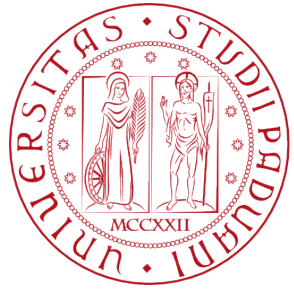
Experimental evaluation of optimal schedulers based on partitioned proportionate fairness

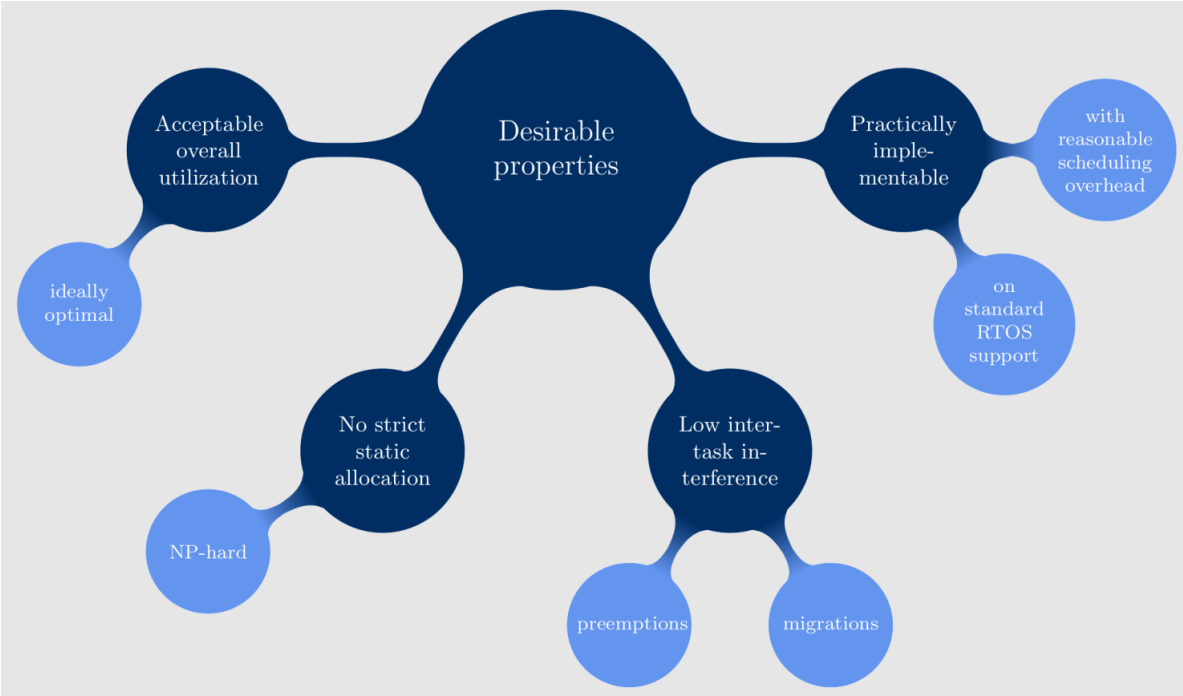
Davide Compagnin

TACLe Phd forum Y2, Novi Sad
May 4-5, 2015



Università
degli Studi
di Padova

Multiprocessor scheduling prerequisites



- Balancing good theoretical properties and practical requirements
  - **Low interference** and **high system utilization**
  - Standard **RTOS** support and reasonable **scheduling overheads**

Multiprocessor scheduling state-of-the-art

- **Partitioned approaches**
  - Reduce to single-core scheduling with well-known solutions
  - Bin-packing $\rightarrow$ NP-hard
  - In general cannot guarantee high utilization (50% bound)
- **Global approaches**
  - Work-conserving $\rightarrow$ Sustain relatively higher utilizations
  - Large shared scheduling structures $\rightarrow$ higher contention
  - Larger scheduling overheads (e.g., job migration)
- **Hybrid approaches**
  - Combinations that attenuates drawbacks of both
  - In general more difficult to implement
  - May require non-standard RTOS support

Optimality: dividing time into windows and executing tasks **proportionally** into each window $\rightarrow$ large number of preemptions and migrations

Multiprocessor scheduling state-of-the-art

- **Reduction to UNiprocessor** (RUN) - *RTSS'11* and **Quasi-Partitioned Scheduling** (QPS) - *ECRTS'14*
  - **Optimal** for implicit-deadline *periodic*/*sporadic* (QPS only) independent tasks
  - Relax the *proportionate fairness* $\rightarrow$ low interference with **few job migrations**
  - Have a softer approach to off-line bin-packing $\rightarrow$ reduce to P-EDF
- **Addressed issues**:
  - Are RUN and QPS viable in practice?
  - Which one performs better? in which cases?
  - What about their implementation?
  - How do they perform respect to P-EDF and G-EDF?

