7.b Seeking the lost optimality

Where we reflect more deeply into what became of optimality in the multicore world, and look at two ways to achieve it very differently from PFair

Rationale of the selection

- Between 2003 and 2016, multiple research efforts devised multicore scheduling algorithms capable of achieving optimality at lesser costs than with strict P-fairness
- We now look at two such results, which shine for their originality, and shed light on what really are the first principles for optimality in this world
  - Greg Levin *et al.* (2010), DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling
  - Paul Regnier *et al.* (2011), RUN: Optimal Multiprocessor Real-Time Scheduling via Reduction to Uniprocessor

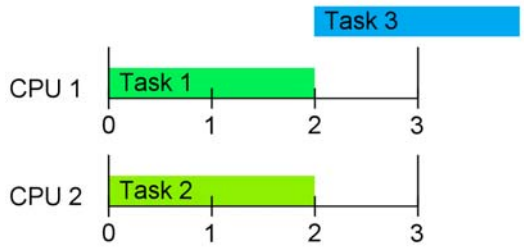
DP-FAIR: A Simple Model for Understanding Optimal Multiprocessor Scheduling

Greg Levin[†] Shelby Funk[‡] Caitlin Sadowski[†]
Ian Pye[†] Scott Brandt[†]
[†]University of California Santa Cruz
[‡]University of Georgia Athens

Problem

Partitioned Schedulers Cannot Be Optimal

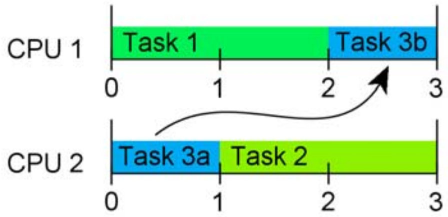
- Example: 2 processors ($m = 2$); 3 tasks, each with 2 units of work required every 3 time units: (3,2)



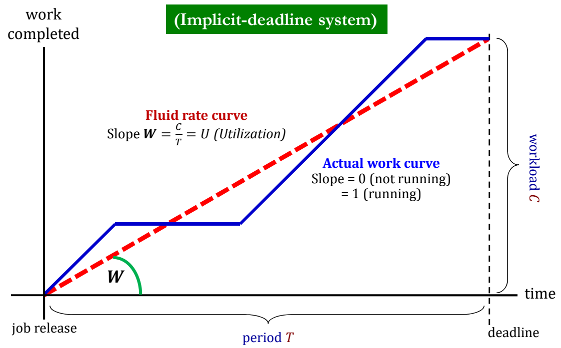
Global Schedulers May Succeed

- Same example, same taskset

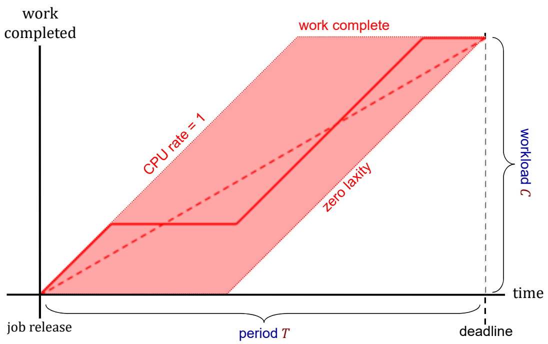
Task 3 may now *migrate* between processors



Fluid Rate Curve



Feasible Work Region



The Grand Challenge (*Mark 1*)

- Design an *optimal* scheduling algorithm for periodic task sets on *multiprocessors*
  - A task set is *feasible* if there exists a schedule that meets all deadlines
  - A scheduler is *optimal* if it can always schedule any feasible task set

Necessary and Sufficient Conditions

- Any set of (independent) tasks needing at most
  - 1 processor for each task τ_i $(\forall i = 1,..,n: U_i \leq 1)$
  - m processors for all tasks $(\sum_i U_i \leq m)$

  is feasible
- **Proof**: small scheduling intervals can approximate the fluid rate curve
  - **Status**: solved (on paper). P-Fair (1996) was the first such optimal algorithm
  - At what cost?

The Grand Challenge (*Mark 2*)

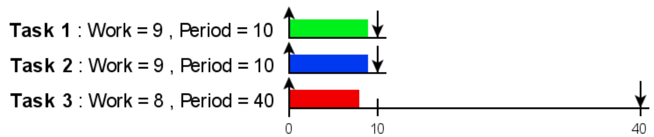
- Design an *optimal* scheduling algorithm with *fewer* context switches and migrations
  - Finding a feasible schedule with *the fewest* migrations is NP-Complete!

