Experimental evaluation of optimal schedulers based on partitioned proportionate fairness

Davide Compagnin, Enrico Mezzetti and Tullio Vardanega
University of Padua - Italy

27th EUROMICRO Conference on Real-Time Systems (ECRTS)
Lund, July 9th, 2015

This project and the research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7 / 2007-2013] under grant agreement 611085

www.proxima-project.eu

Outline

- Motivation of our work
- Brief recall of Reduction to Uniprocessor and Quasi Partitioning Scheduling
- Implementation and evaluation
- Conclusions and future work

Introduction

RUN

Reduction to UNiprocessor
(RTSS-11)

→

QPS

Quasi-Partitioning Scheduling
(ECRTS-14)

optimal

relax the notion of proportionate-fairness

few preemptions and migrations

periodic tasks

sporadic tasks

The big question

RUN	QPS
implemented[1] on top of LITMUS^RT	?
modest run-time overhead comparable to that found in partitioned EDF	

[1] Compagnin, D.; Mezzetti, E.; Vardanega, T., "Putting RUN into Practice: Implementation and Evaluation," (ECRTS-14)

Recall of the algorithms

RUN QPS

off-line phase

multiprocessor scheduling problem → decomposition → uniprocessor scheduling problems

on-line phase

the schedule computed at the uniprocessor level is arranged to build a schedule for the original problem

