RTAS 2019

Job-Class-Level Fixed Priority Scheduling of Weakly-Hard Real-Time Systems

Hyunjong Choi, Hyoseung Kim, Qi Zhu[†]

UNIVERSITY OF CALIFORNIA RIVERSIDE

RTEN Real-time Embedded and Networked Systems Laboratory

Northwestern University[†]

Outline

Outline

Weakly-hard real-time systems

- Many practical systems
  - Tolerable to some deadline misses w/o affecting functional correctness

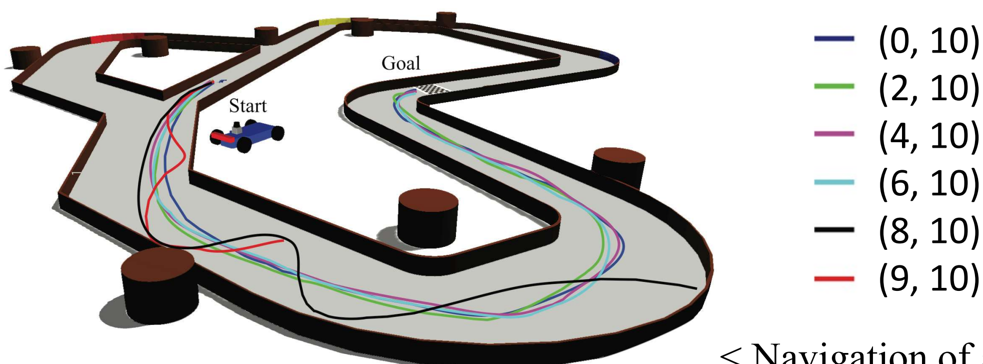
→ **Weakly-hard real-time systems**
to improve resource usage efficiency[‡]

$(\boldsymbol{m}, \boldsymbol{K})$: at most m jobs can miss their deadlines among any K consecutive jobs

‡ G. Bernat, A. Burns, and A. Liamosi, "Weakly hard real-time systems," IEEE transactions on Computers, 2001

Effectiveness of weakly-hard real-time systems

- Navigation of an autonomous vehicle in *Gazebo with ROS*
  - A periodic task: *ControlTask*†
  - Mission: Drive from start to end points
  - Injected deadline misses w.r.t. weakly-hard constraints



< Navigation of an autonomous vehicle – *ControlTask* exp. >

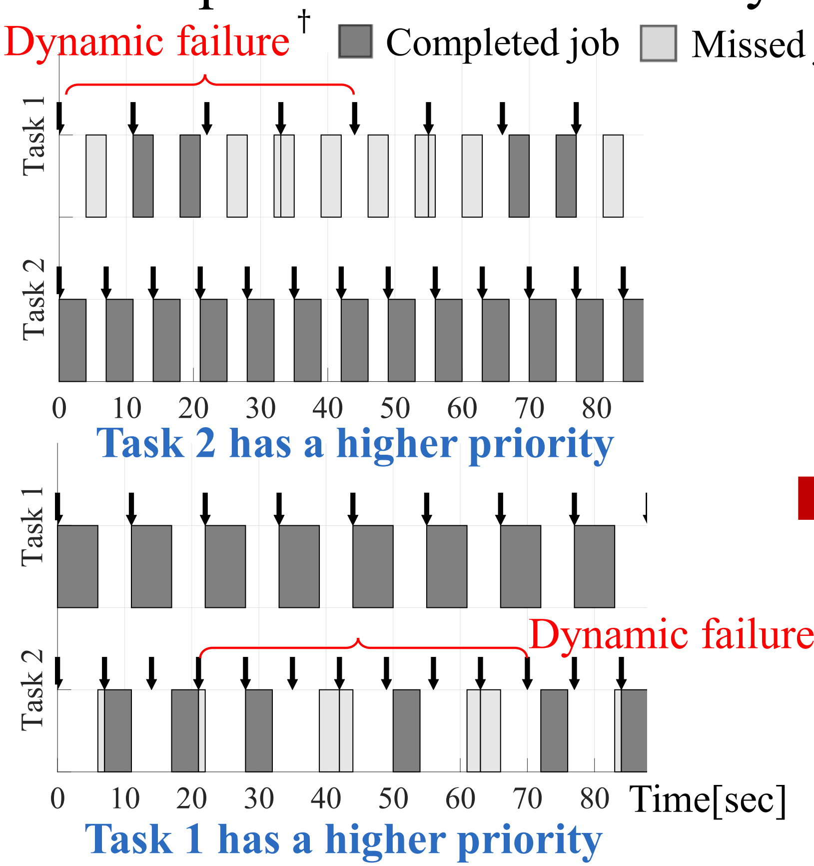
Tasks with bounded deadline misses can produce a functional correctness