The Grand Challenge (*Mark 2*)

- Design an *optimal* scheduling algorithm with *fewer* context switches and migrations
- Status: *solved*, **but …**
  - With solutions that are complex and confusing
- **Our Contributions:** A *simple, unifying theory* for optimal global multiprocessor scheduling *and* a simple optimal algorithm

Greedy Algorithms Fail on Multiprocessors /1

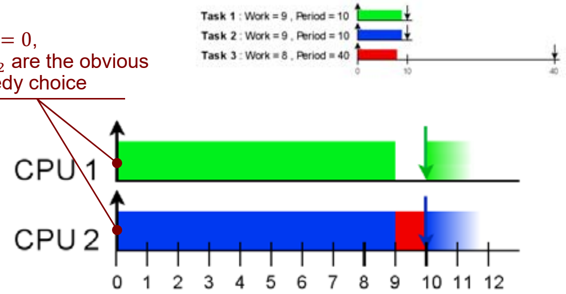
- Example $(n = 3, m = 2)$, implicit deadlines



Utilization: 9/10 + 9/10 + 8/40 = 2

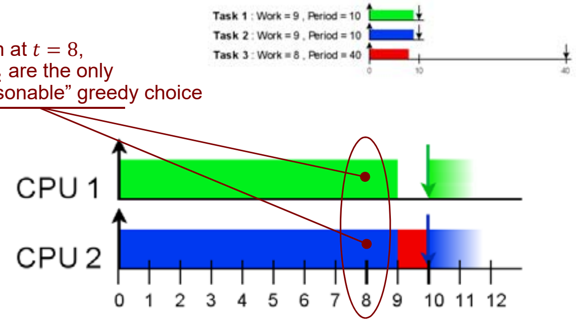
Greedy Algorithms Fail on Multiprocessors /2

At $t = 0$,
τ_1, τ_2 are the obvious greedy choice



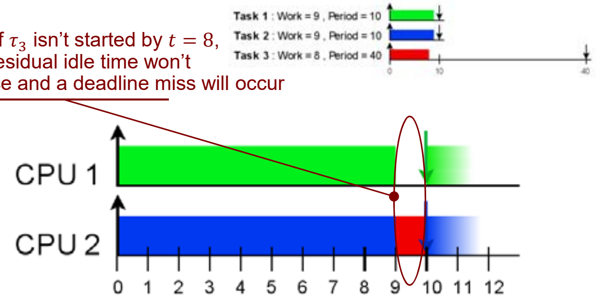
Greedy Algorithms Fail on Multiprocessors /3

Even at $t = 8$,
τ_1, τ_2 are the only "reasonable" greedy choice



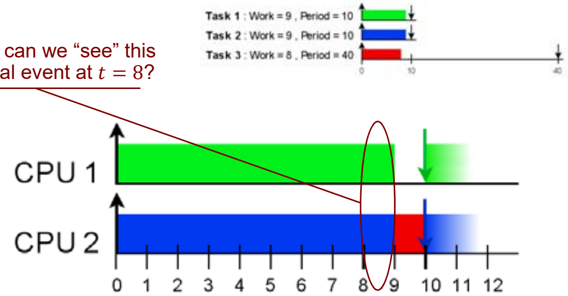
Greedy Algorithms Fail on Multiprocessors /4

Yet, if τ_3 isn't started by $t = 8$, the residual idle time won't suffice and a deadline miss will occur



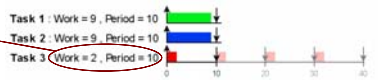
Greedy Algorithms Fail on Multiprocessors /5

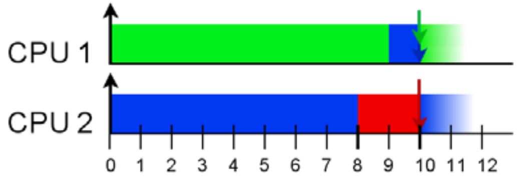
How can we "see" this critical event at $t = 8$?



Proportioned Algorithms Succeed on Multiprocessors /1

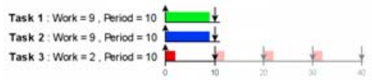
Subdivide τ_3 in $\frac{T_3}{T_{i=1,2}} = 4$ subtasks with the *same* period as τ_1, τ_2 and proportional workload $\frac{C_3}{4} = 2$

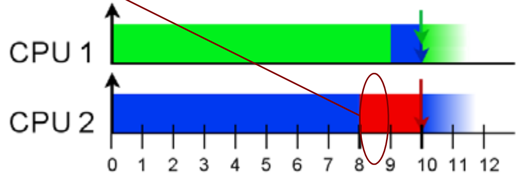




Proportioned Algorithms Succeed on Multiprocessors /2

The new τ_3 has a ***zero-laxity*** event at $t = 8$





Proportional Fairness

- **Insight:** scheduling is easier when all jobs have the same deadline

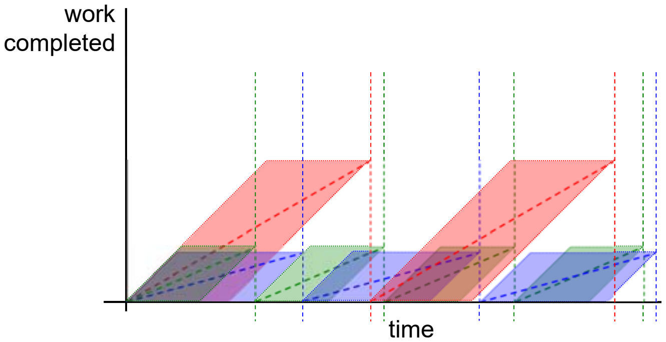
Theorem [Hong, Leung: RTSS 1988, IEEE TCO 1992]
No optimal on-line scheduler can exist for a set of jobs with two or more distinct deadlines on any ($m > 1$) multiprocessor system