Recall of the RUN algorithm

- Reduction to UNiprocessor
  - Reduce the scheduling problem on a SMP to an equivalent problem on a (virtual) uniprocessor system
- Reduction principles
  - **Duality** (*idle-time scheduling*)

$$\tau_i(T_i, u_i) \overset{dual}{\Longleftrightarrow} \tau_i^*(T_i, 1 - u_i)$$

$$SCHED(\mathcal{T} \ni \{\tau_i\}, U) \equiv SCHED(\mathcal{T}^* \ni \{\tau_i^*\}, n - U)$$

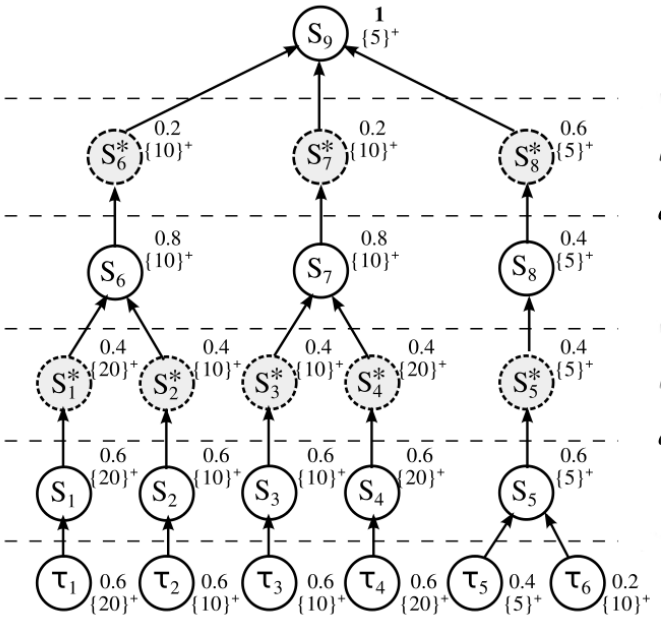
  - **Fixed-rate tasks** and **servers**

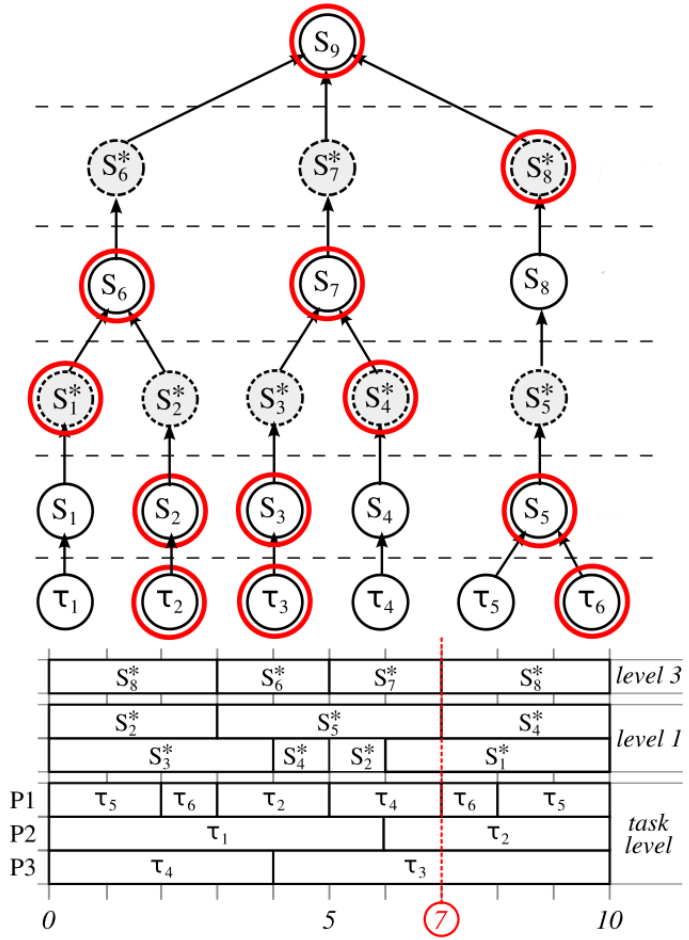
$$\tau_i \overset{\text{def}}{=\!=} (\mu_i, D_i) \Rightarrow S(\textstyle\sum_{\tau_i \in \mathcal{S}} \mu_i, \bigcup_{\tau_i \in \mathcal{S}} D_i)$$