Resource can be reserved for the other tasks

† It sends velocity command to robot base(actuator) at the specified rate defined as a control frequency.

Limitation of task-level fixed-priority scheduling

- Simple taskset with weakly-hard constraints



	Specifications
Task 1	$T_1 = 11, C_1 = 6, m_1 = 2, K_1 = 4$
Task 2	$T_2 = 7, C_2 = 4, m_2 = 4, K_2 = 7$

< A taskset example >

No matter which task has a higher priority, NOT schedulable !

New approach

† Task experiences more than *m* deadline misses in a window of *K* jobs.

Contributions

- Main contributions
  - Propose **a new job-class-level fixed-priority scheduler** based on ***meet-oriented classification*** of jobs of tasks
  - Present the **schedulability analysis framework** for our proposed scheduler
  - Generalization of task-level fixed-priority scheduling
  - Outperforms the latest work in terms of **task schedulability**, **analysis running time**
  - **Implement our scheduler in the Linux kernel** running on Raspberry Pi

Outline

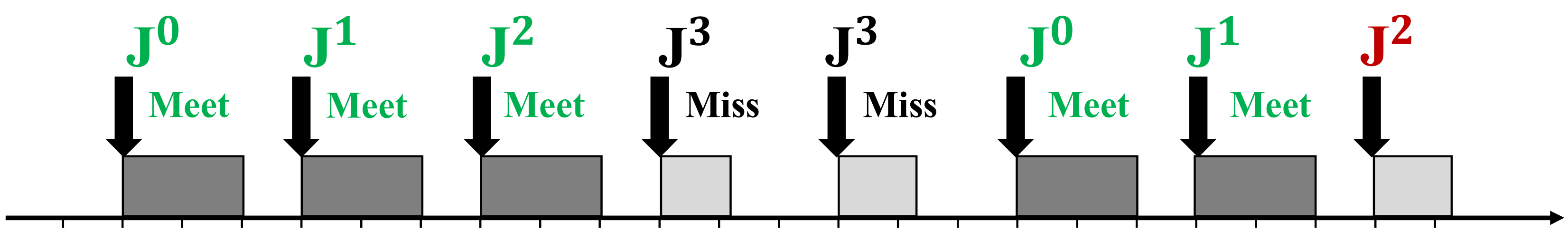
System Model

- Task model
  - $\tau_i \coloneqq (C_i, D_i, T_i, (m_i, K_i))$
    - ✓ C_i: The worse-case execution time
    - ✓ D_i: The relative deadline
    - ✓ T_i: The minimum inter-arrival time
    - ✓ $(m_i,\ K_i)$: The weakly-hard constraints $(m_i < K_i)$. For a hard real-time task, $m_i = 0$ and $K_i = 1$.
- Preemptive scheduling
- Uniprocessor system

Job-Class-Level Fixed-Priority Scheduling

- Job classification
  - Assign different priorities to individual job-classes

Meet-oriented : the number of prior deadlines consecutively met



- For instance, $(m, K) = (2, 4)$ can have job classes: J^0, J^1, and J^2
- Priority of a job-class decrease monotonically

Bounding consecutive deadline misses

- Miss threshold w_i
  - Limit the distance from the current job to the previous deadline-met jobs to bound the number of consecutive deadline misses

$$w_i = \max\left(\left\lfloor \frac{K_i}{K_i - m_i} \right\rfloor - 1, 1\right)$$

  - Ensure enough number of jobs running with the highest priority job-class
  - For instance, $(m, K) = (5, 7)$ where $w_i = 2$ allows 2 consecutive deadline misses

Priority assignment

- A heuristic priority assignment
  - An **extension of the deadline monotonic (DM)** priority assignment

Lemma 3.