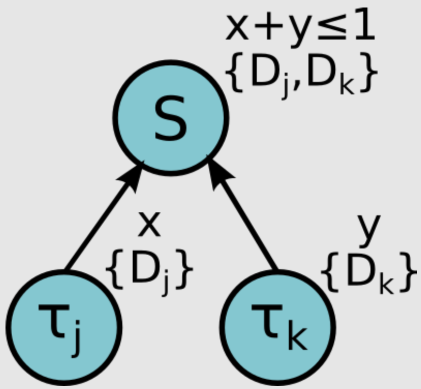
Recall of the algorithms

RUN

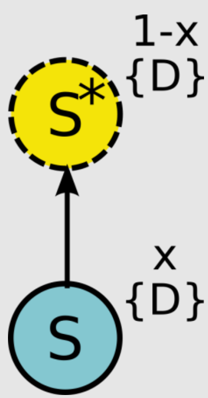
QPS

off-line phase

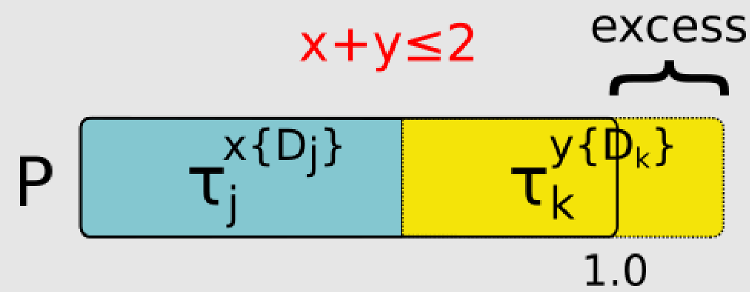
packing



dual



quasi-partition



the unitary processor capacity can be exceeded

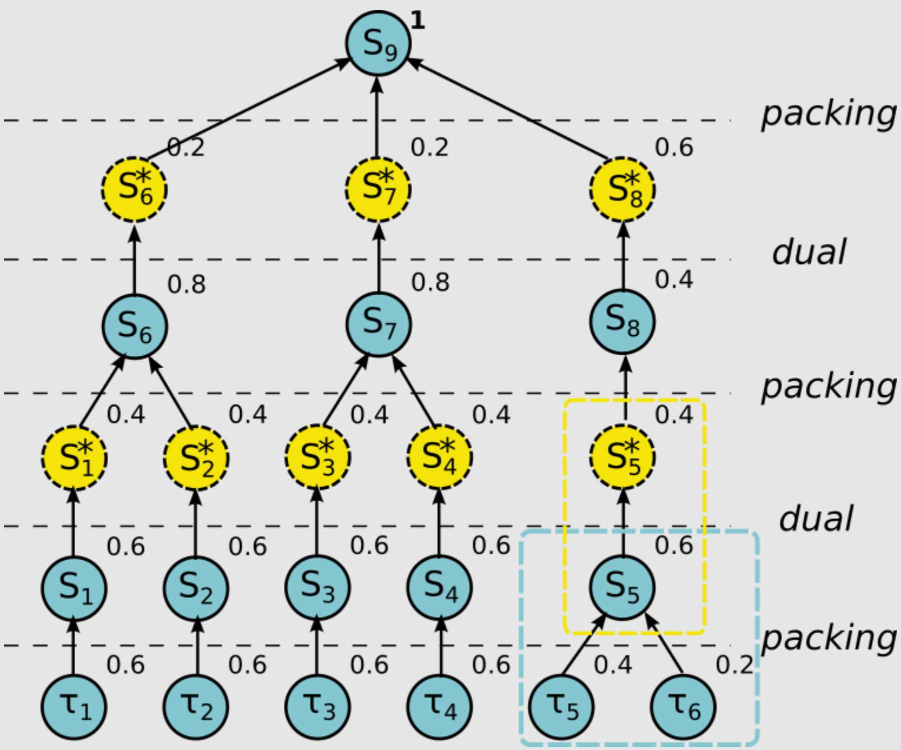
Recall of the algorithms

RUN

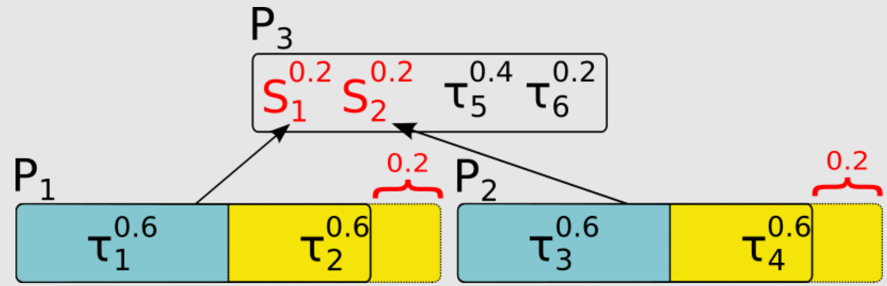
QPS

off-line phase

reduction tree



processor hierarchy



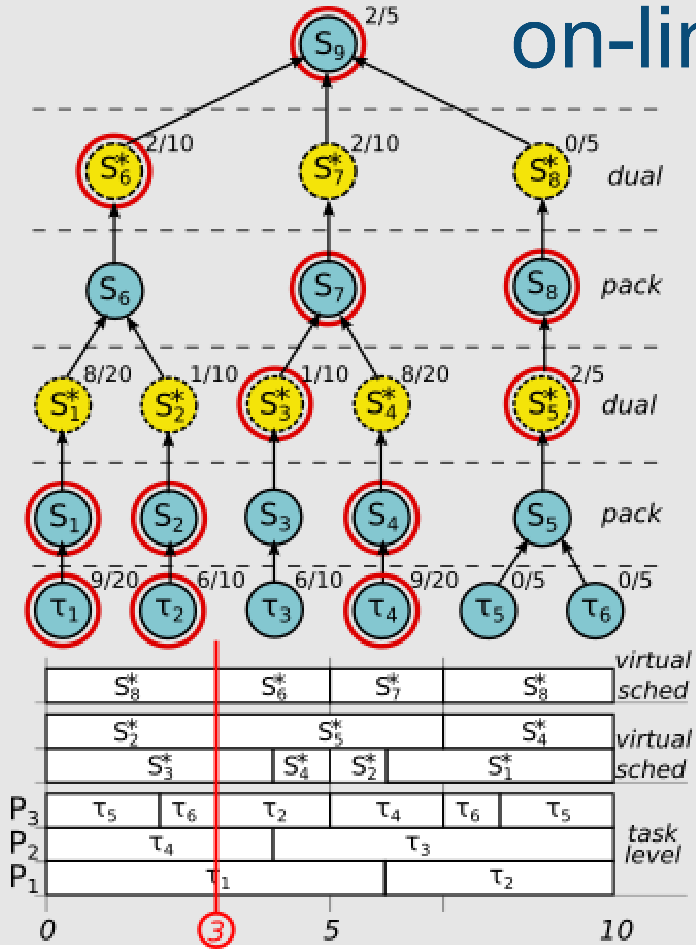
external servers reserve capacity for exceeding parts on a different processor

