AM-Red: An Optimal Semi-Partitioned Scheduler Assuming Arbitrary Affinity Masks

Sergey Voronov and James H. Anderson

University of North Carolina at Chapel Hill

February 14, 2018

Definitions: Jobs and Tasks

Job (a single unit of work)

Release time t_r, relative deadline D (absolute one is $t_d = D + t_r$) and execution time C.
Tardiness of job, completed at t_c: $\max(0, t_c - t_d)$.

Sporadic Task (a sequence of similar dependent jobs)

Characterized by period T and Worst-Case Execution Time (WCET) C.
Releases jobs $j_1, ..., j_k, ...$, such that

- Distance between releases of j_i and j_{i+1} is at least T.
- Relative deadline of j_i is T time units from the release.
- WCET of j_i does not exceed C.

Task tau_i utilization is $U_i = C/T$, task tardiness is supremum of all its jobs tardiness.

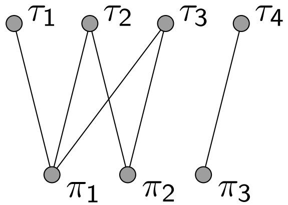
Definitions: Affinity Masks

Affinity Mask α_i of task τ_i

A set of processors, that can execute given task τ_i (any processor that is not in the mask cannot execute the task).

Affinity Graph $AG(\tau)$

n vertices $\tau_1, \ldots, \tau_n$ (representing tasks)
m vertices $\pi_1, \ldots, \pi_m$ (representing cores)
has an edge (τ_i, π_j) if and only if $\pi_j \in \alpha_i$
(i.e., task τ_i can execute on core π_j).



$\alpha_1 = \{1\}$, $\alpha_2 = \{1, 2\}$, $\alpha_3 = \{1, 2\}$, $\alpha_4 = \{3\}$.

Definitions: HRT/SRT-schedulability

Schedulability of τ

τ is *HRT-schedulable* (resp., *SRT-schedulable*) under scheduling algorithm S if each task in τ has zero (resp., bounded) tardiness in any schedule for τ generated by S.

Feasibility of τ

τ is *HRT-feasible* (resp., *SRT-feasible*) if, for any job release sequence, a schedule exists in which each task has zero (resp., bounded) tardiness.

Optimality of Scheduler S

S is *HRT-optimal* (resp., *SRT-optimal*) if every HRT-feasible (resp., SRT-feasible) task set τ is HRT-schedulable (resp., SRT-schedulable) under S.

Problem and Motivation

Affinity masks usage

- simplify cache usage analysis
- reduce I/O-related overheads (interrupts)
- easier load balancing
- ensure security isolation

Problem and Motivation

Affinity masks usage

- simplify cache usage analysis
- reduce I/O-related overheads (interrupts)
- easier load balancing
- ensure security isolation

Affinity masks support

- Linux's SCHED_DEADLINE policy (RT scheduler)
- Windows, Mac OS X
- FreeRTOS, QNX, VxWorks, ...

Problem and Motivation

Affinity masks usage

- simplify cache usage analysis
- reduce I/O-related overheads (interrupts)
- easier load balancing
- ensure security isolation

Affinity masks support

- Linux's SCHED_DEADLINE policy (RT scheduler)
- Windows, Mac OS X
- FreeRTOS, QNX, VxWorks, ...

Affinity masks schedulers

- [Baruah 2013]: HMR, requires future knowledge
- [Brandenburg 2014]: improvement for Linux scheduler, HMR
- [Bonifaci 2016]: HMR, only hierarchical, high overheads

HMR = high migrations rate

No optimal scheduler, available for implementation, exist.

Problem and Motivation

Our goals:

- Build *HRT/SRT-optimal* scheduler that supports *arbitrary* affinity masks
- Make scheduler *as fast as possible* (in the worst case)
- Reduce number of task *migrations* over cores

Problem and Motivation

Our goals:

- Build *HRT/SRT-optimal* scheduler that supports *arbitrary* affinity masks
- Make scheduler *as fast as possible* (in the worst case)
- Reduce number of task *migrations* over cores

What do we do:

- Restrict affinity masks, preserving task set feasibility (affinity graph reduction)

Static vs Dynamic Schedulers

	Static	Dynamic
How to build	Create schedule template Generate "almost static" schedule	No preprocessing Decide at runtime tasks-cores placement
Pros	Easy to analyze Small overheads	Easy to implement No pre-processing
Cons	Hard to adjust schedule Offline phase may take lots of time	Hard to analyze High overheads

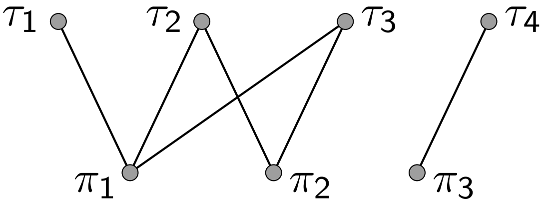
Frame is a template of schedule over $[0, 1)$. We scale this template to $[0, F)$ and repeat.