  - Scheduling decision taken on **reduction tree**

Scheduling on RUN

- **Off-line**: reduction tree
  - Dual + Pack
- **On-line**: EDF rules
  - Virtual scheduling of servers





RUN implementation

- **Successfully implemented**
  - On top of *LITMUS^{RT}* Linux test-bed (MPI)
- Main implementation choices and challenges
  - Scheduling on the reduction tree
    - Data structure
    - Tree updates triggers
    - Relevance of packing policy
  - Mixing global and local scheduling
    - Global release event queue vs local level-0 ready queue
    - Handling simultaneous scheduling events
  - Meeting the full-utilization requirement
    - Variability of tasks' WCET and lower utilization

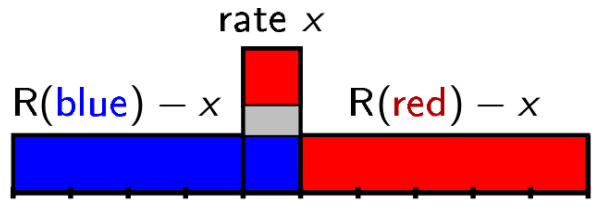
Recall of the QPS algorithm

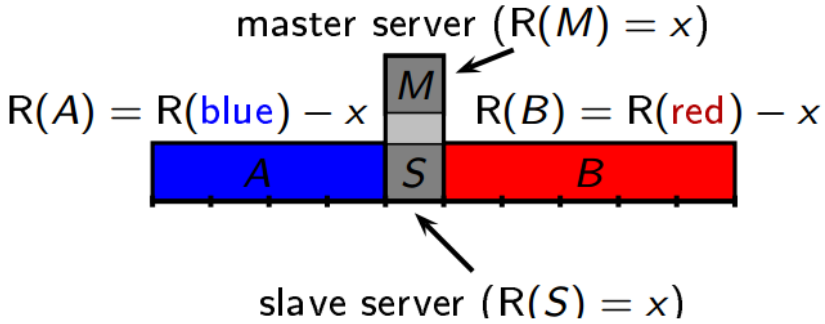
- **Quasi-Partitioning Scheduling**
  - Reduce the scheduling problem on a SMP to a number of equivalent uniprocessor problems

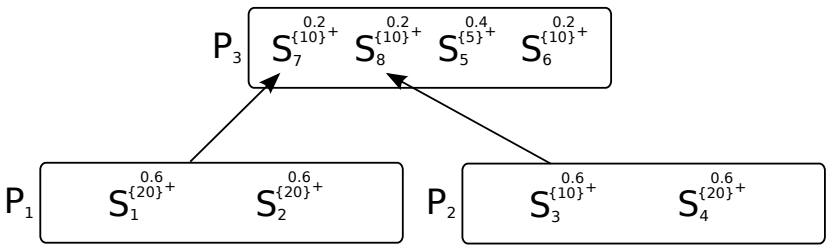
- Quasi-Partitioning
  - **Quasi-Partition** (*idle-time scheduling*)
    - Relaxed bin-packing in which processor capacity can be exceeded, when a proper partition is not found
  - **Fixed-rate tasks** and **servers**
    - An external server is assigned to schedule tasks exceeding the processor capacity on a remote processor
    - External servers induce a *processor hierarchy*
  - Scheduling decision taken on processor hierarchy

Scheduling on QPS

- **Off-line**:
  - Iterative allocation of tasks and servers via quasi-partitioning
- **On-line**: EDF rules
  - Virtual scheduling of servers







QPS implementation

- **Successfully implemented**
  - On top of *LITMUS^{RT}* Linux test-bed (MPI)
- Main implementation choices and challenges
  - EDF scheduling on each processor
    - Servers are scheduled according to their spare budget
    - IPIs are used to synchronized processor activities $\rightarrow$ increase the scheduling interference
    - Relevance of packing policy
  - Mixing global and local scheduling
    - Release events collapsed along the processor hierarchy
    - Job preemptions and completions are local
    - Handling simultaneous scheduling events
  - No full-utilization requirement