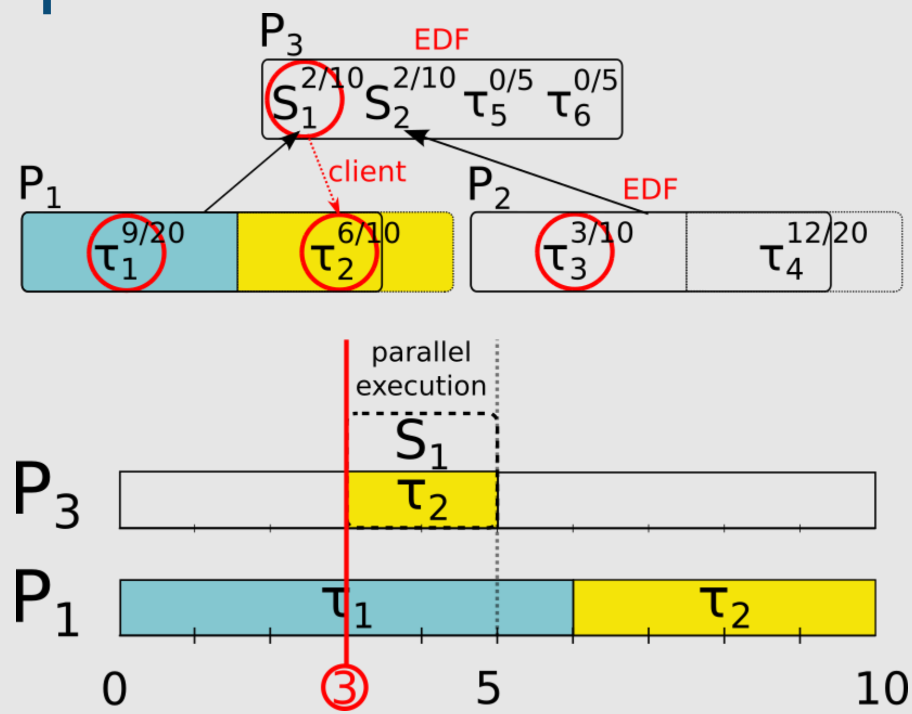
Recall of the algorithms

RUN

QPS

on-line phase





Implementation

RUN QPS

noteworthy differences

global scheduling

- virtual scheduling
- compact tree representation
- node selection is performed
- cpus are assigned to level-0 servers
- timers trigger budget consumption events
- release queue and lock

local scheduling

- tasks are selected by EDF

mostly local scheduling

- P-EDF + processor synchronization
- uniform task and server representation
- budgets consistently updated
- timer triggers budget consumption events
- per-hierarchy release queue and lock

Implementation

RUN QPS

noteworthy differences

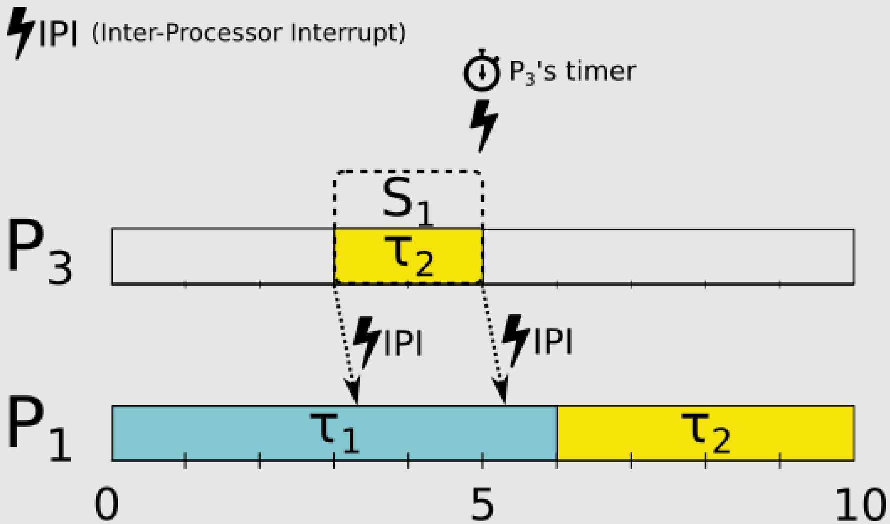
global scheduling

- virtual scheduling
- compact tree representation
- node selection is performed
- cpus are assigned to level-0 servers
- timers trigger budget consumption events
- release queue and lock