Schedulability Conditions: Overview

	Baruah	UB Test	Max Flow
Test	$\forall i: \sum_{j \in \alpha_i} x_{ij} = 1$ $\forall j: \sum_{i} x_{ij} U_i \leq 1$ $\forall i,j: x_{ij} \geq 0$	$\forall S \subset \tau : U_S \leq \alpha_\tau$ For every subset of tasks, its aggregated affinity mask size is at least its utilization	$\lvert f \rvert = U$. Maximal flow over specially constructed graph $FN(\tau)$ is equal to system utilization
Complexity	$\tilde{O}(mn \cdot (m+n)^{2.9})$	$mn \cdot 2^n$	$\tilde{O}(mn\sqrt{m+n})$

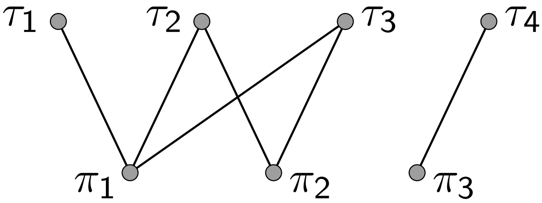
$\tilde{O}$ ignores logarithmic factors: $\tilde{O}(g(n)) = O(g(n) \log^k g(n))$ for some natural number k.

Schedulability Conditions: Utilization Balance

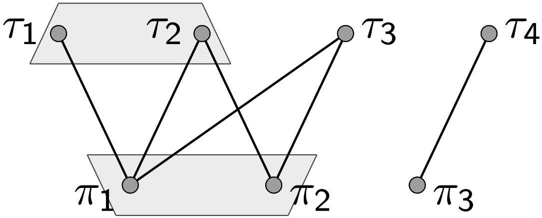


(a) Affinity Graph $AG(\tau)$

Schedulability Conditions: Utilization Balance



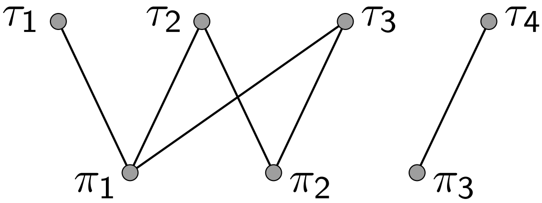
(a) Affinity Graph $AG(\tau)$



(b) Aggregated affinity mask of $\{\tau_1, \tau_2\}$.

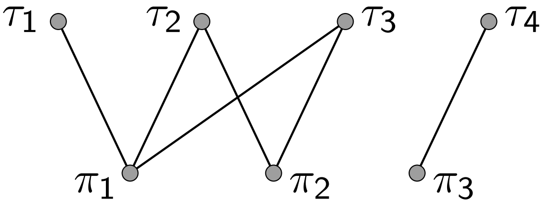
Utilization Balance Feasibility Test check for $S = \{\tau_1, \tau_2\}$: $U_1 + U_2 \leq 2$.

Schedulability Conditions: Max Flow Test

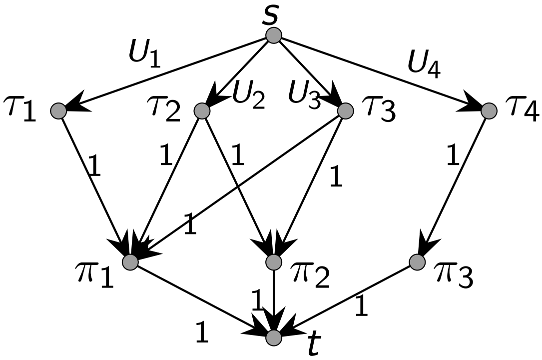


(a) Affinity Graph $AG(\tau)$

Schedulability Conditions: Max Flow Test



(a) Affinity Graph $AG(\tau)$



(b) Flow Network $FN(\tau)$

Max Flow Feasibility Test: the maximum flow over $FN(\tau)$ is U.

