Assunzioni RUN

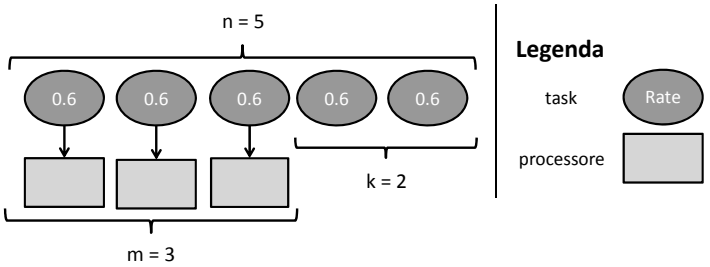
Parametri di modello

- m processori
- *Implicit-deadline task* $\tau_i, i \in \{1..n\}$ con $n = m + k \geq m$
- *Fully utilized system*: non ci devono essere tempi morti
- Ciascun task ha *fixed rate* $R_i = \frac{(d_i - r_i)}{C_i}$,
  - Che esprime la pendenza della fluid curve
- Task indipendenti
- Processori omogenei
- *Migration* e *preemption* sono assunti avere costo zero
- $\sum_{i=1}^{n}(R_i) \leq m$

  ⇒ SI PUO' FARE! Ma come? Quale Σ *scheduling* valido?

1

Esempio



- $\sum_{n=1}^{5}(R_n) = 3$ ⇒ almeno 3 processori
- Voglio trovare lo schedule Σ del **sistema S**

2

Sistema duale

L'algoritmo RUN crea l'**insieme duale** dei *task*

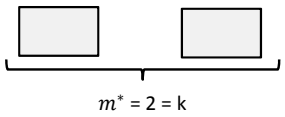
- per ogni *task* τ_i introduciamo τ_i^* che esegue quando τ_i è *idle* e viceversa
- due *schedule* Σ e Σ^* sono duali quando

  $\forall t, \tau_i \in \Sigma(t)$ se e solo se $\tau_i^* \notin \Sigma^*(t)$

Tale insieme di *task* viene eseguito nell'**architettura virtuale** del sistema originale (sistema duale)

