

Deadline-Aware Scheduling of Mixed-Criticality Tasks

Maxime Gonthier

Kyle Chard

Ian Foster

Loris Marchal

Frédéric Vivien



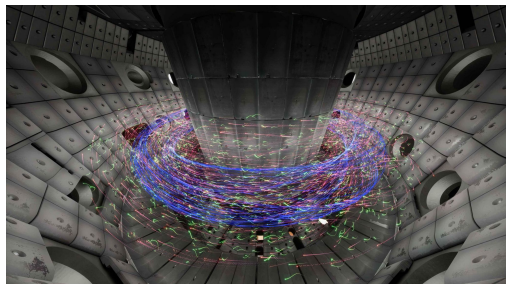
Table of contents

1. **Which problem are we trying to solve?**
2. Theoretical lower bound
3. Approximation & proposed heuristic
4. State-of-the-art competitors
5. Evaluation

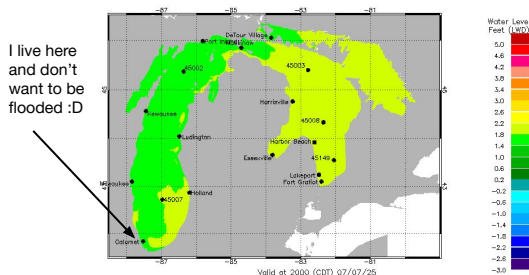


Clusters & cloud support the execution of workloads that differ in criticality (= time sensitivity)

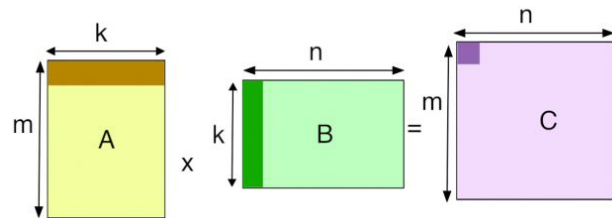
Critical



Plasma simulation
(IonOrb) from DIII-D
National Fusion Facility:
must be computed within
15 minutes