AM-Red Outline

- Run Max Flow Test
- Remove cycles from Share Graph
- Build extended frame
- Build frame

AM-Red = Affinity Masks REDuction

From Flow Solution to Frame: Share Graph

Max Flow Test passed: $|f| = U$

$f(\tau_i, \pi_j)$ is a flow between τ_i and π_j.

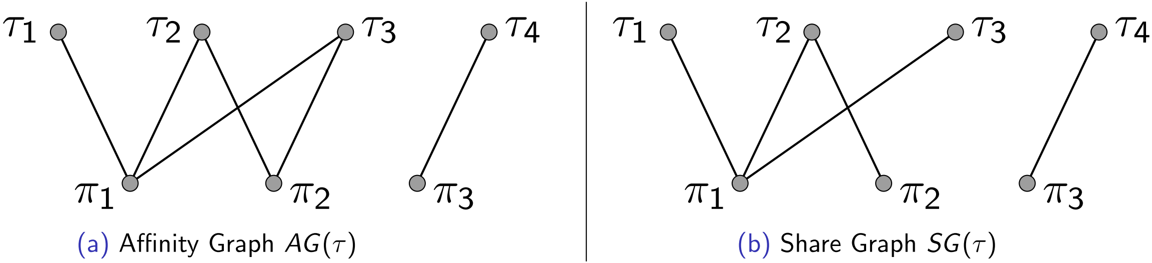
From Flow Solution to Frame: Share Graph

Max Flow Test passed: $|f| = U$

$f(\tau_i, \pi_j)$ is a flow between τ_i and π_j.

Share Graph $SG(\tau)$

Affinity graph $AG(\tau)$ with (τ_i, π_j) edges removed, if $f(\tau_i, \pi_j) = 0$.

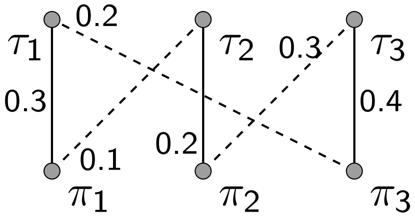


(a) Affinity Graph $AG(\tau)$ (b) Share Graph $SG(\tau)$

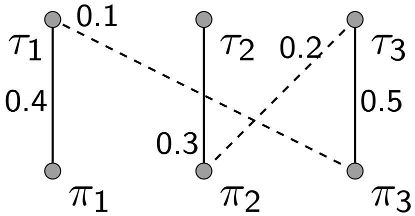
Figure: $f(\tau_3, \pi_2) = 0$, all others edges in $AG(\tau)$ have non-zero f values.

From Flow Solution to Frame: Cycles Removal

If Share Graph $SG(\tau)$ has cycles we can remove them one-by-one.
f_m is minimum of $f(\tau_i, \pi_j)$ over all cycle edges.



(a) Cycle in Share Graph before removal.



(b) "Cycle" in Share Graph after removal.

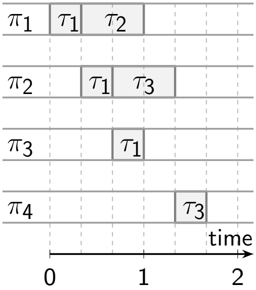
Figure: For dashed edges $f(\tau_i, \pi_j)$ decreases, for solid it increases. $f_m = f(\tau_2, \pi_1) = 0.1$.

From Flow Solution to Frame: Extended Frame

Let $I_E(\tau_i, \pi_j)$ be the union of all continuous intervals on core π_j allocated to task τ_i.

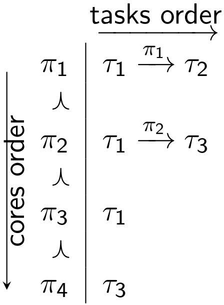
	τ_1	τ_2	τ_3
π_1	1/3	2/3	-
π_2	1/3	-	2/3
π_3	1/3	-	-
π_4	-	-	1/3

$$\text{extended frame}\begin{cases} \forall i,j: & I_E(\tau_i,\pi_j) \text{ is a single continuous interval} \\ \forall i: & \bigcup_j I_E(\tau_i,\pi_j) \text{ is a single continuous interval} \\ \forall j: & \bigcup_i I_E(\tau_i,\pi_j) \text{ is a single continuous interval} \\ \left.\begin{array}{ll} \forall i,j: & |I_E(\tau_i,\pi_j)| = f(\tau_i,\pi_j) \text{ (correct capacity)} \\ \forall i,j_1,j_2: & I_E(\tau_i,\pi_{j_1}) \cap I_E(\tau_i,\pi_{j_2}) = \emptyset \text{ (no overlapping)} \end{array}\right\} \text{frame} \end{cases}$$

