PROXIMA

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 RUN and QPS algorithms
- Implementation and evaluation
- Conclusions and future work

Introduction

RUN	QPS
Reduction to UNiprocessor (RTSS-11)	Quasi-Partitioning Scheduling (ECRTS-14)

Optimal multiprocessor scheduling
Based on proportionate fairness
Designed to reduce # of preemptions and migrations

RUN	QPS
On periodic task-sets	Also on sporadic task-sets

Motivation

RUN

Implemented[1]
on top of LITMUS^RT

Confirming
moderate run-time overhead
in between that of P-EDF and G-EDF

QPS

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

Recall of the algorithms /1

RUN QPS

Off-line phase

Multiprocessor scheduling problem → decomposition → *Uniprocessor* scheduling problems

On-line phase

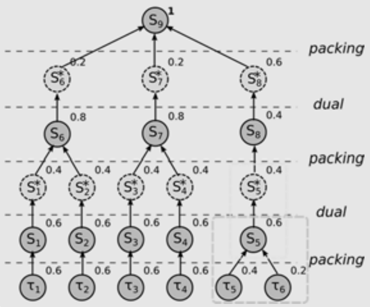
The multiprocessor schedule is "derived" from the corresponding uniprocessor schedule

Recall of the algorithms /2

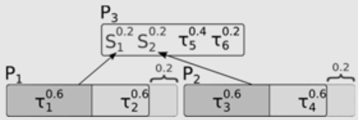
RUN QPS

Off-line phase

Reduction tree



Processor hierarchy



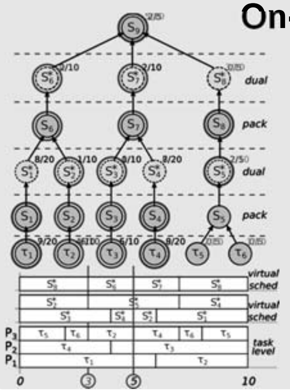
Unitary processor capacity can be exceeded

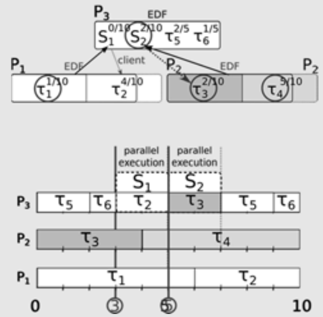
External servers reserve capacity for exceeding parts on a different processor

Recall of the algorithms /3

RUN QPS

On-line phase





Implementation /1

RUN QPS

Notable 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

- With EDF

Local scheduling + Processor synchronization

- uniform representation of tasks and servers
- budgets consistently updated
- timer triggers budget consumption events
- per-hierarchy release queue and lock

Implementation /2

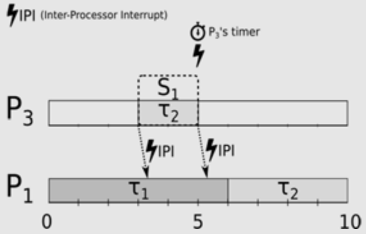
RUN | **QPS**

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
- with EDF

Local scheduling + Processor synchronization



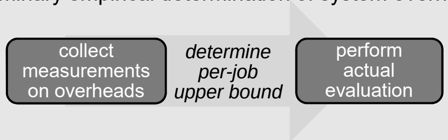
P_3 notifies P_1 of the S_1's execution

Evaluation

- Empirical evaluation instead of simulation
- Focus on scheduling interference
  - Cost of scheduling primitives
  - Incurred preemptions and migrations
- Evaluation limited to periodic task
  - *External servers* are always “active”
  - Sporadic activations would normally have lower utilization
    - Thus reducing the number of preemptions/migrations

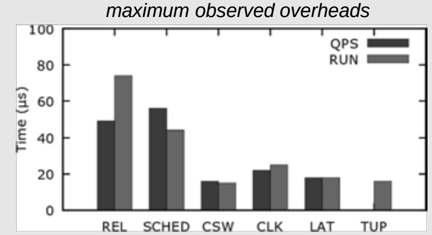
Experimental setup

- LITMUS^{RT} on a 16-cores AMD Opteron 6370P
- Exhaustive measurements over the two algorithms
  - Thousand of automatically generated task sets
  - Harmonic and non-harmonic, with global utilization in 50%-100%
  - Stressing both the off-line and the on-line phases
- Two-step experimental process
  - Preliminary empirical determination of system overheads