local scheduling

- tasks are selected by EDF

processors synchronization



P_3 notifies P_1 of the S_1's execution

Evaluation

- empirical evaluation instead of simulation-based

- focus on scheduling interference
  - cost of scheduling primitives
  - incurred preemptions and migrations

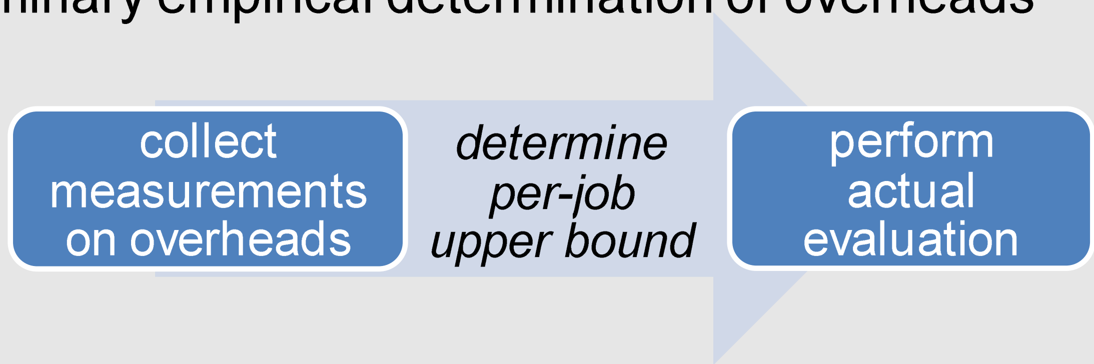
- sporadic tasks were left out

Experimental setup

- LITMUS^RT on a 16-cores AMD Opteron 6370P

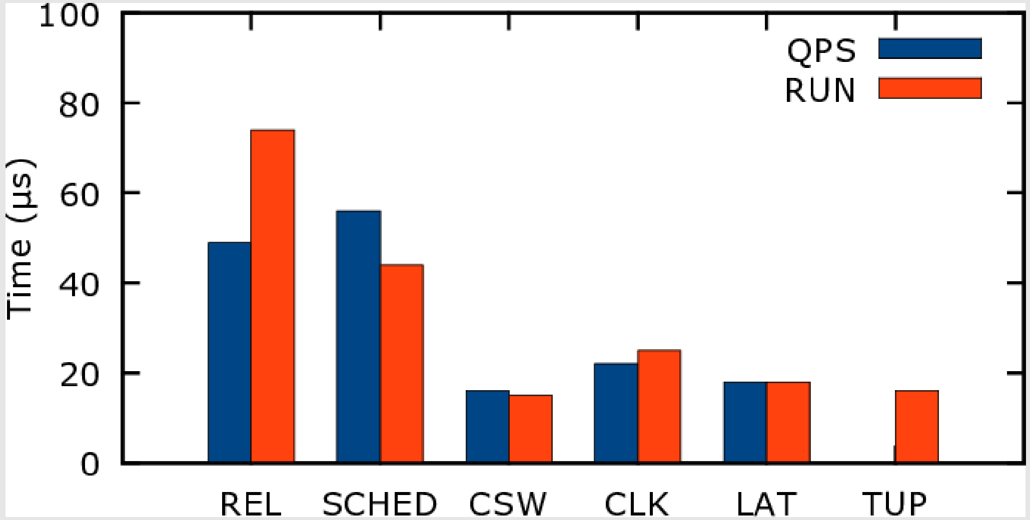
- collected measurements for the two algorithms
  - thousand of automatically generated task sets
  - harmonic and non-harmonic, with global utilization in 50%-100%
  - stressing the off-line and the on-line phases