- **Application:** apply all deadlines to all jobs
  - Assign workloads proportional to utilization
  - Work complete matches fluid rate curve *at every system deadline*

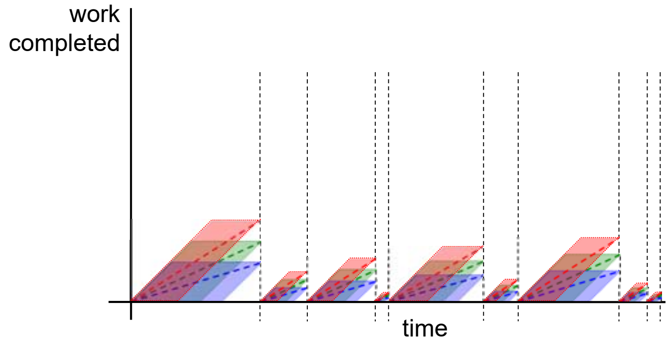
Proportional Fairness is the Key

- All optimal algorithms enforce proportional fairness at all deadlines
  - **P-Fair** (1996): the extreme: proportional fairness at all times
  - **BF, Boundary Fair**
    - D. Zhu, D. Mossé, and R. Melhem, *Multiple-Resource Periodic Scheduling Problem: how much fairness is necessary?*, RTSS, **2003**
  - **LLREF, Largest Local Remaining Execution time First**
    - H. Cho, B. Ravindran, E.D. Jensen, *An Optimal Real-Time Scheduling Algorithm for Multiprocessors*, RTSS, **2006**
  - **EKG, EDF with task splitting and k processors in a group**
    - B. Andersson, E. Tovar, *Multiprocessor Scheduling with Few Preemptions*, RTCSA, **2006**
- Why do they all use proportional fairness?

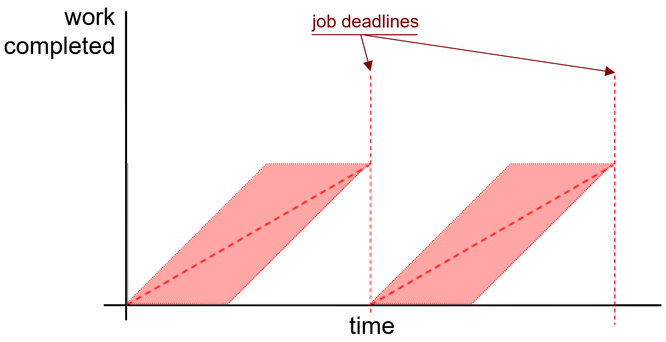
Scheduling Multiple Tasks is Complicated



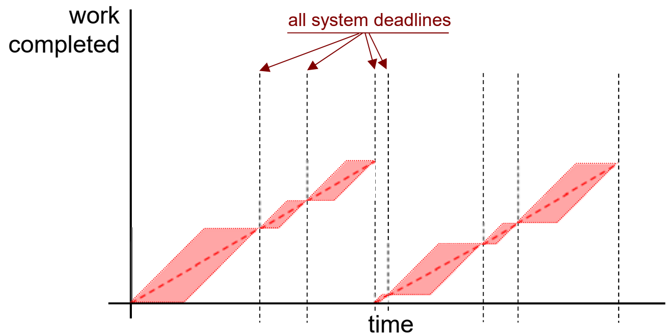
Scheduling Multiple Tasks with Same Deadline is Easy



Actual Feasible Regions



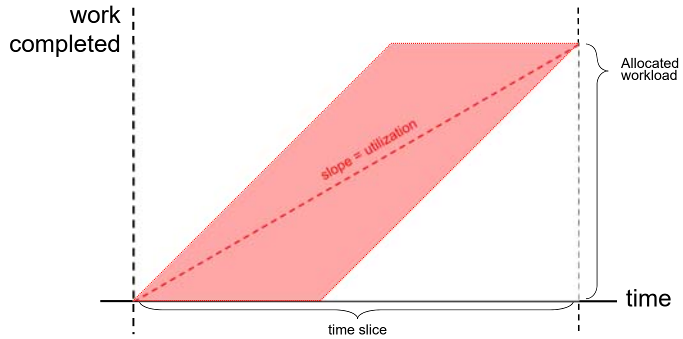
Restricted Feasible Regions Under Deadline Partitioning