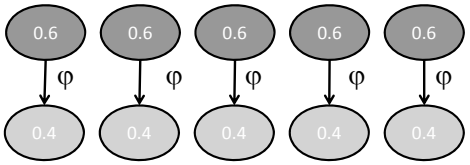
3

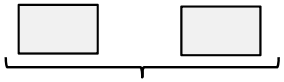
Esempio: S^*



4

Esempio: S^*

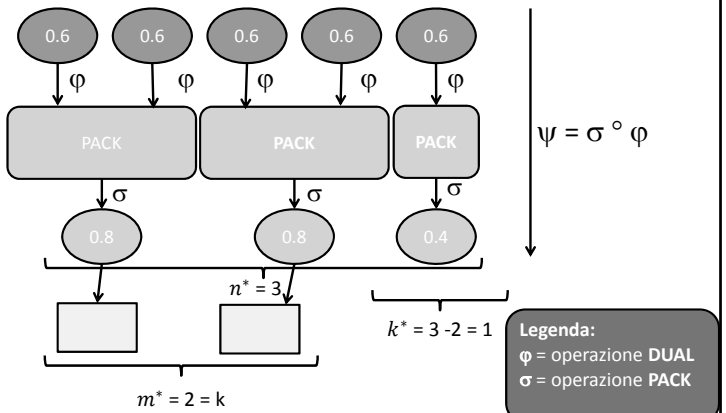




Legenda:
φ = operazione **DUAL**

5

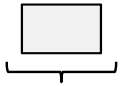
Esempio: S^*



Legenda:
φ = operazione **DUAL**
σ = operazione **PACK**

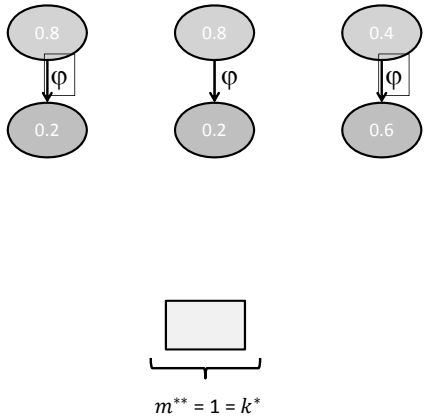
6

Esempio: S^{**}



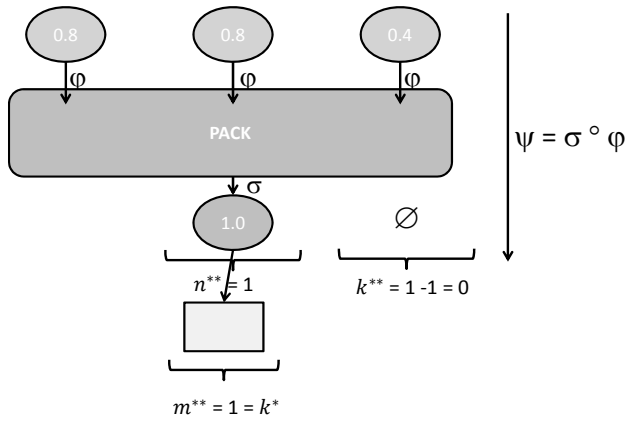
7

Esempio: S^{**}



8

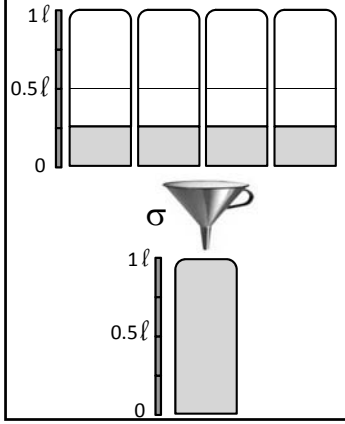
Esempio: S^{**}



9

Algoritmo di riduzione: terminazione

Lemma: $\psi = \left|\sigma \circ \varphi\,(U_1^4 \tau_i)\right| \leq \left\lceil \frac{|\tau|+1}{2} \right\rceil$

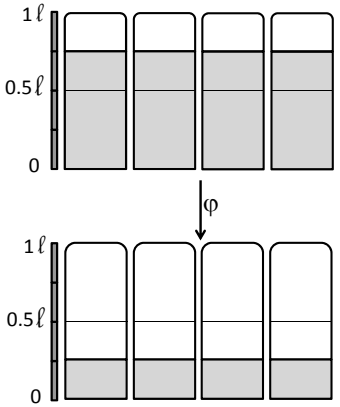


⇒ ψ -REDUCE (σ ° φ) ha termine, perché a ogni passo si riduce il *workload* (*rate* complessivo), e quindi anche il numero di processori necessari, e con σ -PACK almeno dimezzo il numero di task

Teorema terminazione: dopo un numero finito ***p*** di iterazioni l'algoritmo termina riducendo ad un sistema uniprocessore (*rate* complessivo uguale a uno)

Algoritmo di riduzione: terminazione

Lemma: $\psi = \left|\sigma \circ \varphi\,(U_1^4 \tau_i)\right| \leq \left\lceil \frac{|\tau|+1}{2} \right\rceil$



Intuizione

$\sum_1^4 R = 3 \Rightarrow$ m = 3
n = 4
k = 1 < m

Nel sistema duale
$\sum_1^4 R = 1 \Rightarrow m^* = 1 =$ **k**
$n^* = 1$
$k^* = 0$

Passi fondamentali di RUN

Riassumendo:

- Operazione DUAL (**φ**)
- Operazione PACK (**σ**)
- Operazione REDUCE (**ψ** = σ ° φ) riduce la dimensione del problema a ogni passo (~ dimezza)
- Teorema (validità del duale): Σ valido ⇔ Σ^* valido
- Poiché ogni *task* duale rappresenta il tempo *idle* del *task* primario, trovare lo *schedule* del sistema duale è equivalente a trovare quello del sistema primario

Scheduling

L'algoritmo genera un albero dalle foglie alla radice (fase *off-line*)

Lo *schedule* si ricava ripercorrendo a ritroso l'albero generato dall'algoritmo e applicando ad ogni istante *t* le seguenti regole (fase *on-line*)

- ai passi di tipo φ si usa la regola del duale (esegui in $\Sigma(t)$ i task che non eseguono in $\Sigma^*(t)$)
- ai passi di tipo σ si sceglie fra i *task* aggregati usando EDF

Il problema multiprocessore originario è stato quindi ricondotto a una collezione di problemi uniprocessore (**composizionalità** del problema)

PROXIMA

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, July 9th, 2014

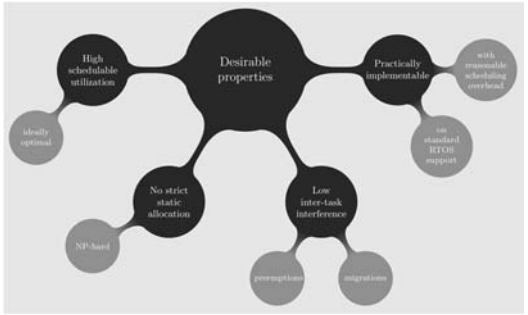
This project and the research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7 / 2007-2013] under grant agreement 611085

www.proxima-project.eu

Outline

- Motivation of our work
- Brief recall of Reduction to UNiprocessor
- RUN implementation and evaluation
- Conclusions and future work