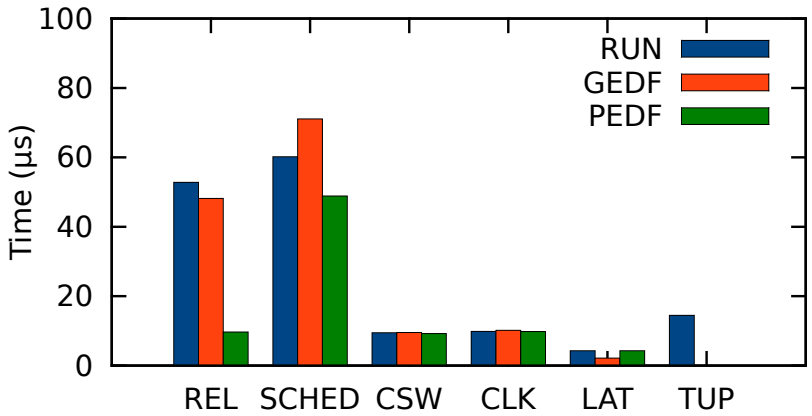
Resuming

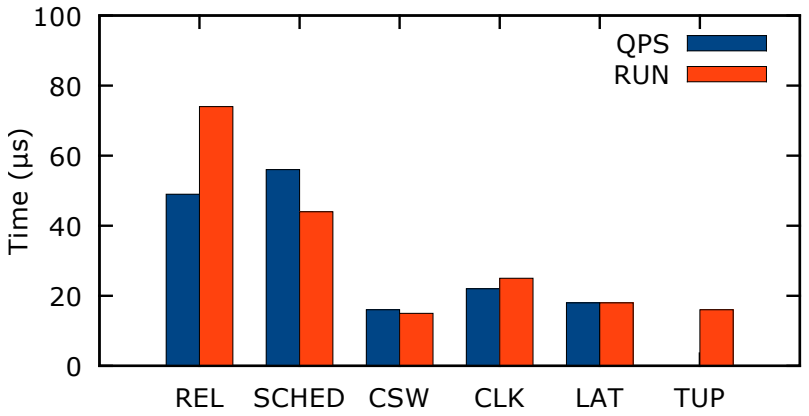
	RUN	**QPS**
Servers allocation	global	per-processor
Tasks allocation	per-server	per-server
Other global data structures	the reduction tree	no
Release primitive	global	per-cluster
Scheduling primitive	local to the server	local to the processor
Additional primitives	budget exhaustion	no

Empirical evaluation

- Compared RUN against **P-EDF**, **G-EDF**, **QPS**
- **Empirical evaluation** instead of simulation-based
- Focus on **scheduling interference**
  - Cost of scheduling primitives
  - Incurred preemptions and migrations
- Experimental setup
  - *LITMUS*^{RT} on 8 and 16 cores
  - Collected measurements for the three algorithms
    - Hundreds of automatically generated task sets
    - Harmonic and non-harmonic, with global utilization in 50%-100%
    - Representative of small up to large tasks
  - Two-step process
    - Preliminary empirical determination of overhead