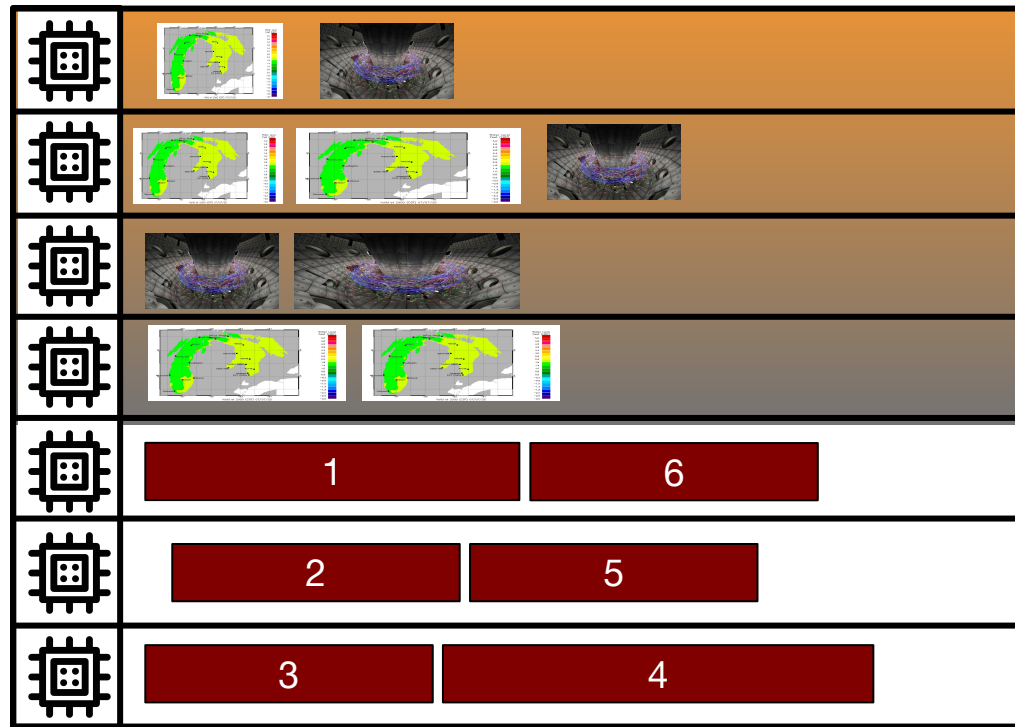


Lake Michigan water level
forecast: must be
computed within a few
hours



GEMM for a performance
benchmark: no deadline

Simplest approach: static provisioning

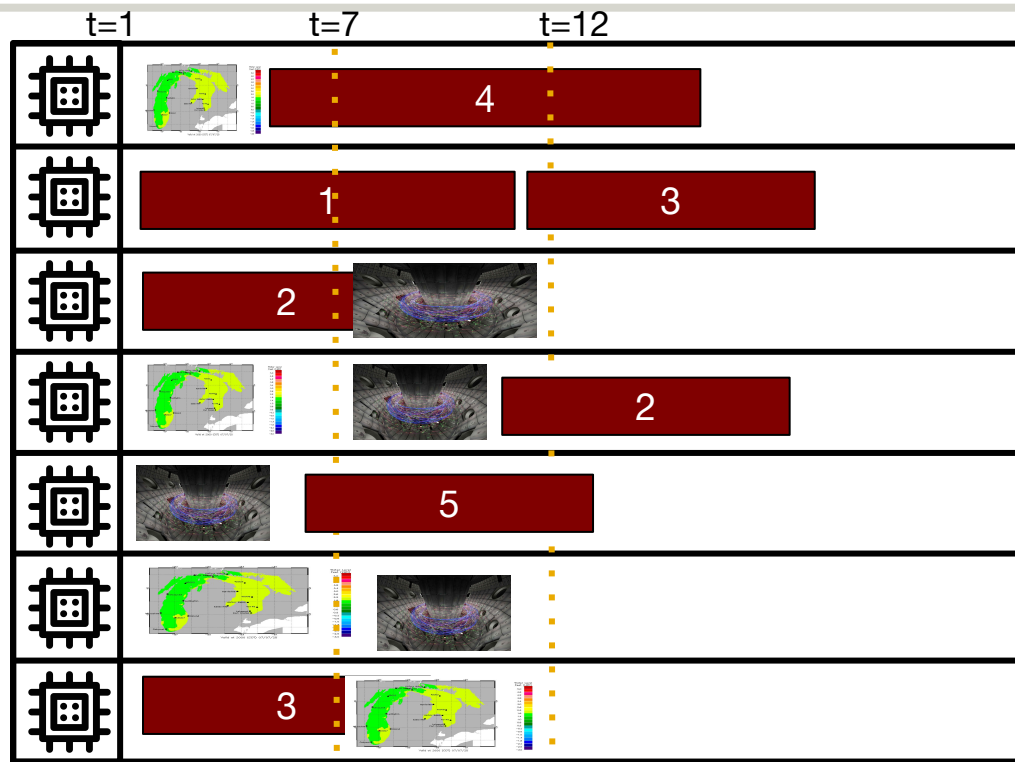


- Statically **reserve** a portion of computing resources for critical tasks

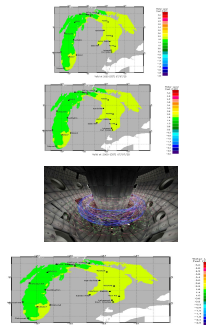
- Often over provision for safety

→ **performance degradation for low-criticality tasks**

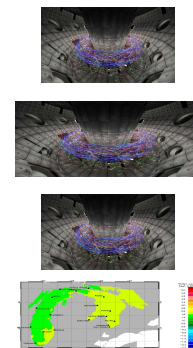
Hybrid approach: shared resources with preemption