Subsumes the task-level DM priority assignment

  - Rule.
    - ✓ Assign higher priority to a job-class with a smaller index
    - ✓ For job-classes with the same index,
      - Higher priority to shorter deadline (q = 0)
      - Higher priority to shorter miss threshold with deadlines for tie-breaking (q > 0)

Algorithm 1 Job-class priority assignment

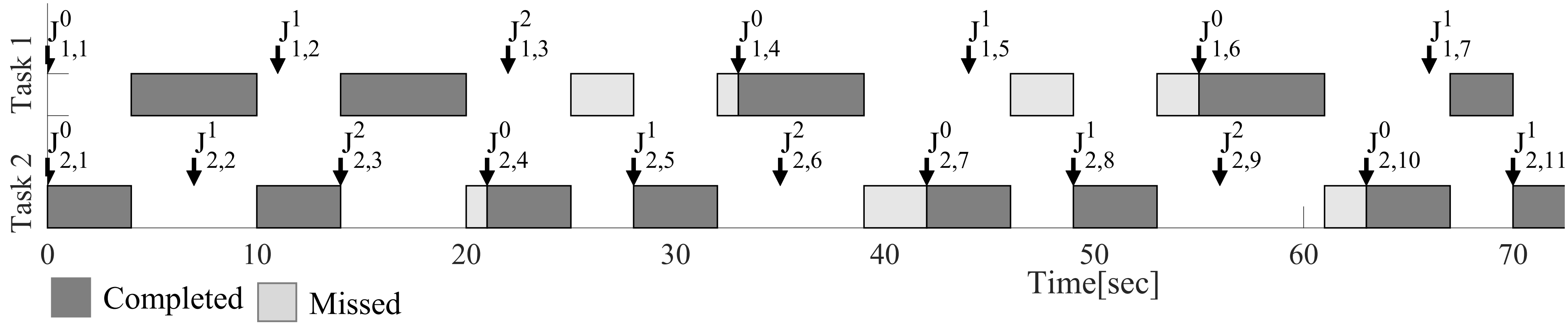
```
Input: Γ: Taskset
N ← |Γ|
Sort τ_i in Γ in ascending order of deadline
for all τ_i ∈ Γ do
    l_i ← K_i − m_i + 1              ▷ l_i: number of job-classes for τ_i
end for
prio ← Σ_{τ_i∈Γ} l_i                  ▷ Priority to be assigned next
if Γ is schedulable by DM then
    for all τ_i ∈ Γ do
        ▷ Assign the same priority to all job-classes of τ_i
        for all q ← 0 to l_i − 1 do
            π_i^q ← prio
        end for
        prio ← prio − 1
    end for
else
    L ← max_{τ_i∈Γ} l_i
    for q ← 0 to L − 1 do
        if q > 0 then
            Sort τ_i ∈ Γ in ascending order of w_i and deadline
        end if
        for all τ_i ∈ Γ do
            if q < l_i then            ▷ Check if q is a valid index
                π_i^q ← prio
                prio ← prio − 1
            end if
        end for
    end for
end if
```

An example of job-class-level scheduling

- With the same taskset at page 6.

	Specifications
Task 1	$T_1 = 11, C_1 = 6, m_1 = 2, K_1 = 4$
Task 2	$T_2 = 7, C_2 = 4, m_2 = 4, K_2 = 7$

