Again on Supplied Bound Functions

Luca Abeni

luca.abeni@santannapisa.it

October 14, 2019

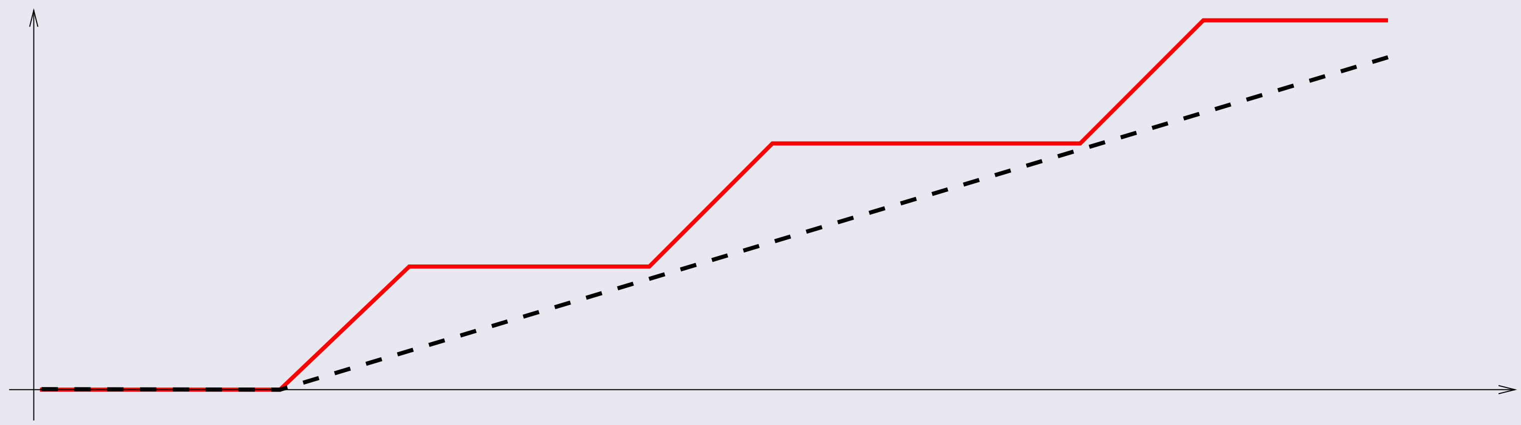
Understanding the Supplied Bound Function

- Supplied bound function $sbf(t)$: minimum amount of time that a VM is guaranteed to receive in a time interval of size t
  - Considers all the possible intervals of size t...
- Strange looking function!
  - Flat for large intervals of time...
  - $\frac{\delta sbf(t)}{\delta t} = 1$ in the other intervals
- Can we “summarise” it with something simpler?
- What about a line ($y = ax + b$)?
  - $sbf(t) < 0$ makes no sense...
  - So, better $sbf(t) = max\{0, at + b\}$

A Linear Approximation

- $sbf(t) = max\{0, at + b\}$... $at + b$ is below 0 for $t < -b/a$
  - Let's rewrite the equation... $at + b = a(t - \Delta)$ with $\Delta = -b/a$

$$sbf(t) = \begin{cases} 0 & \text{if } t < \Delta \\ a(t - \Delta) & \text{otherwise} \end{cases}$$



Interpreting the Linear Approximation

- $t < \Delta \Rightarrow sbf(t) = 0$: Δ is the *allocation delay* for the VM
  - Worst-case delay between the VM becoming active and the root scheduler scheduling it
  - How much time should I wait before the root scheduler starts giving the CPU to my VM?
- a (sometimes referred as α) is the *bandwidth* of the VM
  - Minimum fraction of CPU time reserved for the VM after the initial delay
- Of course, (a, Δ) should be so that $a(t - \Delta)$ is below the real $sbf()$

Periodic Servers Revisited

- How to compute (a, Δ) for a periodic server (Q^s, T^s)?
  - $a = \frac{Q^s}{T^s}$, $\Delta = 2(T^s - Q^s)$
- So, after the initial delay $2(T^s - Q^s)$ the VM is really receiving the expected fraction of CPU time (Q^s/T^s)
  - If we reduce T^s (keeping Q^s/T^s unchanged)...
  - ...$sbf(t)$ tends to the "fluid allocation"!
- Why not using very very small server periods?
  - Of course there is a reason...

The Design Problem

- Given a component (set of tasks and a local scheduler)...
  - Described by a time demand function (workload for fixed priorities)
- ...Find a root scheduler (and scheduling parameters) able to respect the components' temporal constraints
- Problem reduced to solving "$sbf(t) \geq dbf(t)$" for a set of points
  - Must be verified for all the points in case of EDF
  - Must be verified for at least one point in case of fixed priorities

Simplified Design

- $sbf(t) \geq dbf(t)$
- Using $sbf(t) = a(t - \Delta)$...

$$a(t - \Delta) \geq dbf(t) \Rightarrow \Delta \leq t - \frac{dbf(t)}{a}$$

- Solve this for every $(t, dbf(t))$, and plot the solution on a $a - \Delta$ plane...
- ...Then compute the intersection (for EDF) or union (for fixed priorities)

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 global) scheduler
- Multi-processor scheduling strategies: global vs partitioned

Multi-CPU Schedulers

- Global scheduler model:
  - Multi Supply Function
  - Pessimistic, because the worst cases often cannot happen simultaneously
- How to use MSF? Depends on the local scheduler
  - Global EDF (or Global FP) analysis...
  - Compute a (pessimistic) workload and compare it with the multi supply function
- What about a simpler solution? Let's try partitioned scheduling
  - But... What does "global" or "partitioned" mean?
  - Let's see... Multi-processor real-time scheduling in less than 10 slides!

