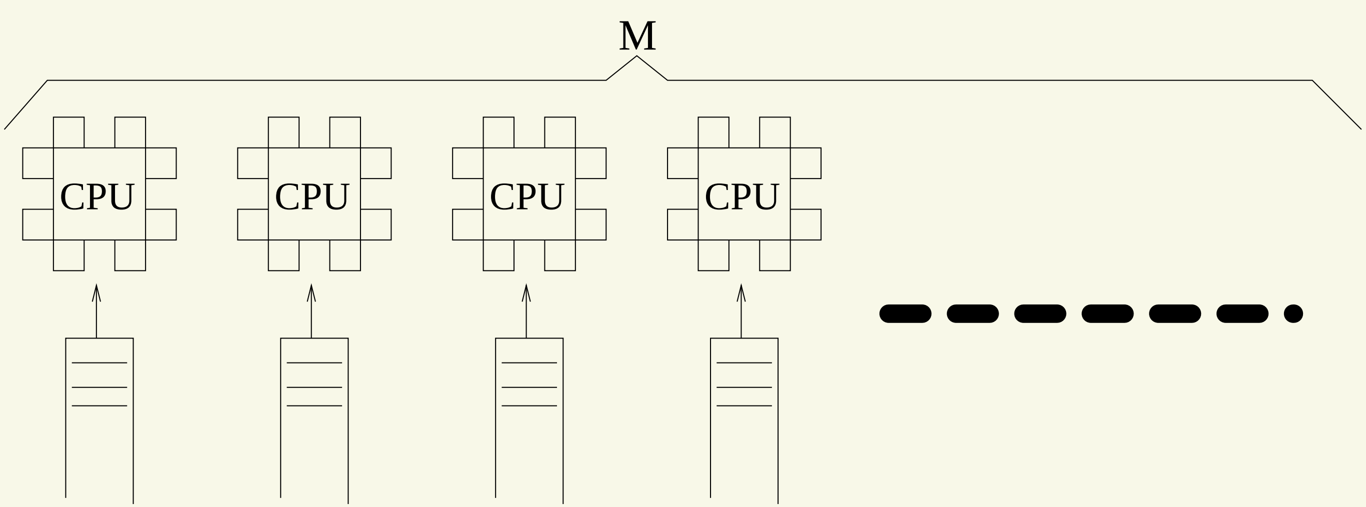
- UniProcessor Systems
  - A schedule $\sigma(t)$ is a function mapping time t into an executing task $\sigma : t \rightarrow \mathcal{T} \cup \{\tau_{idle}\}$ where $\mathcal{T}$ is the set of tasks running in the system
  - τ_{idle} is the *idle task*
- For a multiprocessor system with M CPUs, $\sigma(t)$ is extended to map t in vectors $\tau \in (\mathcal{T} \cup \{\tau_{idle}\})^M$
- Scheduling algorithms for $M > 1$ processors?
  - Partitioned scheduling
  - Global scheduling

The Quest for Optimality

- UP Scheduling:
  - N periodic tasks with $D_i = T_i$: (C_i, T_i, T_i)
  - Optimal scheduler: if $\sum \frac{C_i}{T_i} \leq 1$, then the task set is schedulable
  - EDF is optimal
- Multiprocessor scheduling:
  - Goal: schedule periodic task sets with $\sum \frac{C_i}{T_i} \leq M$
  - Is this possible?
  - Optimal algorithms

Partitioned Scheduling - 1

- Reduce $\sigma : t \rightarrow (\mathcal{T} \cup \{\tau_{idle}\})^M$ to M uniprocessor schedules $\sigma_p : t \rightarrow \mathcal{T} \cup \{\tau_{idle}\}$, $0 \leq p < M$
  - Statically assign tasks to CPUs
  - Reduce the problem of scheduling on M CPUs to M instances of uniprocessor scheduling
  - Problem: system underutilisation

