Putting RUN into practice

Implementation and evaluation

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

26th EUROMICRO Conference on Real-time Systems (ECRTS)
Madrid, 9 July 2014

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
Outline

- Motivation
- Brief recap of Reduction to UNiprocessor
- RUN implementation and evaluation
- Conclusions and future work
Multiprocessor scheduling requisites

- Seeking balance between theoretical properties and viability
  - Low runtime overhead 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
- Sustain relatively higher utilizations
- Large shared scheduling structures
- Larger scheduling overheads (e.g., job migration)

**Hybrid approaches**
- Flexibility to attenuate the drawbacks of P- and G- approaches
- More difficult to implement
- May require non-standard RTOS support

- **Reduction to UNiprocessor (RUN)**
  - **Optimal** for *implicit-deadline periodic independent* tasks
  - Low interference with **few job migrations**
  - Reduces to P-EDF when a perfect partitioning exists
Recap of the RUN algorithm

- **Reduction to UNiprocessor (RTSS’11)**
  - *Semi-partitioned* algorithm (for lack of better term)
  - *Optimal* without resorting to *proportionate fairness*

- **Reduction principles**
  - **Duality**
    \[
    \tau_i(T_i, u_i) \leftrightarrow \tau_i^*(T_i, 1 - u_i)
    \]
    \[
    SCHED(T_n, U, m) \equiv SCHED(T_n^*, n - U, n - m)
    \]
  - **Fixed-rate tasks** and **servers**
    \[
    \tau_i \overset{\text{def}}{=} (\mu_i, D_i) \Rightarrow S(\sum_{\tau_i \in S} \mu_i, \bigcup_{\tau_i \in S} D_i)
    \]

- **Scheduling decision taken on reduction tree**

- **Questions**
  - *Can it be implemented on standard RTOS support?*
  - *What is the cost of maintaining the reduction tree at run time?*
Scheduling on RUN

- **Off-line:** reduction tree
  - **Dual + Pack**

- **On-line:** **EDF rules**
  - Virtual scheduling of servers
    - Virtual jobs
    - Proportionate execution

\[\sigma \text{ (pack)}\]

\[\varphi \text{ (dual)}\]

\[\sigma \]

\[\varphi \]

\[\tau_1, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6\]

\[\begin{array}{cccc}
S_8^o & S_6^o & S_7^o & S_8^o \\
S_7 & S_5^t & S_3^t & S_4^t \\
P1 & \tau_5 & \tau_6 & \tau_2 & \tau_4 & \tau_6 & \tau_5 \\
P2 & \tau_1 & \tau_3 & \tau_2 & \tau_5 \\
P3 & \tau_4 & \tau_3 & \tau_2 & \tau_5 \\
\end{array}\]
RUN implementation

- For real
  - On top of LITMUS\textsuperscript{RT} Linux test-bed (UNC, now MP-SWI)
  - Thus relying on an abstraction of \textit{standard} RTOS support

- Main implementation choices and challenges
  - \textit{Scheduling on the reduction tree}
    - How to organize the data structure
    - How to perform virtual scheduling and trigger tree updates
    - Intrinsic influence of the packing policy
  - \textit{Mixing global and local scheduling}
    - Global release event queue vs. local \textit{level-0} ready queue
    - Handling simultaneous scheduling events
      - Job release, budget exhaustion (possibly from different sub-trees)
  - \textit{Meeting the full-utilization requirement}
    - Variability of tasks’ WCET and lower utilization
Empirical evaluation

- **Empirical evaluation** instead of simulation-based

- **Focus on** scheduling interference
  - Cost of scheduling primitives
  - Incurred preemptions and migrations

- **RUN** compared against **P-EDF** and **G-EDF**
  - RUN shares something in common with both
  - Way better than **Pfair** (S-PD² in LITMUS$^{RT}$)
    - RUN has superior performance for preemptions and migrations
Experimental setup

- **LITMUS**\textsuperscript{RT} on an 8-core AMD Opteron\textsuperscript{TM} 2356
- Collected measurements for RUN, P-EDF, G-EDF
  - Hundreds of automatically generated task sets
  - Harmonic and non-harmonic, with global utilization @ 50\%-100%
  - Representative of small up to large tasks
- **Two-step process**
  - Preliminary empirical determination of overheads
Primitive overheads and empirical bound

- Expectations confirmed
  - P-EDF needs lighter-weight scheduling primitives

- **Tree update** (TUP) triggered upon
  - *Budget exhaustion* event
  - Job release → REL includes TUP

- Empirical upper bound on RUN scheduling overhead
  - \[ OH_{RUN}^{Job} = REL + \widehat{SCHED} + CLK + k \times (TUP + \widehat{SCHED} + \max(PRE, MIG)) \]

  \[ k = \left\lceil \frac{3p + 1}{2} \right\rceil \quad \text{and} \quad \widehat{SCHED} = SCHED + CSW + LAT. \]
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 core scheduling primitives

![Graph showing average job release and schedule over utilization cap and utilization cap offset]

Average job release

Average schedule
Per-job scheduling overhead

Harmonic task set

Non-harmonic task set
Conclusions and future work

- Good news on RUN from this evaluation
  - It can be **practically** and **efficiently implemented**
  - It may exhibit **very modest kernel overhead**
    - Acceptable even on non-harmonic task sets
  - It causes a tiny amount of **migrations**
    - Hence low inter-task interference

- Essential improvements
  - Handle **sporadic task sets**
  - Allow sharing of **logical resources**

- Further work
  - Better understanding of the role of **packing policies**
    - Affecting the reduction tree, hence preemptions/migrations
  - Further **comparisons** against other optimal solutions
    - High interest in *Quasi-Partitioned Scheduling (QPS)*
Evaluation against S-PD²

Observed preemptions and migrations

Per-job kernel overhead
Reduction tree at run time

S6

S7

S8

S9

backward_ptr

first_child_ptr

sibling_ptr

D : earliest deadline
B : current Budget
circled : flag