Multiprocessor Scheduling

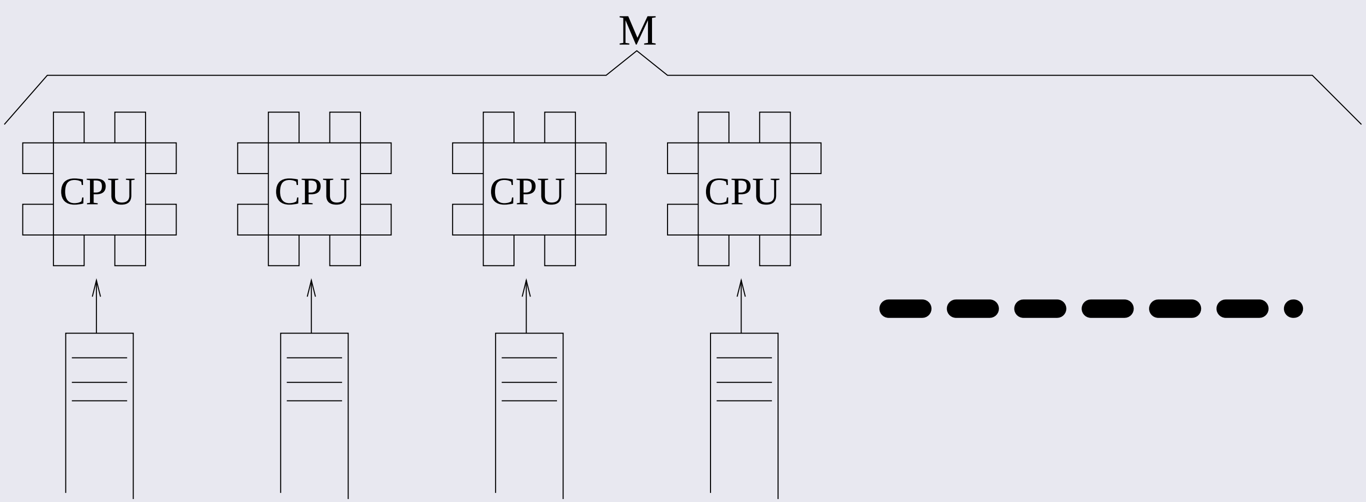
- UniProcessor Systems
  - A schedule $\sigma(t)$ is a function mapping time t into an executing task $\sigma : t \rightarrow \mathcal{T} \cup \{\tau_{idle}\}$ where $\mathcal{T}$ is the set of tasks running in the system
  - τ_{idle} is the *idle task*
- For a multiprocessor system with M CPUs, $\sigma(t)$ 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

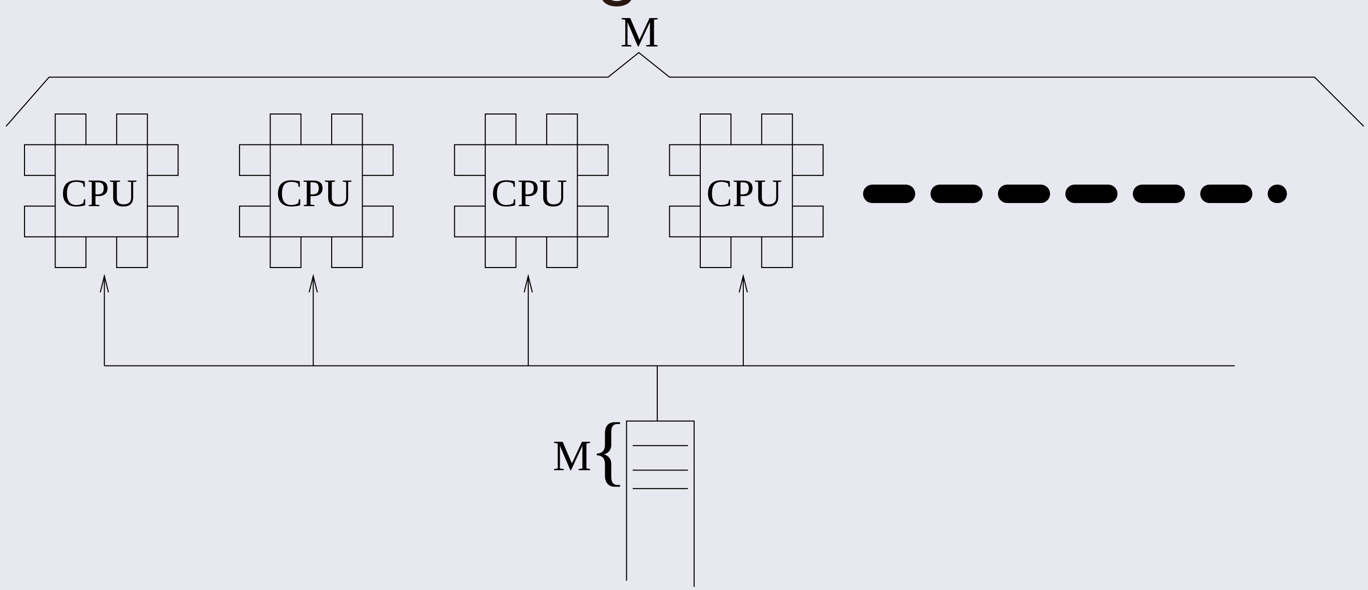


Partitioned Scheduling - 2

- 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 preempt 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!