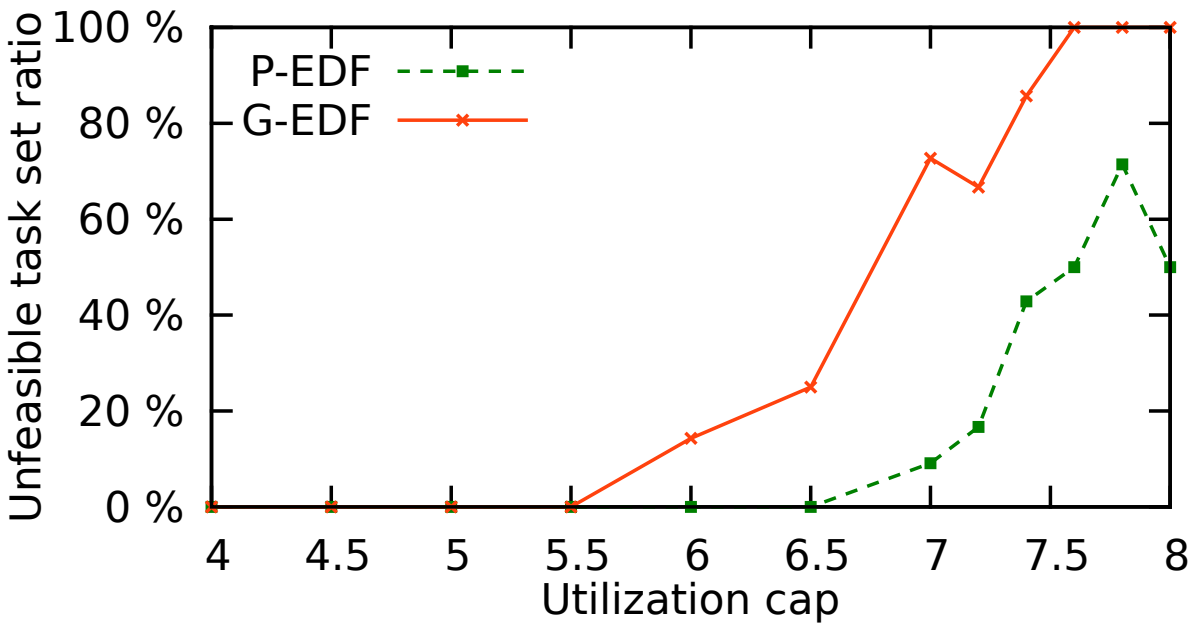
Primitive overheads and empirical bound





- Expectations confirmed
  - P-EDF exhibits less onerous primitives
- Tree update (TUP) triggered upon
  - Budget exhaustion event
  - Job release $\rightarrow$ REL includes TUP
- Empirical upper bound on RUN and QPS scheduling overhead
  - The scheduling events a job can suffer at most depends on the reduction levels/processor hierarchy depth