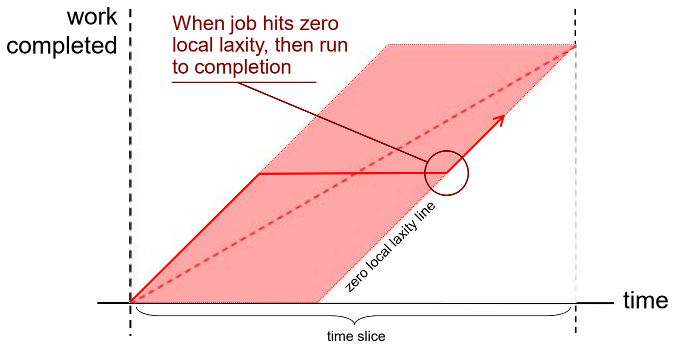
The DP-Fair Scheduling Policy

- Partition time into *slices* based on all system deadlines
- Allocate each job a per-slice workload equal to its utilization × the length of the slice
- Schedule jobs within each slice in any way that obeys the following three rules:
  1. Always run a job with zero *local laxity*
  2. Never run a job with no workload remaining in the slice
  3. Do not voluntarily allow more idle processor time than $(m - \sum U_i) \times$ (*length of slice*)

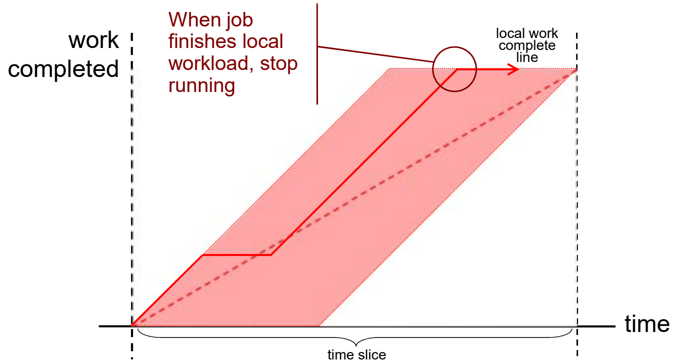
DP-Fair Work Allocation



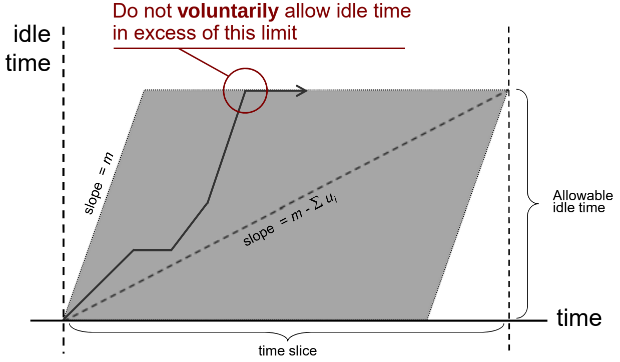
DP-Fair Scheduling Rule #1



DP-Fair Scheduling Rule #2



DP-Fair Scheduling Rule #3



DF-Fair Guarantees Optimality

- We say that a scheduling algorithm is *DP-Fair* if it follows these three rules
- **Theorem:** Any DP-Fair scheduling algorithm for periodic tasks is optimal

DP-Fair Implications

- (Partition time into slices)
  \+ (Assign proportional workloads)

  Optimal scheduling is almost trivial
  - Minimally restrictive rules allow great latitude for algorithm design and adaptability
- What is the simplest possible algorithm?

(EXAMPLE OF EXAM ASSIGNMENT) UNDERSTANDING THE RUN ALGORITHM

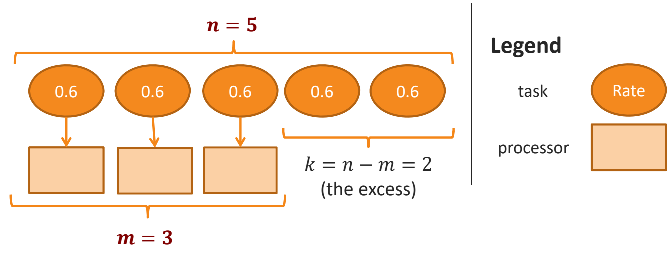
PhD seminar on Real-Time Systems, University of Bologna, July 2014

RUN Assumptions

Model parameters

- $m > 1$ homogeneous (symmetric) processors
- n implicit-deadline, independent, periodic tasks $\tau_i, i \in \{1..n\}$
- $n = m + k, k \geq 0$
- Fixed-rate tasks $U_i = \frac{C_i}{T_i}$ $\sum_{i=1}^{n} U_i \leq m$
- Fully utilized system: no idle time (add filler task if needed)
- *Migration* and *preemption* costs included in c_i

Example /1



- $U_i = 0.6 \;\; \forall \tau_i, i = \{1, \dots, n = 5\}$
- $\sum_{i=1}^{n} U_i = 3 = m$ (fully utilized system)
- What schedule $\sum$ for $\mathbf{S} = \{\{\boldsymbol{\tau_i}\}, \boldsymbol{m}\}$?