- two-step process
  - preliminary empirical determination of overheads



Primitive overheads and empirical bound

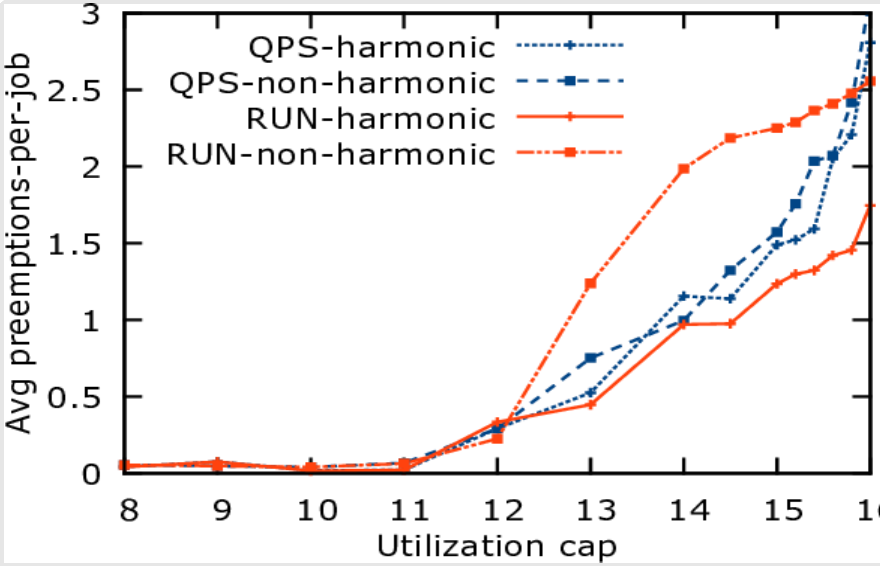
maximum observed overheads

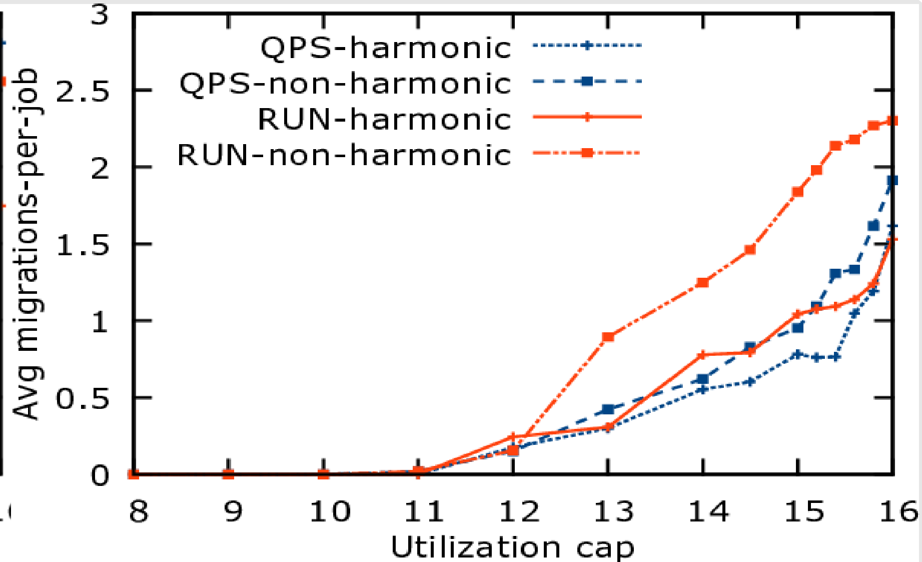


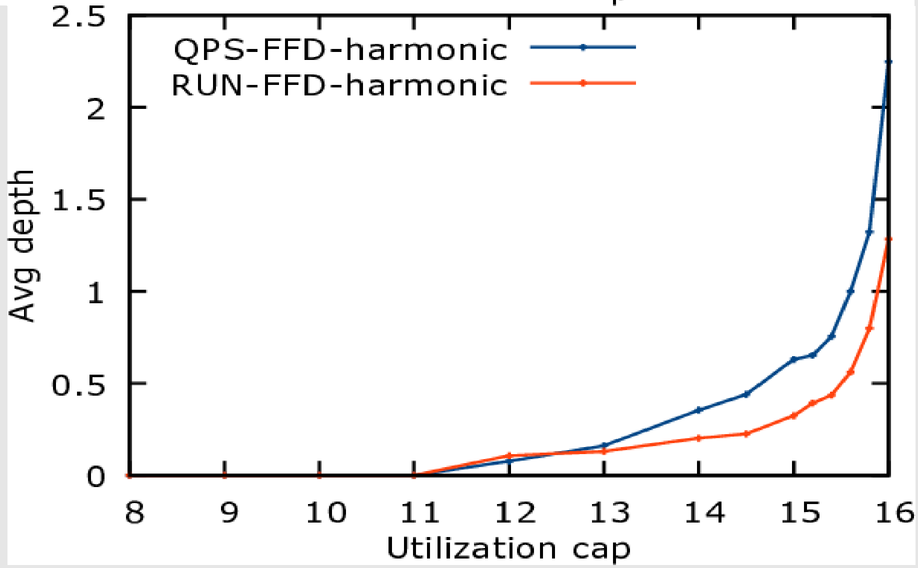
- expectation confirmed
  - QPS needs lighter-weight scheduling primitives