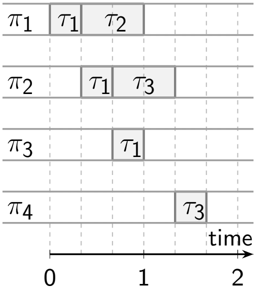


From Flow Solution to Frame: Extended Frame Construction

for $\pi_j \in$ cores order **do**
    for $\tau_i \in$ tasks order for π_j **do**
        allocate interval for τ_i on π_j
        (right after previous one)



	τ_1	τ_2	τ_3
π_1	$1/3$	$2/3$	-
π_2	$1/3$	-	$2/3$
π_3	$1/3$	-	-
π_4	-	-	$1/3$



From Flow Solution to Frame: Proper Order Construction

tasks order

cores order

$$\begin{array}{c|l} \pi_1 & \tau_1 \xrightarrow{\pi_1} \tau_2 \\ \curlywedge & \\ \pi_2 & \tau_1 \xrightarrow{\pi_2} \tau_3 \\ \curlywedge & \\ \pi_3 & \tau_1 \\ \curlywedge & \\ \pi_4 & \tau_3 \end{array}$$

Proper Order properties:

$\forall i, j$: τ_i appears in task order of core π_j if and only if $f(\tau_i, \pi_j) > 0$

$\forall i$: task τ_i can be non-first task on at most one core

$\forall i$: if task τ_i is non-first on core π_j, then on previous cores τ have not appeared

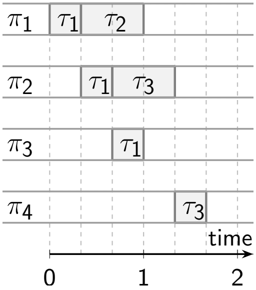
How to get Proper Order:

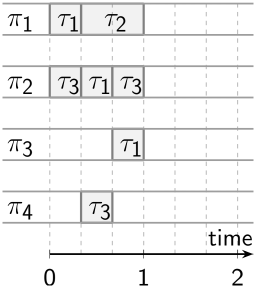
- Run BFS over acyclic Share Graph
- Order cores in the discovery order
- Order tasks for each core in the discovery order

From Flow Solution to Frame: Extended Frame to Frame

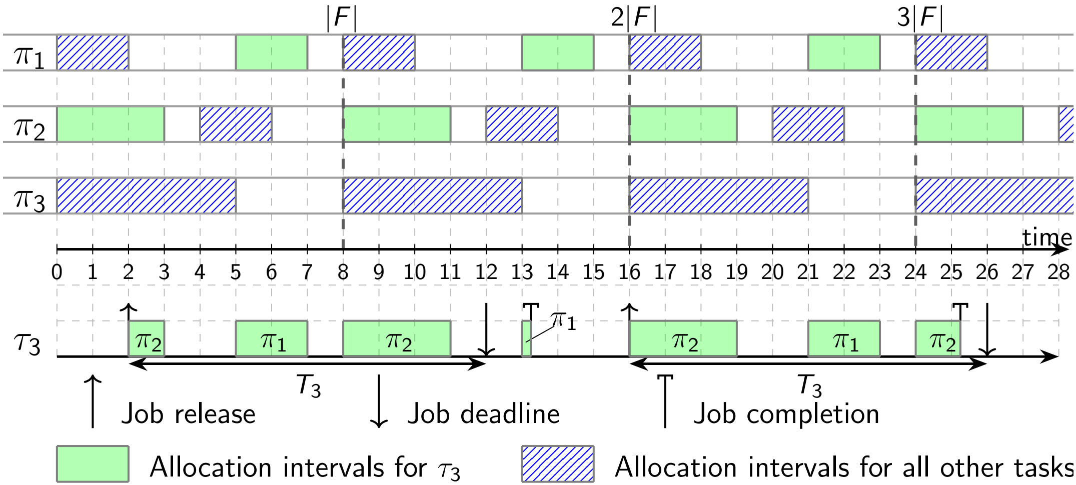
$[t \leftarrow t \mod 1]$ creates a correct schedule for $[0, 1]$ from an extended frame.