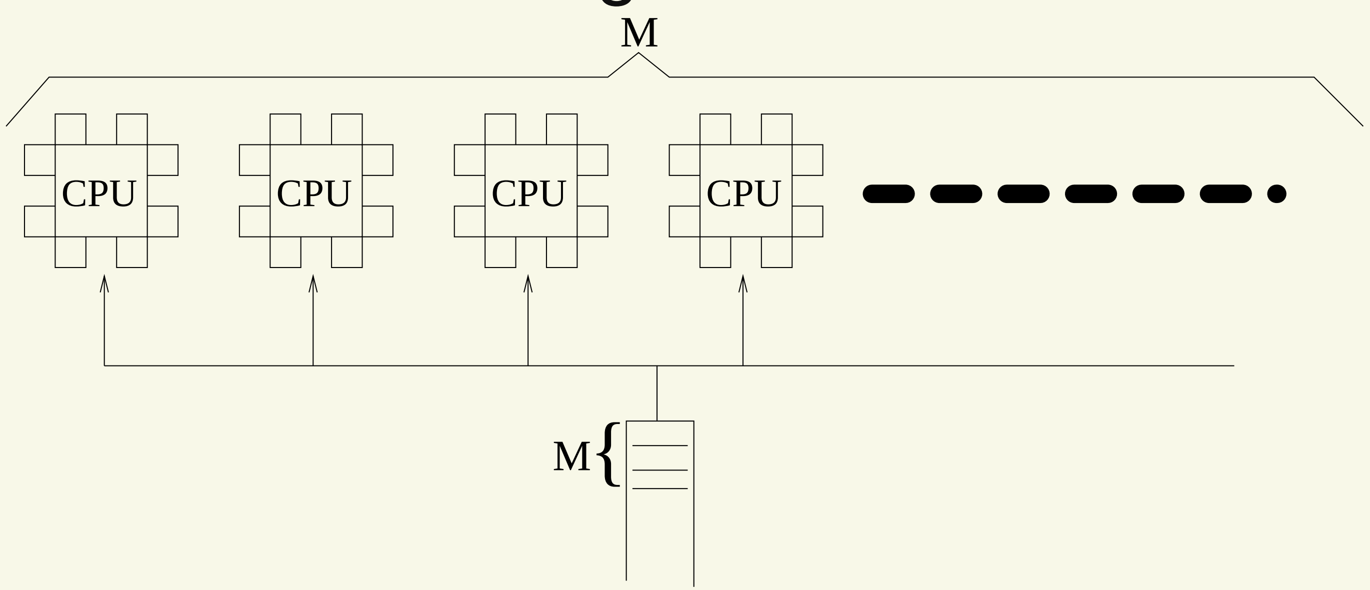


Partitioned Scheduling - 2

- Reduce an M CPUs scheduling problem to M single CPU scheduling problems and a bin-packing problem
- CPU schedulers: uni-processor, EDF can be used
- Bin-packing: assign tasks to CPUs so that every CPU has load ≤ 1
    - Is this possible?
- Think about 2 CPUs with $\{(6, 10, 10), (6, 10, 10), (6, 10, 10)\}$

Global Scheduling

- One single task queue, shared by M CPUs
  - The first M ready tasks are selected
  - What happens using fixed priorities (or EDF)?
  - Tasks are not bound to specific CPUs
  - Tasks can often migrate between different CPUs
- Problem: schedulers designed for UP...



Global Scheduling - Problems

- Dhall's effect: U^{lub} for global multiprocessor scheduling can be 1 (for RM or EDF)
  - Pathological case: M CPUs, $M+1$ tasks. M tasks $(\epsilon, T-1, T-1)$, a task (T, T, T).
  - $U = M\frac{\epsilon}{T-1} + 1$. $\epsilon \to 0 \Rightarrow U \to 1$
- Global scheduling can cause a lot of useless migrations
  - Migrations are overhead!
  - Decrease in the throughput
  - Migrations are not accounted for...

Global Scheduling for Soft Tasks

- Dhall's Effect $\rightarrow$ global EDF and global RM have $U^{lub} = 1$
  - With $U > 1$, deadlines can be missed
  - Global EDF / RM are not useful for hard tasks
- However, global EDF can be useful for scheduling soft tasks...
- When $U \leq M$, global EDF guarantees an upper bound for the *tardiness*!
  - Deadlines can be missed, but by a limited amount of time

Multi-Core Root and Local Schedulers

- Two different cases: multiple physical CPUs and multiple virtual CPUs
  - The host has multiple CPUs / cores: global or partitioned root scheduler
  - The VM is composed by multiple (virtual) CPUs / cores: global or partitioned local scheduler
- Root scheduler: using a global or partitioned approach only changes the admission test
  - Partitioned scheduler: M instances of uni-processor admission test
  - Global scheduler: more complex admission test (multi-CPU TDA)
- Local scheduler: things are more complex...