< A taskset example >

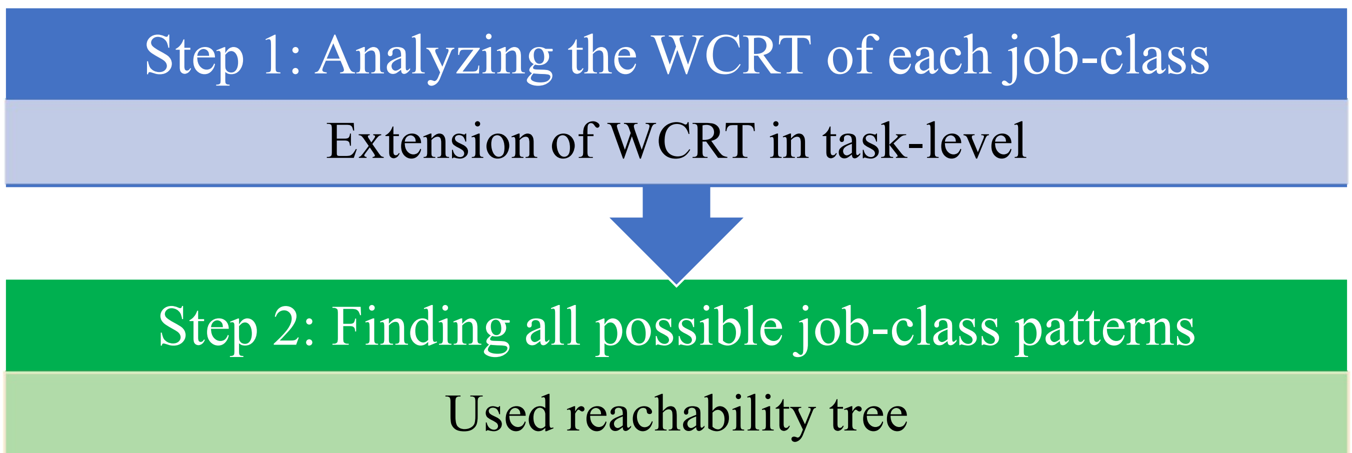


Schedulable !!

Outline

Schedulability Analysis

- The schedulability analysis of tasks with weakly-hard constraints under job-class-level scheduling decompose



< Schedulability analysis process of job-class-level scheduler >

Worse-case response time of job-classes

- Worse-case response time of J_i^q is bounded by the recurrence:

$$R_i^{q,n+1} \leftarrow C_i + \sum_{\tau_k \in \Gamma - \tau_i} W_i^q(R_i^{q,n}, \tau_k)$$

✓ W_i^q is an upper-bound of interference imposed on J_i^q

$$W_i^q(t, \tau_k) = \min\left(\sum_{\forall p : \pi_i^q < \pi_k^p} \left\lceil \frac{t + J_k}{\eta(J_k^p)} \right\rceil \times C_k , \left\lceil \frac{t + J_k}{T_k} \right\rceil \cdot C_k \right)$$

✓ Each job-class has a *different minimum job-class inter-arrival time*, $\eta(J_k^p)$

Lemma 8.

Generalization of the task-level iterative response time test for hard real-time tasks.

† M. Josephand P. Pandya, "Finding response times in a real-time system," The Computer Journal, 1986.

Schedulability check

- Schedulability test of a task with $m_i/K_i \geq 0.5$

Lemma 10.

*A task τ_i is **always schedulable** if the **ratio of m_i/K_i is greater than or equal to 0.5** and it **satisfies the prerequisite** given by Lemma 9.*

Step 1: Show at least 1 deadline met in K_i window by using a **necessary condition**