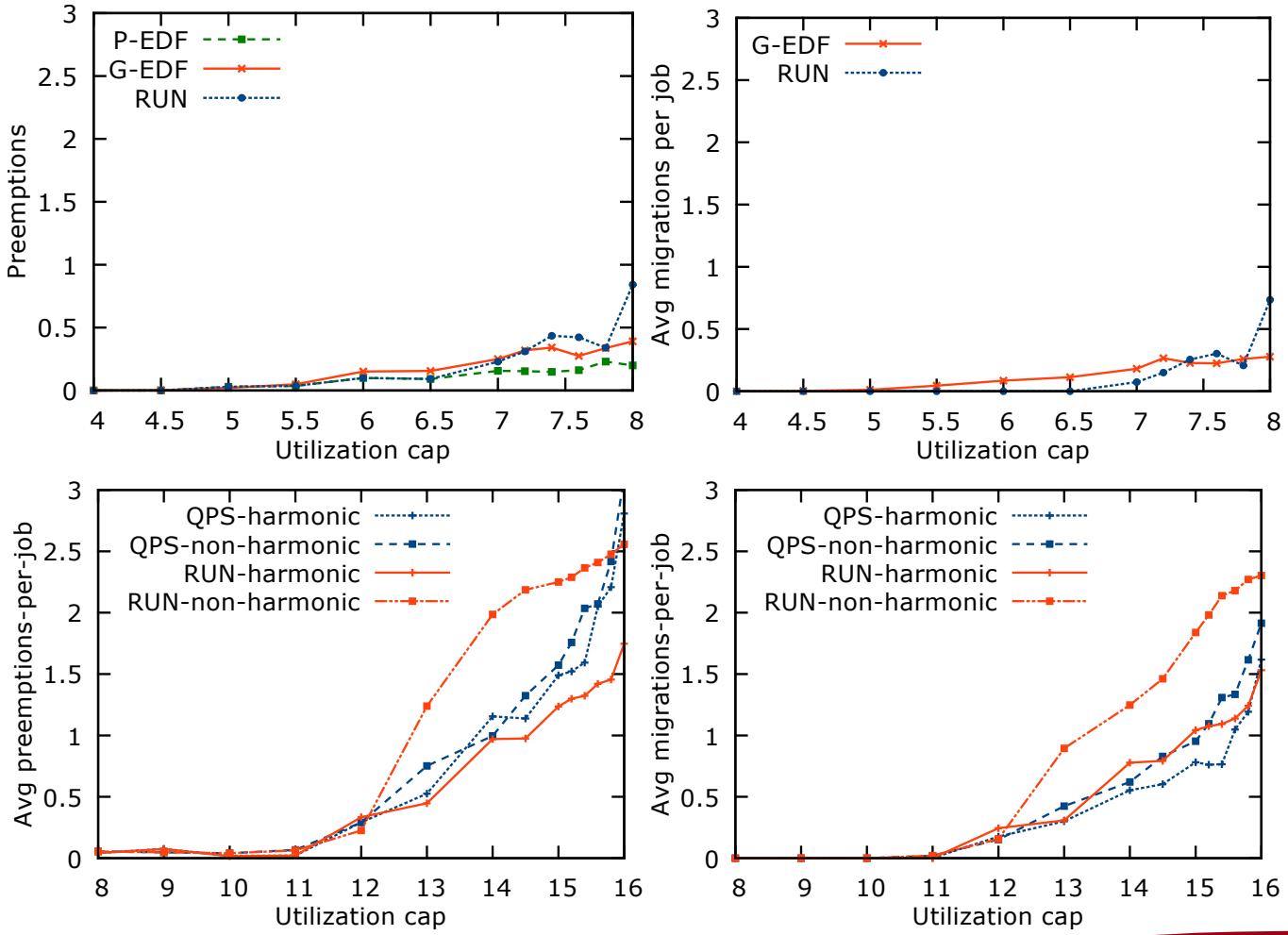
Empirical schedulability



- Task sets exhibiting at least one miss
- RUN suffered no misses
  - Optimality and tailored overhead