	τ_1	τ_2	τ_3
π_1	$1/3$	$2/3$	-
π_2	$1/3$	-	$2/3$
π_3	$1/3$	-	-
π_4	-	-	$1/3$





AM-Red : Sporadic Tasks Scheduling



AM-Red : Complexity

Stage	Complexity	Bound
Max Flow	$\tilde{O}(mn\sqrt{m+n})$	$\Omega(mn)$
Cycles Removal	$O(m^2n^2)$	$\Omega(mn)$
Extended Frame	$O(m+n)$	$\Omega(m+n)$
Frame	$O(m+n)$	$\Omega(m+n)$

At most $m-1$ migrating tasks; at most $2m-2$ migrations per frame.

$\tilde{O}$ ignores logarithmic factors: $\tilde{O}(g(n)) = O(g(n)\log^k g(n))$ for some natural number k.

AM-Red : Complexity

Stage	Complexity	Bound
Max Flow	$\tilde{O}(mn\sqrt{m+n})$	$\Omega(mn)$
Cycles Removal	$O(m^2n^2)$	$\Omega(mn)$
Extended Frame	$O(m+n)$	$\Omega(m+n)$
Frame	$O(m+n)$	$\Omega(m+n)$

At most $m-1$ migrating tasks; at most $2m-2$ migrations per frame.

Max Flow + Cycles Removal = find and modify $f(\tau_i, \pi_j)$.

$\tilde{O}$ ignores logarithmic factors: $\tilde{O}(g(n)) = O(g(n) \log^k g(n))$ for some natural number k.

AM-Red Enhancements: Acyclic Masks

$AG(\tau)$ does not have any cycles $\Rightarrow$ $SG(\tau)$ does not have cycles.

Stage	Complexity
Max Flow	$O(m+n)$
Cycles Removal	0
Extended Frame	$O(m+n)$
Frame	$O(m+n)$

Max Flow: run BFS over $AG(\tau)$ and apply Ford-Fulkerson algorithm.
FF heuristic: augmenting path searches over vertexes in reversed discovery order.

Input data size: $O(m+n)$.

AM-Red Enhancements: Hierarchical Masks

For any two masks α_i, α_j: $\alpha_i \subset \alpha_j$, or $\alpha_i \supset \alpha_j$, or $\alpha_i \cup \alpha_j = \emptyset$.

Importance: follows multiprocessor architecture.

Input data size: $O(mn)$,
special packing is needed to ensure $O(m+n)$.

Hierarchical masks specialty: at most $2m-1$ unique masks and tree masks structure.

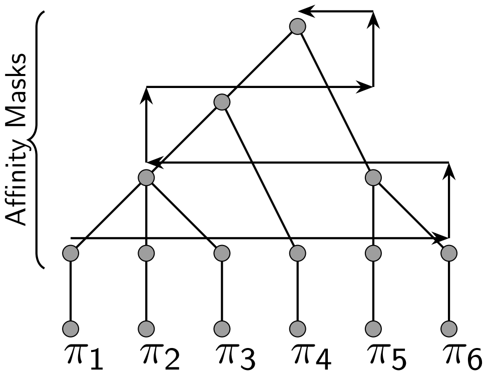
AM-Red Enhancements: Hierarchical Masks

For any two masks α_i, α_j: $\alpha_i \subset \alpha_j$, or $\alpha_i \supset \alpha_j$, or $\alpha_i \cup \alpha_j = \emptyset$.

Importance: follows multiprocessor architecture.

Input data size: $O(mn)$,
special packing is needed to ensure $O(m + n)$.

Hierarchical masks specialty: at most $2m - 1$ unique masks and tree masks structure.



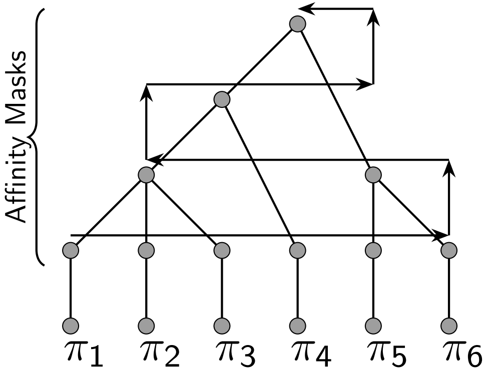
AM-Red Enhancements: Hierarchical Masks

For any two masks α_i, α_j: $\alpha_i \subset \alpha_j$, or $\alpha_i \supset \alpha_j$, or $\alpha_i \cup \alpha_j = \emptyset$.