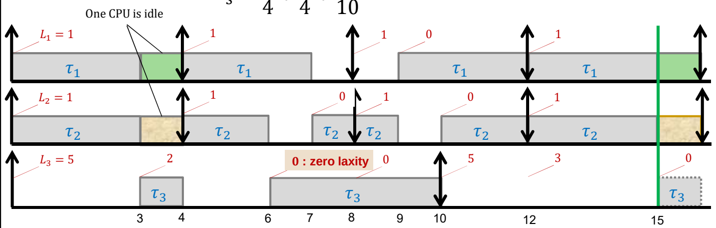
Duality

- The (primal) problem of scheduling $\boldsymbol{S = \{\tau_1 = (c_1, T_1), \dots, \tau_n = (c_n, T_n)\}, m}$ has a *dual* problem that consists of scheduling $\boldsymbol{S' = \{\tau_1' = (T_1 - c_1, T_1), \dots, \tau_n' = (T_n - c_n, T_n)\}, (n - m)}$
- With this definition of duality
  - *Laxity in primal is work remaining in the dual*
  - *A work-complete event in the primal is zero-laxity in the dual*
  - *And vice versa*
- **Corollary**: any scheduling problem with $\boldsymbol{m}$ processors, $\boldsymbol{n = m + 1}$ tasks, and $\sum_1^n U_i = \boldsymbol{m}$ may be scheduled by applying EDF to its uniprocessor dual
  - If we can schedule n tasks on m processors, then we can also schedule the dual of those n tasks on $n - m$ processors
  - This is so because the scheduling events in the dual system map to scheduling events in the primal system

The G-LLF example at page 372 …

$$S = \{\tau_1 = (3,4), \tau_2 = (3,4), \tau_3 = (5,10)\}, \boldsymbol{n = 3}, H_S = 20$$

$$U_s = \frac{3}{4} + \frac{3}{4} + \frac{5}{10} = 2.0 \;\rightarrow \boldsymbol{m = 2}$$

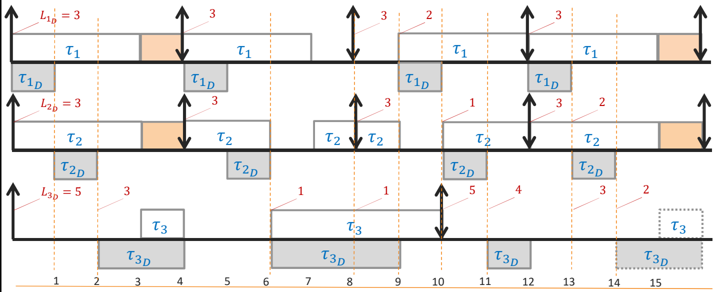


- At $t = 15$ the CPU time remaining is $T_R = m \times (H_S - t) = \mathbf{10}$
- Yet, the time needed is $T_N = e_1 + e_2 + e_3 = \mathbf{11}$

Duality solves that unsolvable problem

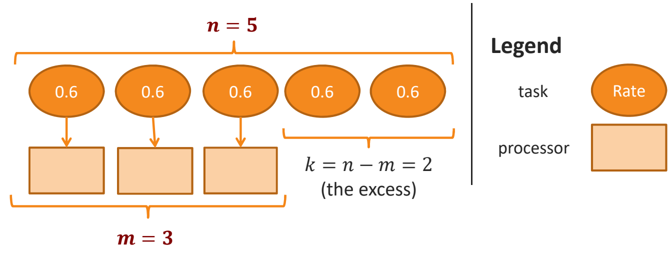
$S = \{\tau_1 = (3,4), \tau_2 = (3,4), \tau_3 = (5,10)\}, n = 3, U_s = \frac{3}{4} + \frac{3}{4} + \frac{5}{10} = 2.0 \rightarrow m = 2$

$S_D = \{\tau_{1_D} = (1,4), \tau_{2_D} = (1,4), \tau_{3_D} = (5,10)\}, U_{S_D} = \frac{1}{4} + \frac{1}{4} + \frac{5}{10} = 1.0 = n - m$



The dual (LLF) schedule leaves no idle time

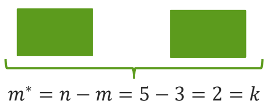
Example /1



- $U_i = 0.6 \ \forall \tau_i, i = \{1, \ldots, n = 5\}$
- $\sum_{i=1}^{n} U_i = 3 = m$ (fully utilized system)
- What schedule Σ for $S = \{\{\tau_i\}, m\}$?