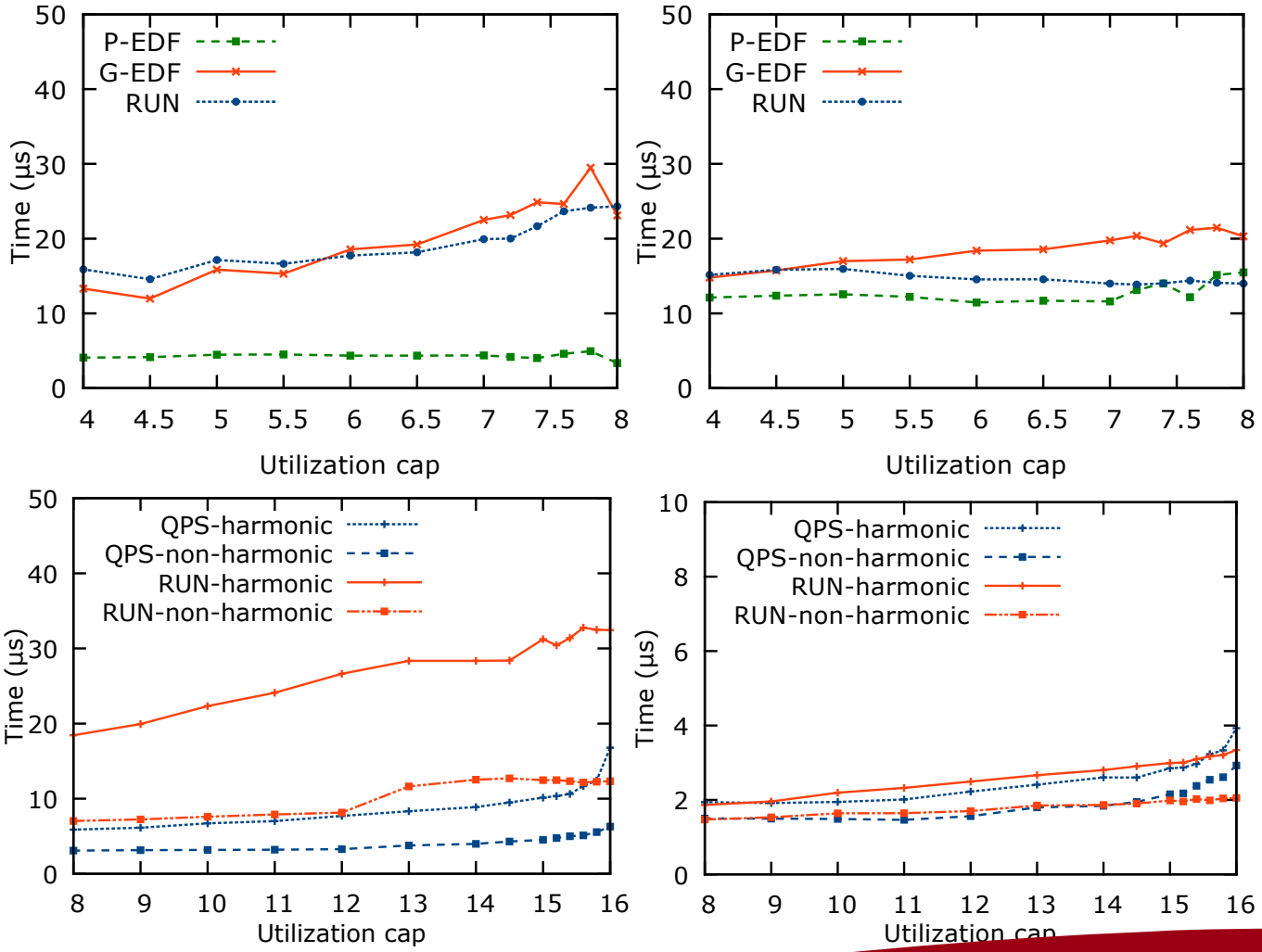
Kernel interference

- Observing average preemptions and migrations

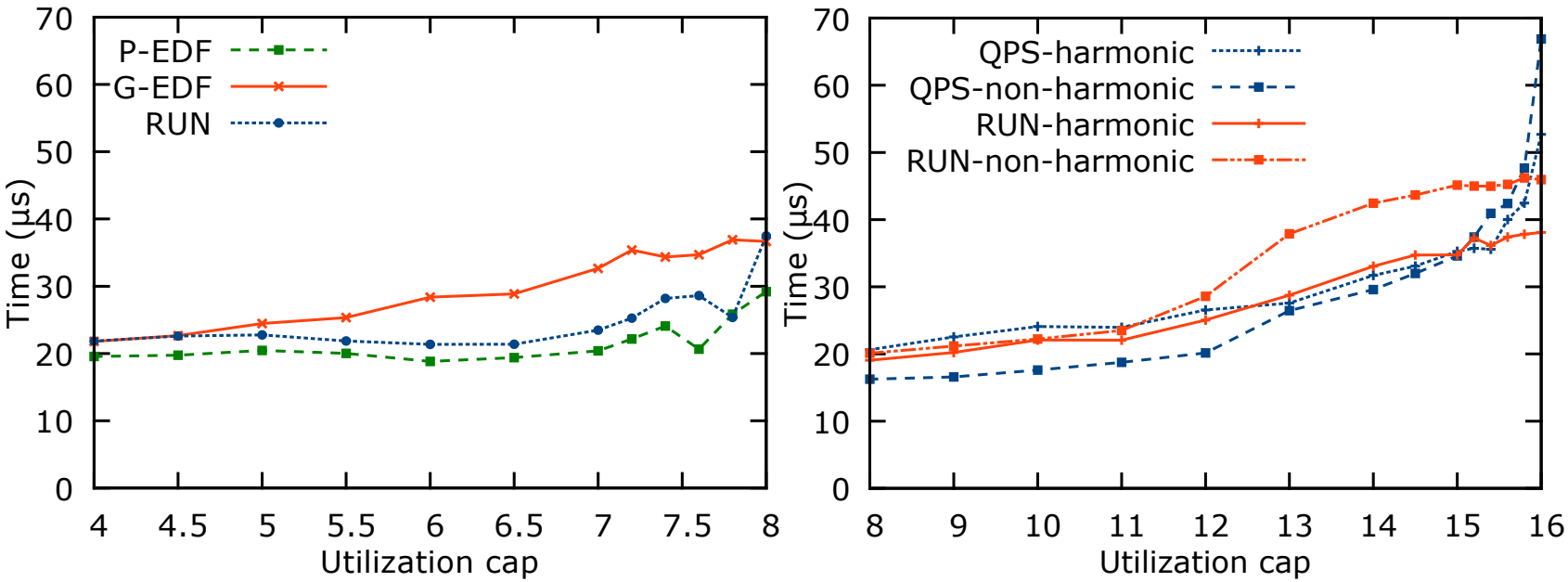


Scheduling cost

- Average cost of the scheduling primitives



Overall per-job scheduling overhead



Conclusions

- What can be drawn on RUN and QPS from our evaluation
  - Can be **practically** and **efficiently implemented**
  - May exhibit **very modest kernel overhead** similar to P-EDF
  - Allow to reduce **migrations**, thus inter-task interference
  - QPS incurs lower overhead than RUN except near full system utilization
    - Lightweight nature of the scheduling primitives
    - But high sensitive to packing