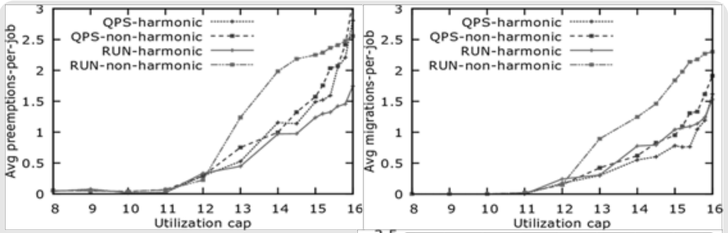
Primitive overheads and empirical bound

maximum observed overheads

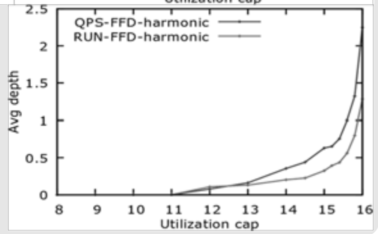


- Expectation was confirmed
  - QPS has lighter-weight scheduling primitives
  - And does not need Tree Update Operations (TUP)
- Empirical upper bound on the scheduling overhead
  - Based on theoretical bounds on the scheduling structures (RUN tree and QPS hierarchy)

Per-job scheduling interference

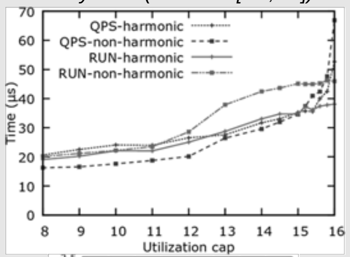


- Determined by preemptions and migrations
- In relation to reduction-tree and processor hierarchy depth

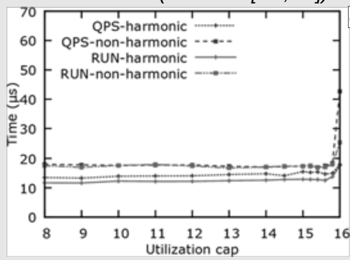


Overall per-job overhead

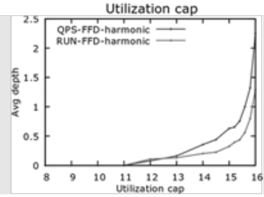
heavy tasks (utilization [0.5;0.9])



medium tasks (utilization [0.1;0.5])



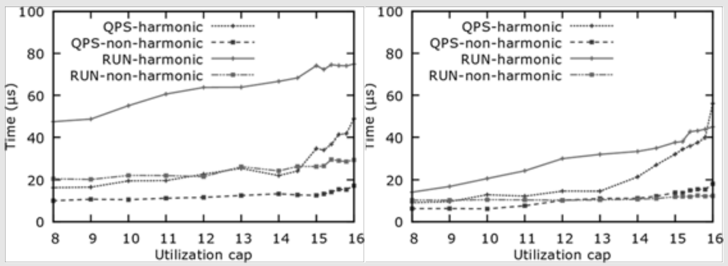
d23



- QPS is more susceptible to packing than RUN
- Lighter-weight tasks ease the partitioning problem
  - And lead to less complex scheduling structures

Scheduling primitives

max release *max schedule*



- Maximum observed cost of core scheduling primitives
  - *Release* and *Schedule*
  - Variation under increasing system utilization

Conclusions and future work

- QPS benefits from partitioned scheduling
  - Hence improves over RUN for cost of scheduling primitives
- … but is more susceptible to the off-line phase
  - QPS's need for processor synchronization hits performance badly with higher processor hierarchies
- RUN exhibits an almost constant overhead
  - Induced by its global scheduling nature
  - Which in turn may penalize it at lower system utilization
- Future work
  - Mainly interested in evaluating how this class of algorithms may behave when the number of processing units increases
    - Considering also how different implementation may affect the algorithm scalability

Slide 15

d23 remeber that we are talking about the avg cost for job here (so we expected to be constant on fully partitioned systems)

davide, 16/06/2015

PROXIMA

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