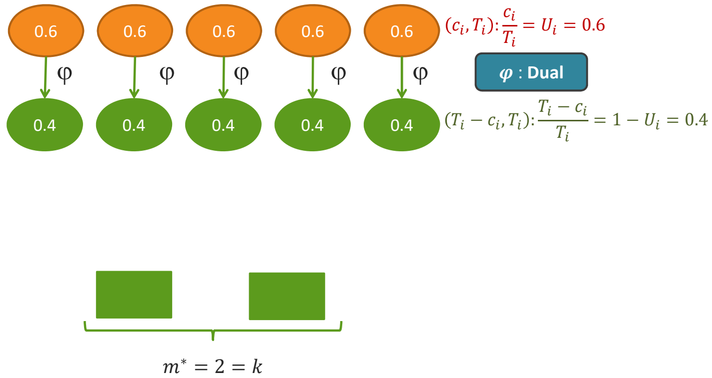
Example /2

- Consider the dual of this $\{n = 5, m = 3\}$ system

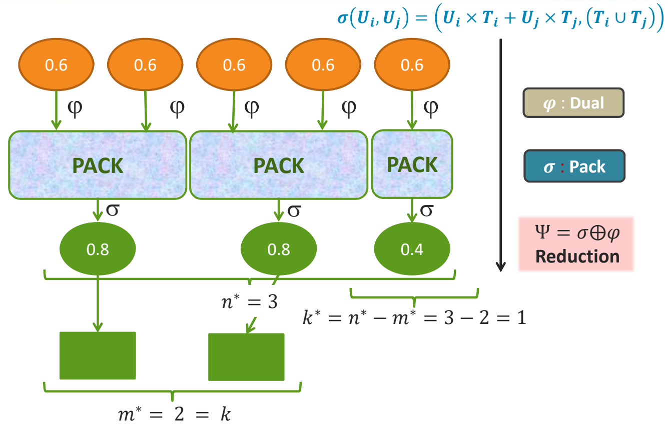


The dual should run on m^* processors

Example /3

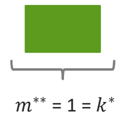


Example /4

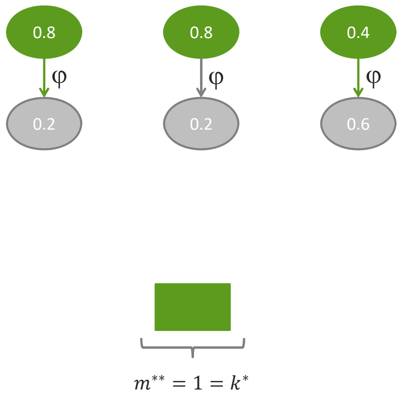


Example /5

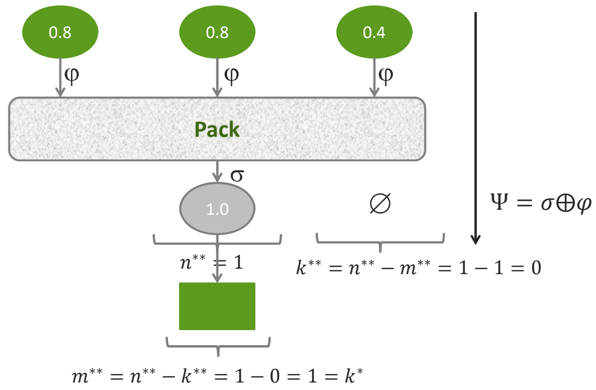
The $(n^* = 3, m^* = 2)$ system still cannot be partitioned feasibly
Yet, applying duality to it seems promising since the dual would need $n^* - m^* = 1$ processor, which would **R**EDUCE the problem TO a **UNI**PROCESSOR case



Example /6

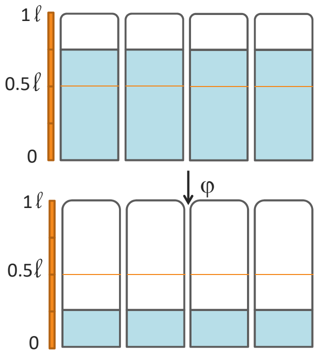


Example /7



Why does reduction terminate? /1

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



Intuition

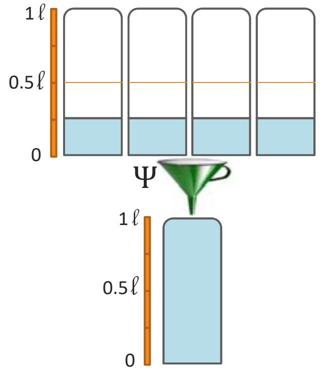
$\sum_1^{n=4} U_i = 3 \Rightarrow m = 3$
$\boldsymbol{k} = n - m = 4 - 3 = 1$

In the dual system

$\sum_1^4 U_i^* = n - m = 1 \Rightarrow m^* = 1$
$n^* = 1$ (*after packing*)
$k^* = n^* - m^* = 0$ *no leftover*

Why does reduction terminate? /2

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



- Reduction $\Psi = \sigma \oplus \varphi$ terminates as every step of it lowers the residual workload and the # of processors needed to run it
- The packing operation (at least) halves the number of tasks to schedule
- **Termination theorem:** after a finite number ***p*** of reduction steps, the system is reduced to a uniprocessor with full workload

How does RUN work /1

- A pair of basic operators
  - φ: **Dual**
  - σ: **Pack**
- A higher-order $\Psi = \sigma \oplus \varphi$: **Reduce** operation lowers (~ halves) the size of the problem at every step
- **Theorem** (validity of the dual): Σ valid ⇔ Σ* valid
- Since every dual task represents the idle time of its primal, finding a feasible schedule for the dual (which is easier) determines a feasible schedule for its primal