$$(w_i + 1) \cdot \alpha \leq K_i \qquad \mathrm{WCRT}(J_i^0) \leq D_i$$

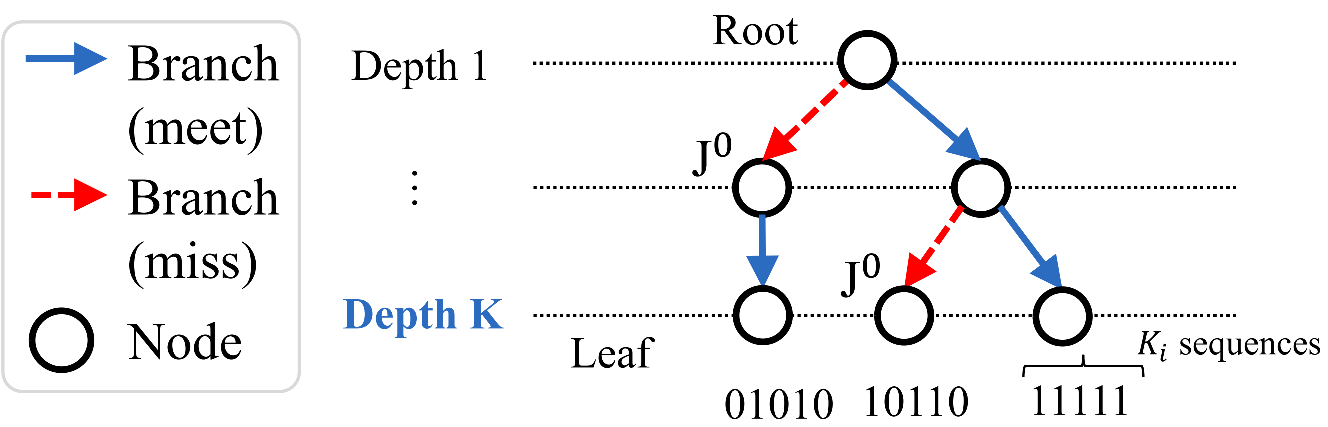
Step 2: Show that the number of deadline met satisfies the constraint

$$\frac{1}{w_i+1} \geq \frac{K_i - m_i}{K_i} \quad \Rightarrow \quad \left\lfloor \frac{K_i}{K_i - m_i} \right\rfloor \leq \frac{K_i}{K_i - m_i}$$

Always true as $m_i \leq K_i - 1$

Reachability tree

- For tasks with $m_i/K_i < 0.5$, find all possible job-class patterns for K_i job executions using ***reachability tree***



< Reachability tree >

Lemma 13.

The reachability trees of a task τ_i ***represent all possible job-class patterns*** *that the task can experience at its runtime for K_i* execution window

Outline

Implementation cost

- Measure runtime overhead of the proposed scheduler implementation
- Experimental setup
  - Linux kernel v4.9.80 running on Raspberry Pi 3
  - ARM Cortex-A53 @ clock frequency of 1.2 GHz
  - Run 5 tasks with period of 20ms to 40ms for 10 minutes (118,569 jobs)

Type		Mean	Max	Min	99%th
Updating μ-pattern†		0.3002	1.1460	0.1040	0.6250
Updating job-class index		1.5035	11.8750	0.5210	2.5000
Changing task priority		4.7633	28.9580	3.0210	11.3020
Rollback	Checkpointing	1.9413	9.3230	1.2500	3.2290
	Recovery	6.1257	24.8430	0.4680	8.3146

< Runtime overhead [μs] >

† Represents a sequence of deadline met and missed jobs of a task, (G. Bernat, A. Burns, and A. Liamosi. "Weakly hard real-time systems", 2001)

Schedulability experiments

- The evaluation is conducted in two ways:
  - **Comparison** with other weakly-hard scheduling schemes (WSA[†], RTO-RM[*])
    - ✓ WSA: delayed completion for deadline-missed jobs
    - ✓ RTO-RM: job abort for deadline-missed jobs
  - **Exploration** of the proposed scheduler under diverse experimental conditions
  - Performance metric : *percentage of schedulable taskset*, *analysis running time*