- QPS gets rid of Tree update operations (TUP)
- empirical upper bound on the scheduling overhead

$$max(OH^{Job}_{RUN}, OH^{Job}_{QPS})$$

Kernel Interference



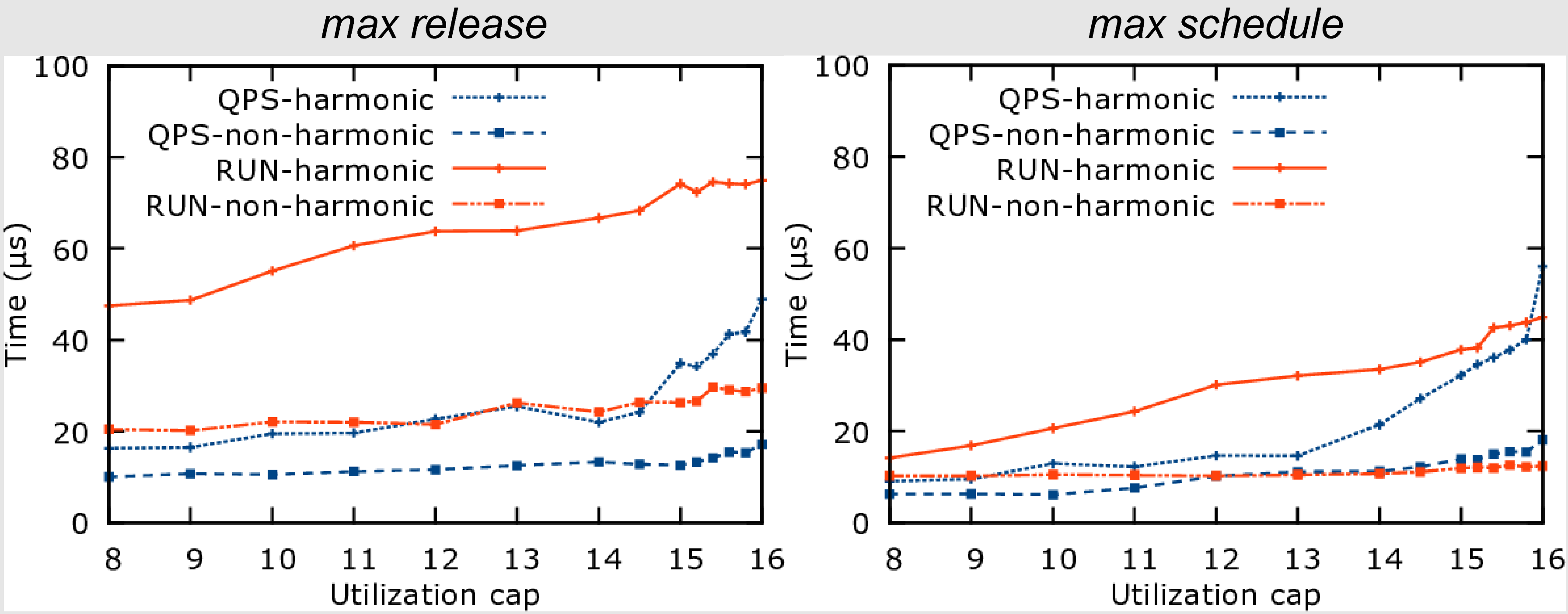




- observing preemptions and migrations at increasing the reduction-tree/processor hierarchy depth