Multiprocessor scheduling requisites



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

Multiprocessor scheduling state-of-the-art

Partitioned approaches	Global approaches	Hybrid approaches
Reduce to single-core scheduling with well-known solutions	Work-conserving Sustain relatively higher utilizations	Flexibility to attenuate the drawbacks of P- and G- approaches
Bin-packing → NP-hard	Large shared scheduling structures	More difficult to implement
In general cannot guarantee high utilization (50% bound)	Larger scheduling overheads (e.g., job migration)	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

Recall of the RUN algorithm

- Reduction to UNiprocessor (RTSS'11)
  - *Semi-partitioned* algorithm
  - *Optimal* without resorting to *proportionate fairness*
- Reduction principles
  - **Duality**

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

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

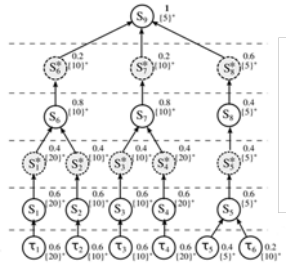
  - **Fixed-rate tasks** and **servers**

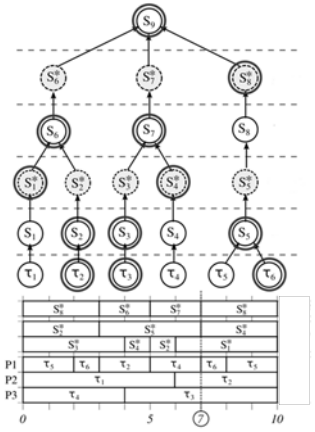
$$\tau_i \overset{\text{def}}{=\!=} (\mu_i, D_i) \Rightarrow S(\textstyle\sum_{\tau_i \in \mathcal{S}} \mu_i, \bigcup_{\tau_i \in \mathcal{S}} D_i)$$

- Scheduling decision taken on **reduction tree**
- Questions
  - *Can we implement it on standard RTOS support?*
  - *Does handling the reduction tree incur unacceptably large overhead?*

Scheduling on RUN

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





RUN implementation

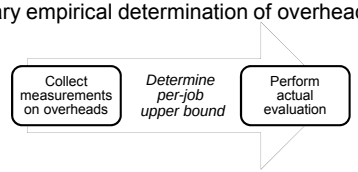
- **Successfully implemented**
  - On top of **LITMUS^{RT}** Linux test-bed (UNC)
  - Thus relying on an abstraction of *standard* RTOS support
- Main implementation choices and challenges
  - *Scheduling on the reduction tree*
    - How to organize the data structure
    - How to perform virtual scheduling and trigger tree updates
    - Intrinsic influence of the packing policy
  - *Mixing global and local scheduling*
    - Global release event queue vs. local *level-0* ready queue
    - Handling simultaneous scheduling events
      - Job release, budget exhaustion (possibly from different sub-trees)
  - *Meeting the full-utilization requirement*
    - Variability of tasks' WCET and lower utilization

Empirical evaluation

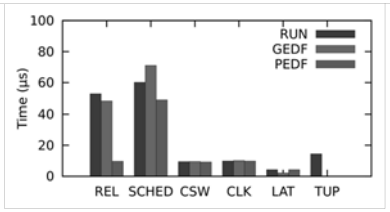
- **Empirical evaluation** instead of simulation-based
- Focus on **scheduling interference**
  - Cost of scheduling primitives
  - Incurred preemptions and migrations
- Compared RUN against **P-EDF** and **G-EDF**
  - RUN shares something in common with both
  - Had a preliminary evaluation on **Pfair** (S-PD^2 in LITMUS^{RT})
    - Inferior performance in terms of preemptions and migrations

Experimental setup

- **LITMUS^{RT}** on an 8-core AMD Opteron™ 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



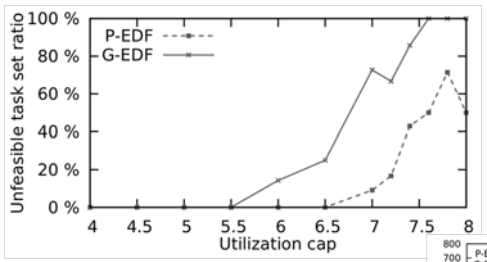
Primitive overheads and empirical bound

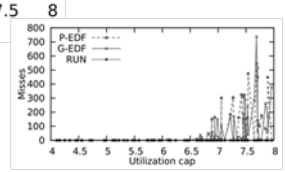


- Expectations confirmed
  - P-EDF needs lighter-weight scheduling primitives
- **Tree update** (TUP) triggered upon
  - *Budget exhaustion* event
  - Job release → REL includes TUP
- Empirical upper bound on RUN scheduling overhead
  - $OH^{Job}_{RUN} = REL + \widehat{SCHED} + CLK + k \times (TUP + \widehat{SCHED} + max(PRE, MIG))$