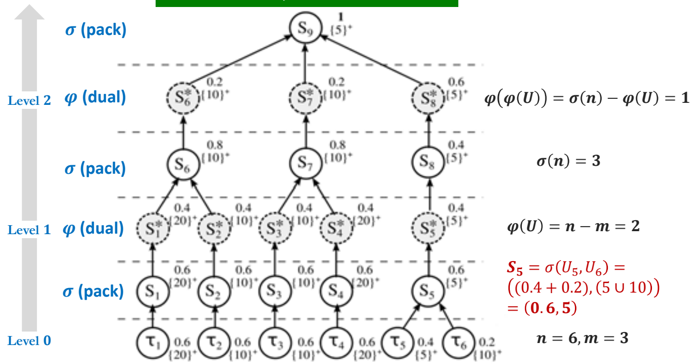
How does RUN work /2

Algorithm 1: Outline of the RUN algorithm

I. OFF-LINE:
A. Generate a reduction sequence for $\mathcal{T}$;
B. Invert the sequence to form a server tree; — Servers are aggregates of tasks
C. For each proper subsystem $\mathcal{T}'$ of $\mathcal{T}$;
Define the client/server at each virtual level; — Each task in a server is a client of it
II. ON-LINE:
Upon a scheduling event: ;
A. If the event is a job release event at level 0 ;
1. Update deadline sets of servers on path up to root;
2. Create jobs for each of these servers accordingly;
B. Apply Rules 1 & 2 to schedule jobs from root to leaves, determining the m jobs to schedule at level 0;
C. Assign the m chosen jobs to processors, according to some task-to-processor assignment scheme;

Example: off-line phase

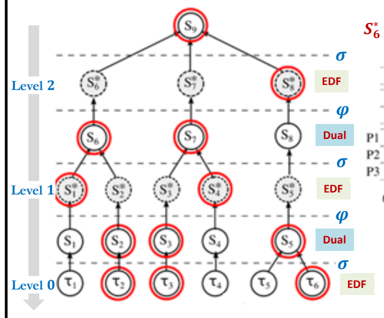
Reduction Tree, built from the leaves

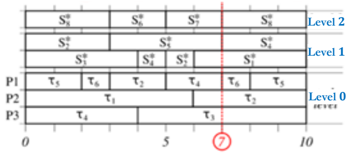


Example: on-line phase (seen at time $t = 7$)

Reduction Tree, used from the root

$S_6^* = (2,10), S_7^* = (2,10), S_8^* = (3,5) \rightarrow U^* = 1.0$





At $t = 7$, S_8^* should be scheduled
Its dual, S_8, should *not* run, thus $S_6 = \sigma(S_1^*, S_2^*)$, $S_7 = \sigma(S_3^*, S_4^*)$ should, with $U = 2$ and, at $t = 7$, S_1^*, S_4^* should run simultaneously
Hence, their respective dual, S_1, S_4 should *not*, but S_2, S_3 and S_5 (dual of S_8) should, with $U = 3$, which means τ_2, τ_3 and τ_6 run at $t = 7$

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, 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

www.proxima-project.eu

RUN implementation

- **For real**
  - On top of **LITMUS^RT** Linux test-bed (UNC, now MP-SWI)
  - Relying on *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 less-than-full utilization

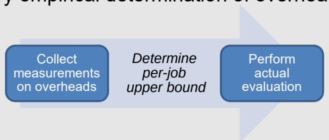
Empirical evaluation

- Real implementation instead of plain simulation
- Focus on **scheduling interference**
  - Cost of scheduling primitives
  - Incurred preemptions and migrations
- RUN compared against **P-EDF** and **G-EDF**, already native on **LITMUS**^{RT}
  - RUN shares something in common with both
- Much better than **Pfair** (S-PD^2 in LITMUS^{RT})
  - RUN has superior performance for preemptions and migrations

446 09/07/2014 PROXIMA

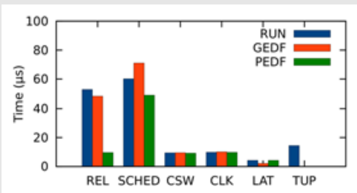
Experimental setup

- **LITMUS**^{RT} on an 8-core AMD Opteron^{TM} 2356
- Measurement runs 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



447 09/07/2014 PROXIMA

Primitive overheads and empirical bound



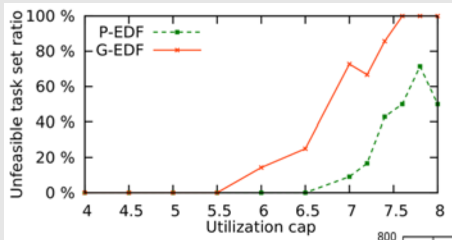
- Expectations confirmed
  - P-EDF needs lighter-weight scheduling primitives
  - RUN reduces to P-ED when a perfect portioning exists
- **Tree update** (TUP) triggered upon
  - *Budget exhaustion* event
  - Job release → REL includes TUP
- Empirical upper bound on RUN scheduling overhead

$$OH^{Job}_{RUN} = REL + \widehat{SCHED} + CLK + k \times (TUP + \widehat{SCHED} + max(PRE, MIG))$$