Multi-Core Scheduling in the Guest

- Guest scheduler (local scheduler): once a VM / component has been selected by the root scheduler, select a component's task
  - If the component runs on multiple (virtual) CPUs, can use a partitioned or global approach...
- Partitioned scheduling in the guest is easy
  - Every (virtual) CPU has its sbf; use it for schedulability analysis
- Global scheduling: on a physical machine, the M highest priority tasks are scheduled
  - VM: the m' highest priority tasks of the guest must be scheduled on physical CPUs
  - m': number of scheduled virtual CPUs

Global Scheduling in the Guest

- Assume a component is scheduled on 2 virtual CPUs...
- ...And has 3 fixed priority ready tasks
- The guest/local scheduler selects the 2 highest priority tasks and schedules them
  - Now, assume that the root scheduler schedules one of the 2 virtual CPUs and preempts the other one...
  - What happens if the guest schedules the highest priority task on the virtual CPU that is not scheduled???
- The guest/local scheduler must be aware of what the root scheduler is doing!!!
- If it is not, use partitioned scheduling in the guest!

Practical Example

- Example to understand the issue:
  - Real-time tasks $\tau_1 = (60, 100), \tau_2 = (10, 200)$
  - 2 vCPU threads, with (runtime,period) = $(90, 100)$ and $(40, 100)$
  - PSF says the tasks are schedulable
- Scheduling τ_1 on vCPU 1 and τ_2 on vCPU 2, all deadlines are respected...
- ...But if τ_1 is scheduled on vCPU 2, it misses deadlines!
  - The guest scheduler must know the runtime and period associated to each virtual CPU
  - Or, it must be informed when the runtime of a virtual CPU has been consumed!

Root Scheduler in Hypervisors

- Bare-metal hypervisor: implements a vCPU scheduler
  - Example: Xen provides an “RTDS” scheduler (deferrable server)
- Hosted hypervisor: use the host kernel scheduler
  - Example: using KVM, QEMU creates a thread for each virtual CPU (vCPU thread)
  - The `SCHED_DEADLINE` scheduling class can be used as a periodic server
- Not possible to use a global scheduler in the guest
- What to do if there is no hypervisor (OS-level virtualization, etc...)?

OS-Level Virtual Machines

- Do not virtualize the hardware, but the OS/kernel
  - Host kernel: virtualize its services to provide isolation among guests
- Based on *control groups* (`cgroups`) and *namespaces*
  - namespaces: isolate and virtualise system resources
  - cgroups: limit, control, or monitor resources used by groups of tasks
- No guest kernel $\Rightarrow$ no separate local scheduler
- No vCPU thread to be scheduled
  - What to schedule? Containers/groups of tasks $\rightarrow$ cgroups

Container-Based Virtual Machines

- Modify `SCHED_DEADLINE` to schedule groups of tasks (cgroups)
  - Tasks inside the group: fixed priority scheduling
  - Based on Linux containers $\leftarrow$ modifications of the control groups scheduler for real-time tasks
  - Can be used with `lxc`, Docker, etc...
- Reuse `SCHED_DEADLINE` code and control groups interface
  - Plug `SCHED_DEADLINE` (hard CBS algorithm) in real-time control groups

Container-Based Real-Time Scheduling — 1

- Result: scheduling hierarchy
  - `SCHED_DEADLINE` (CBS) as a root scheduler
  - Fixed priorities as a second level scheduler
- Associate runtime and period to each virtual CPU of the VM
  - No runtime migration
  - Can use *cpusets* to control the number of virtual CPUs
- Real-Time virtual machine implementation based on containers!
  - Supports both partitioned and global scheduling