$k = \lceil (3p+1)/2 \rceil$ and $\widehat{SCHED} = SCHED + CSW + LAT$

Empirical schedulability

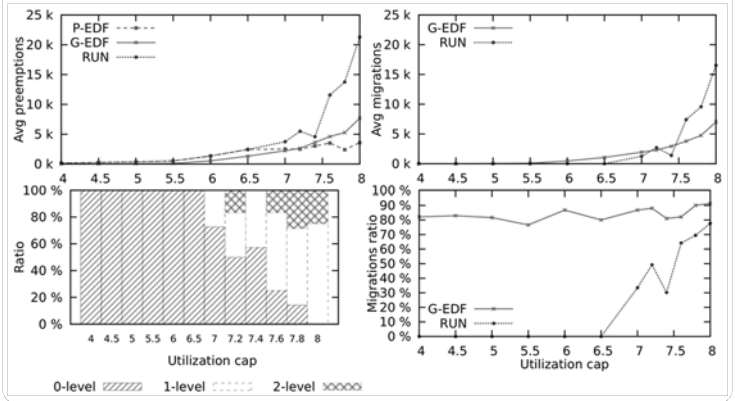




- Task sets exhibiting at least one miss
- RUN suffered **no misses**
  - Optimality and tailored overhead

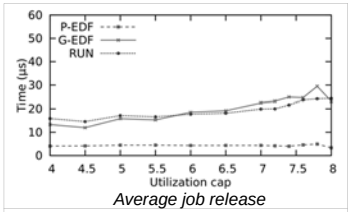
Kernel interference

- Observing average preemptions and migrations

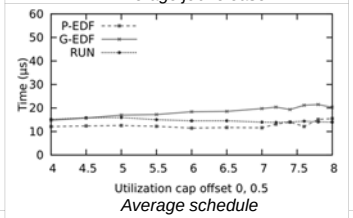


Scheduling cost

- Average cost of core scheduling primitives

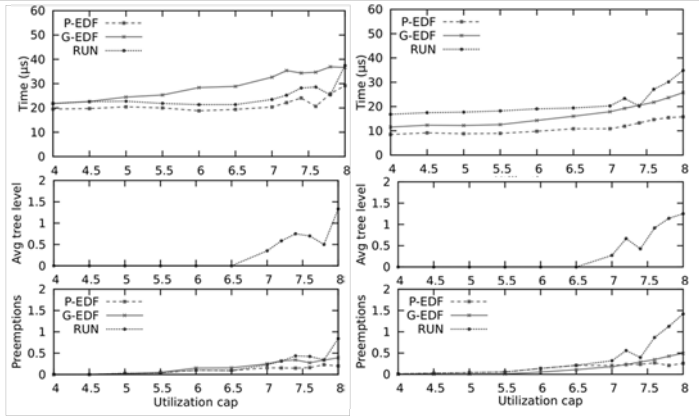


Average job release



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)

PROXIMA

Putting RUN into practice

Implementation and evaluation

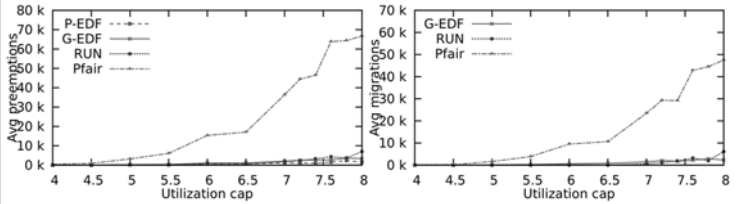
Davide Compagnin, Enrico Mezzetti and Tullio Vardanega
University of Padua - Italy

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

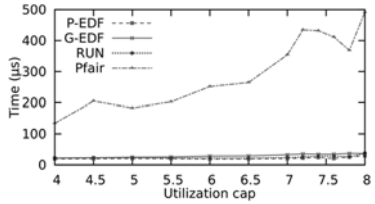
This project and the research leading to these results has received funding from the European Community's Seventh Framework Programme [FP7 / 2007-2013] under grant agreement 611085

www.proxima-project.eu

Evaluation against S-PD²



Observed preemptions and migrations



Per-job kernel overhead

Reduction tree data structure