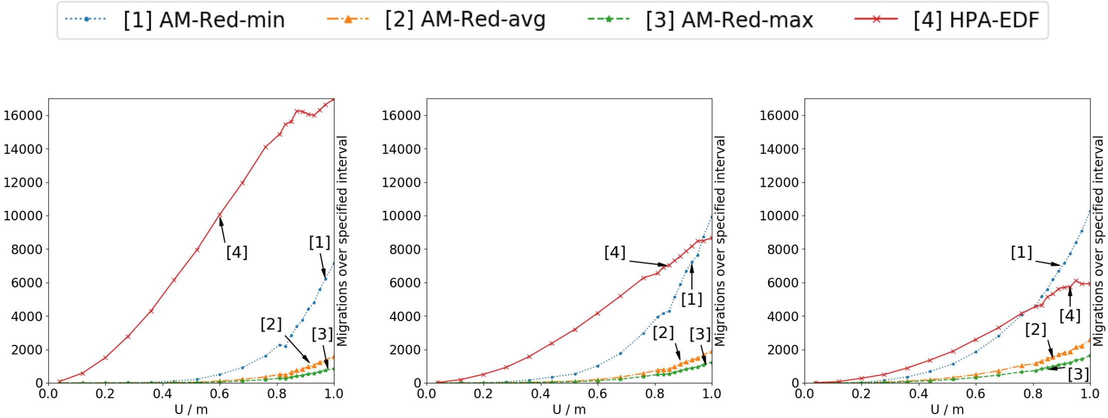
Importance: follows multiprocessor architecture.

Input data size: $O(mn)$,
special packing is needed to ensure $O(m + n)$.

Hierarchical masks specialty: at most $2m - 1$ unique masks and tree masks structure.

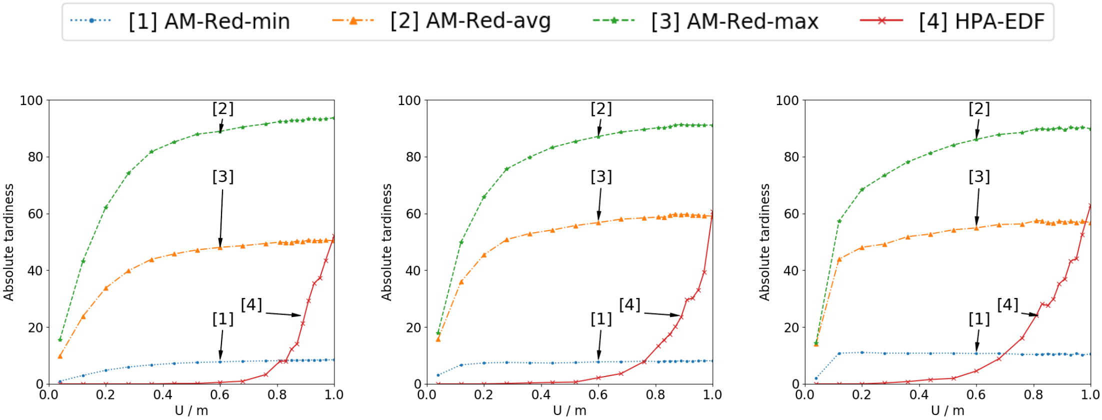


Max Flow: compute shown order and apply Ford-Fulkerson algorithm.
(note that direct flow network $FN(\tau)$ construction requires $\Omega(mn)$, so we avoid it)
FF heuristic: augmenting path searches over tasks vertexes according to masks order.



(a) Light tasks.
(b) Medium tasks.
(c) Heavy tasks.

Figure: Exp. 1 (hierarchical masks): total number of migrations under AM-Red and HPA-EDF (assuming periodic releases), averaged over the generated task sets, as a function of relative system utilization.



(a) Light tasks. (b) Medium tasks. (c) Heavy tasks.

Figure: Exp. 2 (hierarchical masks): maximum tardiness, averaged over the generated task sets, as a function of relative system utilization.

References

- [Baruah 2013] Multiprocessor feasibility analysis of recurrent task systems with specified processor affinities.
- [Brandenburg 2014] Linux's processor affinity API, refined: shifting real-time tasks towards higher schedulability
- [Bonifaci 2016] Multiprocessor real-time scheduling with hierarchical processor affinities.