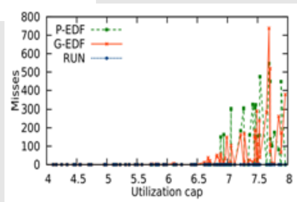
$$k = \lceil (3p+1)/2 \rceil \qquad \widehat{SCHED} = SCHED + CSW + LAT$$

448 09/07/2014 PROXIMA

Empirical schedulable utilization



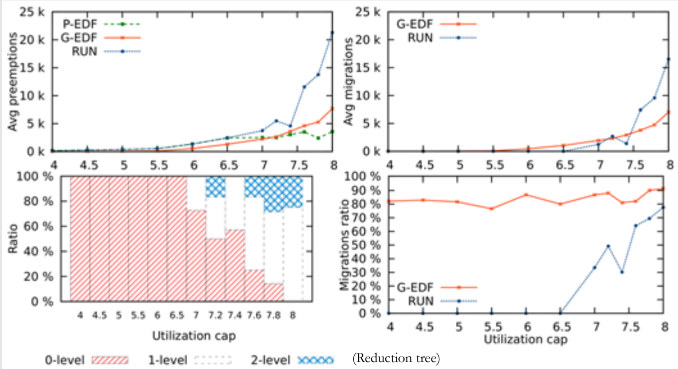
- In either P- or G-EDF, some task sets exhibited deadline misses
- RUN suffered **no misses** ever
  - Empirical evidence of optimality



449 09/07/2014 PROXIMA

Kernel interference

- Average preemptions and migrations



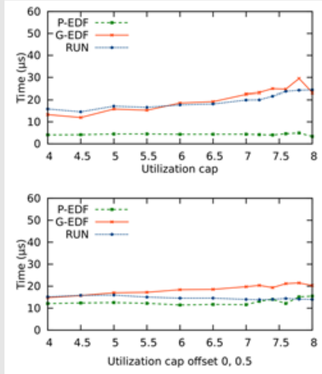
450 09/07/2014 PROXIMA

Scheduling cost

- Average cost of core scheduling primitives

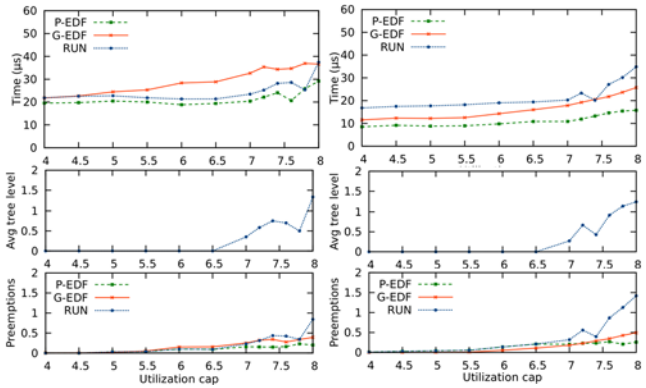
Average job release

Average schedule



451 09/07/2014 PROXIMA

Per-job scheduling overhead

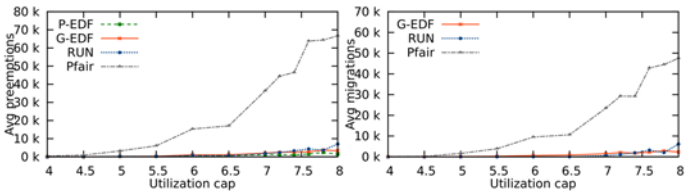


Harmonic task set

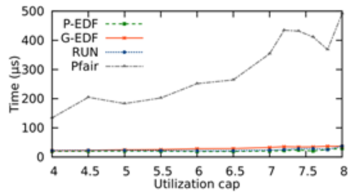
Non-harmonic task set

452 09/07/2014 PROXIMA

Evaluation against S-PD^2 (Pfair)



Observed preemptions and migrations



Per-job kernel overhead

453 09/07/2014 PROXIMA

Summary

- The DP-Fair algorithm shows us that optimal scheduling for multicore processors need *not* be greedy and instead can dispatch *parsimoniously*
  - This algorithm proved very difficult to implement, surprisingly, owing to the lack of adequate RTOS support
- The RUN algorithm shows us how the principle of *duality* allows reducing multicore scheduling to a (simple) uniprocessor case
  - This algorithm, although so unusual, was easier to implement and proved as efficient as on paper

Selected readings

- S. Funk, G. Levin, G., *et al.* (**2011**)
  DP-FAIR: a unifying theory for optimal hard real-time multiprocessor scheduling
  DOI: 10.1007/s11241-011-9130-0
- E. Massa, G. Lima, P. Regnier (**2016**)
  From RUN to QPS: new trends for optimal real-time multiprocessor scheduling
  DOI: 10.1504/IJES.2016.080390
- D. Compagnin, E. Mezzetti, T. Vardanega (**2014**)
  Putting RUN into Practice: Implementation and Evaluation
  DOI: 10.1109/ECRTS.2014.27