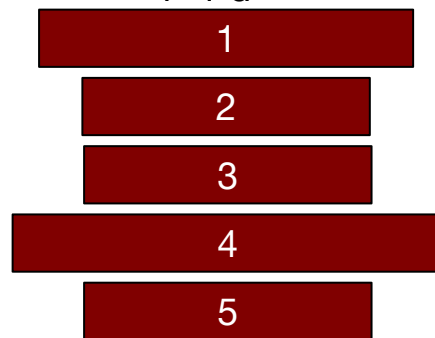
$r=1$ $d=12$



$r=7$ $d=12$

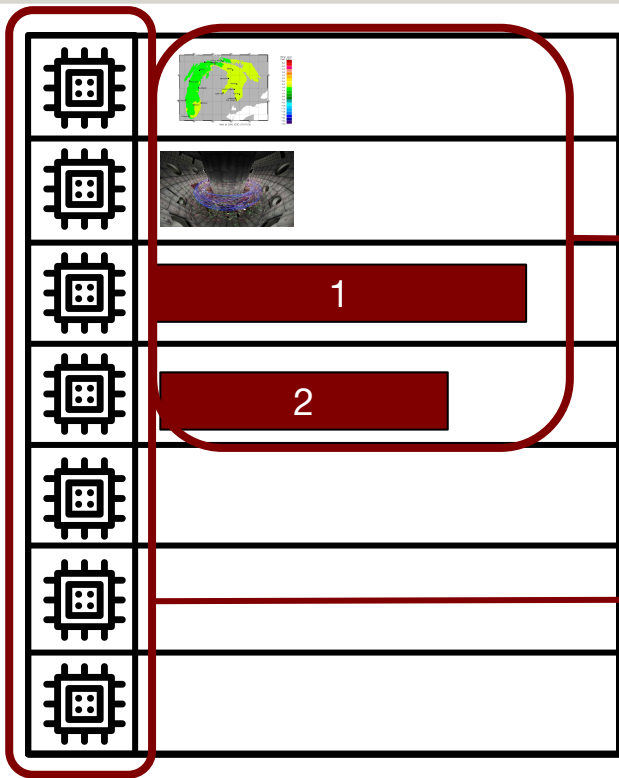


$r=1$ $d=\emptyset$



→ **wasted resources**

Problem: How can we use a shared pool of resources for mixed-criticality tasks without preemption?



Independent, exclusive and single node tasks:

- r : release time (both offline and online)
- p : processing time
- d : deadline or \emptyset

No preemption, checkpointing or migration

Homogeneous processors

Goal:

Maximize # of critical task completed in time

&

minimize F_{max} of non-critical tasks

Table of contents

1. Which problem are we trying to solve?
- 2. Theoretical lower bound**
3. Approximation & proposed heuristic
4. State-of-the-art competitors
5. Evaluation



Theoretical lower bound through binary search & linear programming

- Lower bound exist for $1|online-r;restarts|\sum F$ and $P|online-r;preemption|\sum 1-U$
- Not for mixed-criticality

- → allow **preemption** and **migration**:¹
 - Time sharing
 - **Fix a target flow** F_{target} → **gives a deadline to all non-critical tasks** → $r + F_{target}$
 - Create sorted list of intervals using release times and deadlines
- For each consecutive pair in the list, $x_{i,k}$ is the time assigned to T_i in interval $[t_k, t_{k+1}]$ on any processor
- **Build a linear program** with $x_{i,k}$. If solution, resulting time allocations is a feasible schedule with F_{target} → **binary search** on F_{target}

¹Similar to “Minimizing the Stretch When Scheduling Flows of Divisible Requests” Legrand et al. Journal of Scheduling (2008)

Table of contents

1. Which problem are we trying to solve?
2. Theoretical lower bound
- 3. Approximation & proposed heuristic**
4. State-of-the-art competitors
5. Evaluation