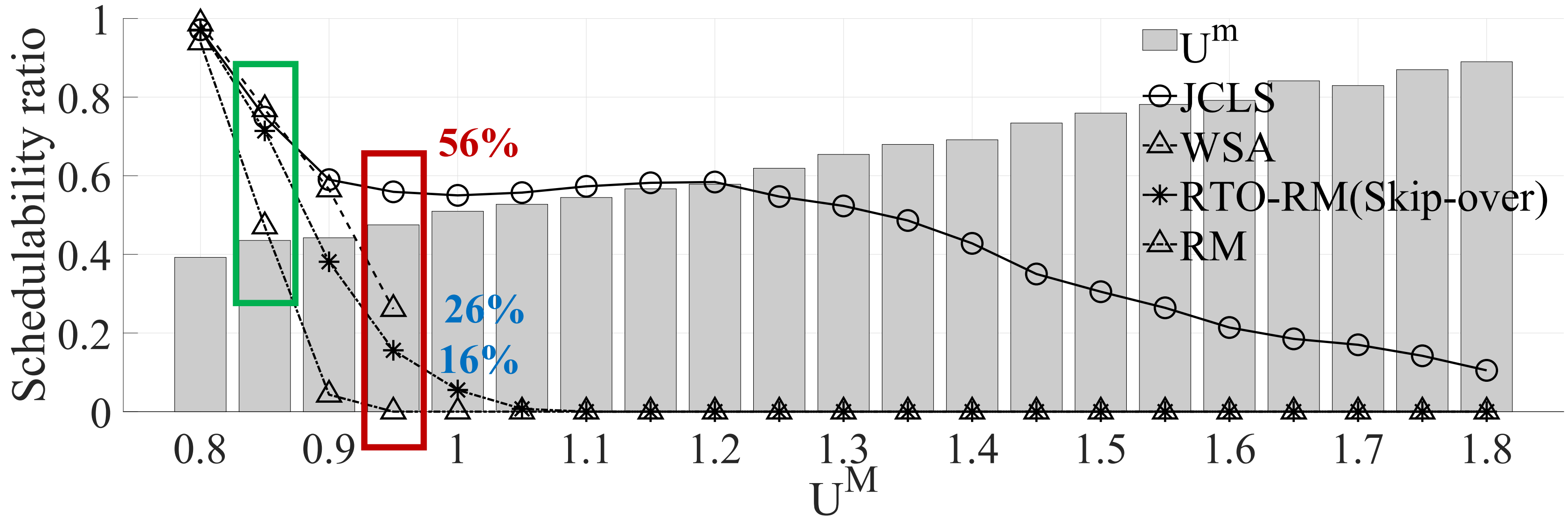
- Taskset generation

	Number of tasksets	Task utilization (UUniFast algorithm[#])	Task period [ms]	K range
Value	1,000	[0.8, 1.8]	[10, 1000]	{5, 10, 15}

[†] Y. Sun and M. D. Natale, "Weakly hard schedulability analysis for fixed priority scheduling of periodic real-time tasks," *TECS*, 2017
[*] G. Koren and D. Shasha. "Skip-over: Algorithms and complexity for overloaded systems that allow skips", RTSS, 1995
[#] E.Biniand G.C.Buttazzo. "Measuring the performance of schedulability tests", Real-Time Systems, 2005

Taskset schedulability

- Comparison of schedulability ratio with other schemes
  - 1,000 tasksets with 20 tasks
  - $K_i = 10, m_i = [1, 9]$, common (m, K) for a taskset



➡ **JCLS better utilizes CPU resource when there are overloaded weakly-hard tasksets**

Analysis running time

- Time to determine the schedulability of a given taskset
  - By the number of tasks in a taskset (10, 30, and 50 tasks)
  - 1,000 tasksets, $K_i = 10, m_i = [1, 9]$
  - JCLS (on Raspberry Pi 3), WSA (on Intel Core-i7 for CPLEX Optimizer)

Number of tasks	Approach	Mean	Max
10	JCLS	0.0010	0.0046
	WSA	0.2739	114.2892
30	JCLS	0.0112	0.0432
	WSA	25.7284	1800.5996
50	JCLS	0.0331	0.1463
	WSA	78.5982	3002.5189

< Analysis running time [sec] >

➡ **The analysis time of JCLS is shorter than that of WSA**
More applicable to runtime admission control

Conclusion & Future work

- Conclusion
  - **New job-class-level fixed-priority scheduling** and **analysis** for weakly-hard real-time systems
  - Proposed scheduler **outperforms prior work** with respect to taskset schedulability and analytical complexity
  - Proposed approach is effective in overloaded situations (e.g., maximum utilization is higher than 1)

- Future work
  - Address the pessimism of our schedulability analysis when the ratio of m_i/K_i is less than 0.5

Thank you

Job-Class-Level Fixed Priority Scheduling of Weakly-Hard Real-Time Systems

Hyunjong Choi, Hyoseung Kim, Qi Zhu

Q & A