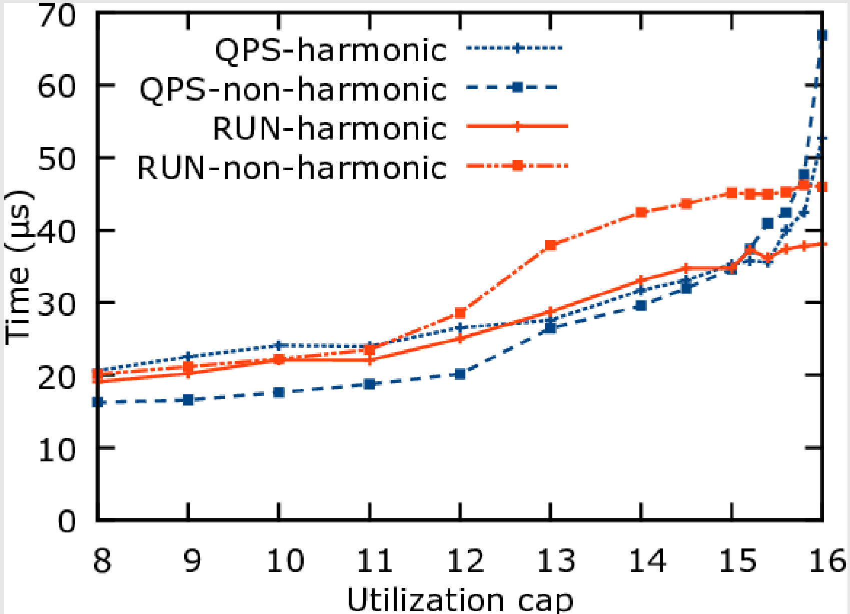
Scheduling cost

- maximum cost of core scheduling primitives

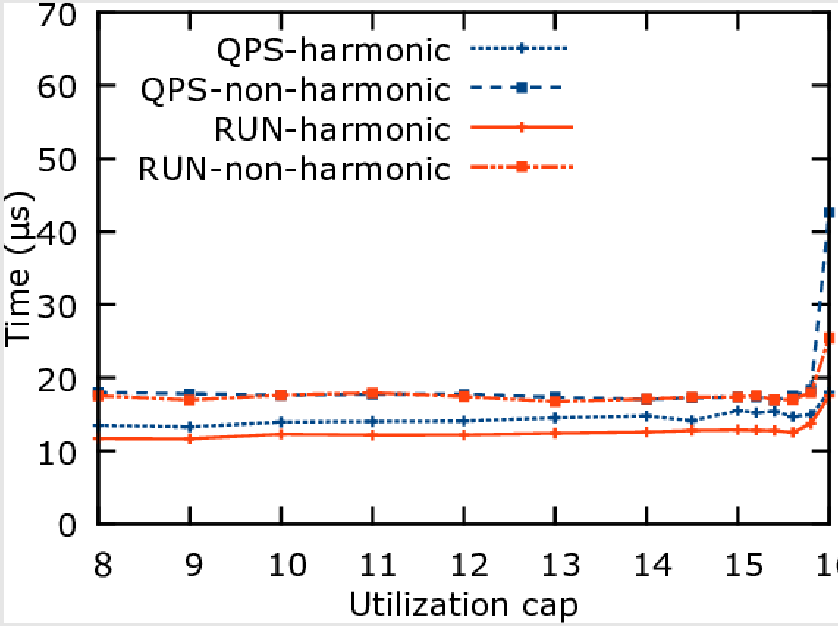


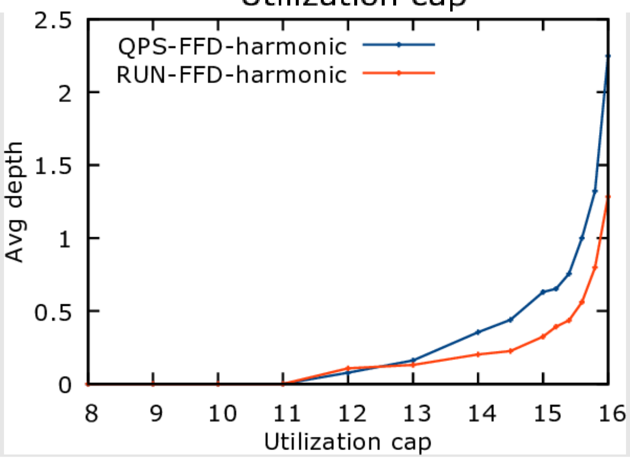
Per-job scheduling overhead

heavy tasks (utilization [0.5;0.9])



medium tasks (utilization [0.1;0.5])





- QPS is more susceptible to packing than RUN
- lightweight tasks favorite partitioning

Conclusions and future work

- QPS naturally embraces a partitioned design
  - overall improvement on the scheduling primitives
  - RUN needs a global scheduling coordination
- … but is more affected by the off-line phase
  - the processor hierarchy depth increases at full utilization
  - it incurs the additional overhead of processor synchronization
  - QPS works poorly at full-utilization
- global scheduling makes RUN less susceptible to the packing effect
  - updating the reduction tree is almost a constant time activity
- further work