A $\frac{1}{2}$ -approximation from the Group Interval Scheduling Maximization Problem (GISMP)

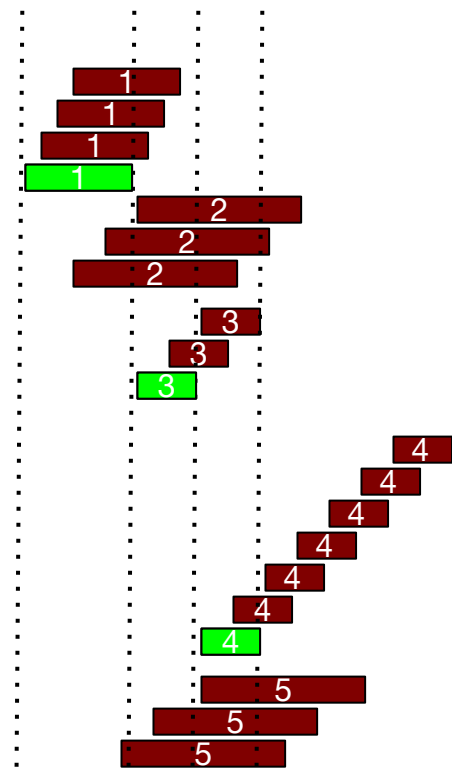
GISMP: finding largest set of non-overlapping intervals

Interval: task's possible time frames in which it can be executed

Goal: execute as many different tasks as possible

A $\frac{1}{2}$ -approximation exist for 1 processor: it always schedules at least half as many tasks as an optimal algorithm:

1. Allocates tasks 1 by 1 so that the next selected interval is the one with the earliest finish time
2. Remove intervals of tasks intersecting with selected interval



A $\frac{1}{2}$ -approximation from the Group Interval Scheduling Maximization Problem (GISMP)

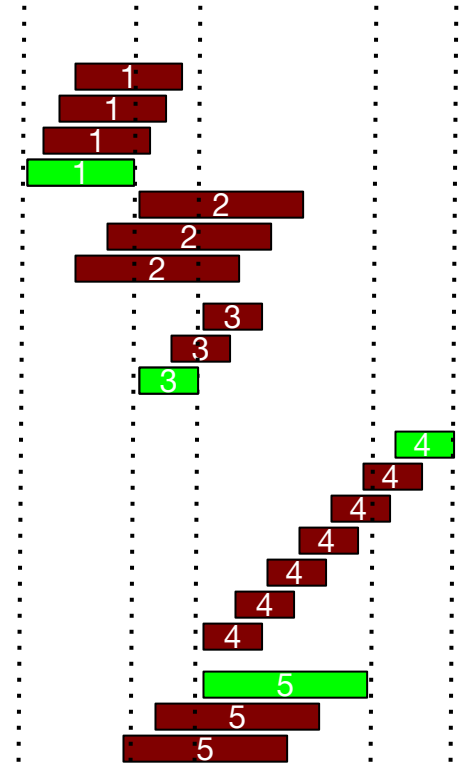
GISMP: finding largest set of non-overlapping intervals

Interval: task's possible time frames in which it can be executed

Goal: execute as many different tasks as possible

A $\frac{1}{2}$ -approximation exist for 1 processor: it always schedules at least half as many tasks as an optimal algorithm:

1. Allocates tasks 1 by 1 so that the next selected interval is the one with the earliest finish time
2. Remove intervals of tasks intersecting with selected interval