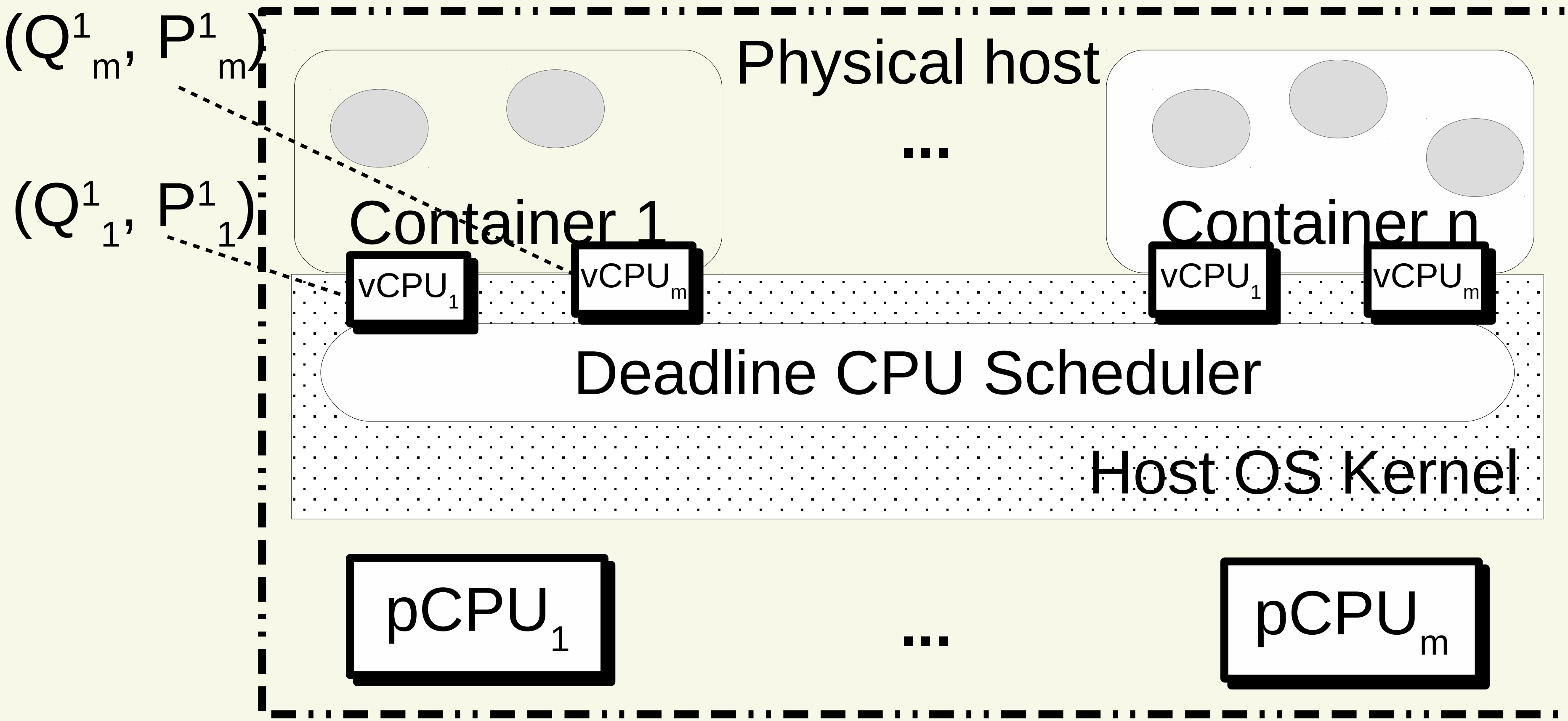
Container-Based Real-Time Scheduling — 2

- Supports multiple CPUs
  - For the moment, same runtime / period on all the CPUs
  - Can be easily fixed / improved
- One single (host) scheduler: can support global (fixed-priority) scheduling in the guest
  - No need to implement communication between two different schedulers
  - When the runtime of a vCPU is 0, the scheduler can migrate tasks (push)

Implementation

- Linux scheduling class: selects *entities*
  - Scheduling entities associated to tasks
  - `SCHED_DEADLINE` $\rightarrow$ dl entities
- Associate dl entities to queues of fixed priority tasks (rt runqueues)
  - A dl entity / rt runqueue per virtual CPU
  - When the dl entity is selected, get the highest priority task from the rt runqueue

Scheduling Hierarchy



Experimental Comparison

- VM with 4 virtual CPUs
  - Runtime $10ms$ and period $100ms$ on each virtual CPU
- CPU hungry real-time task in the guest
  - KVM-based VM: the task only executes for 10% of the CPU time on one core
  - lxc-based VM: the task only executes for 10% of the CPU time on all the 4 cores
- Because with KVM the guest has no way to know when the virtual CPU runtime is consumed (and to migrate the task)

Combining KVM and cgroup Scheduling

- How to use global guest schedulers in KVM-based VMs?
  - Scheduler para-virtualisation: not so easy in this case...
  - Avoid invasive changes and reduce the overhead
- Idea: try to combine KVM vCPU scheduling with real-time control groups
  - Schedule KVM vCPU threads in a control group
  - Hierarchical container-based scheduler for the control group
  - Need to pass information about thread priorities from guest to host