  - toward many-cores: mixing RUN with message passing

Experimental evaluation of optimal schedulers based on partitioned proportionate fairness

Davide Compagnin, Enrico Mezzetti and Tullio Vardanega
University of Padua - Italy

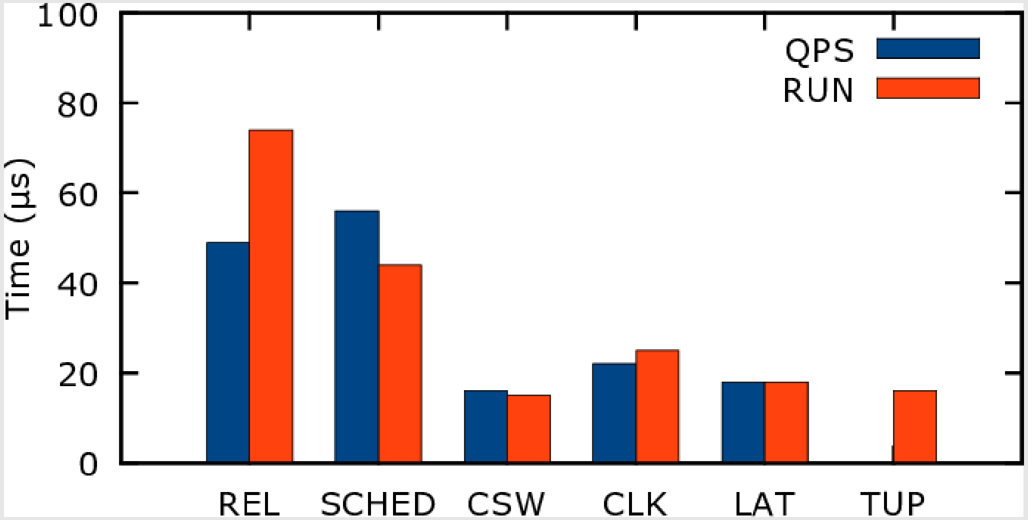
27th EUROMICRO Conference on Real-Time Systems (ECRTS)
Lund, July 9th, 2015

This project and the research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7 / 2007-2013] under grant agreement 611085

www.proxima-project.eu

Primitive overheads and empirical bound

maximum observed overheads



- empirical upper bound on the scheduling overhead
  - $OH^{Job}_{RUN} = REL + \widehat{SCHED} + CLK + k \times (UPT + \widehat{SCHED} + max(PRE, MIG))$
    where $k = \lceil (3p+1)/2 \rceil$
  - $OH^{Job}_{QPS} = REL + \widehat{SCHED} + CLK + k \times (\widehat{SCHED} + max(PRE, MIG))$
    where $k = \lceil m/2 \rceil$
  - $\widehat{SCHED} = SCHED + CSW + LAT$