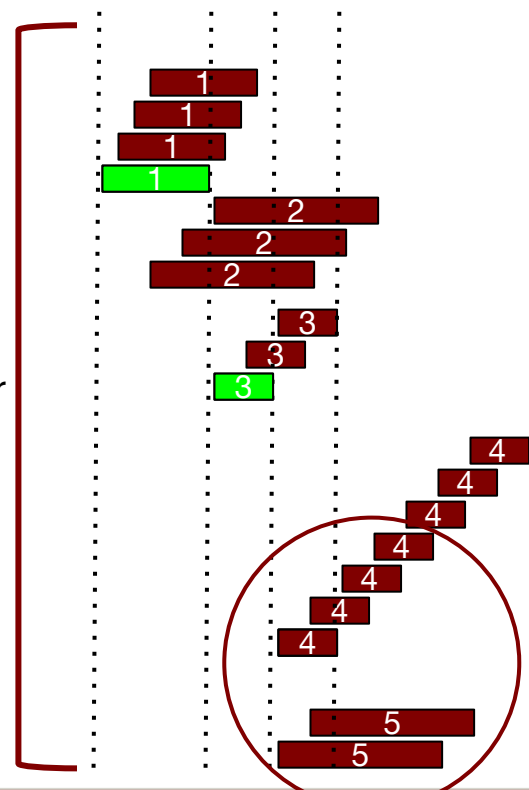
General case approximation: *Greedy*

We **adapt the approximation to ℓ processors** and prove (read the paper for more details :D) that there is a polynomial $\frac{1}{2}$ -approximation algorithm:

1. Consider processors 1 by 1
2. Allocates tasks 1 by 1 so that the next selected interval is the one with the earliest finish time
3. Remove intervals of tasks intersecting with selected interval
4. Once no more tasks can be scheduled on a processor, continue with the next processor

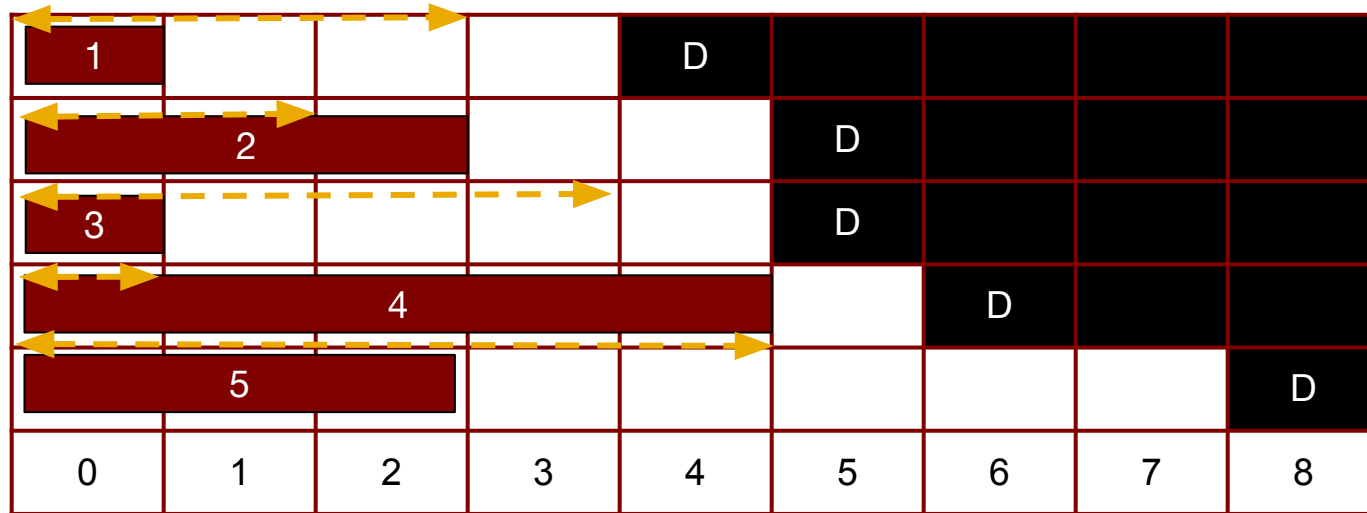
Issue: Offline algorithms can defer urgent tasks: a newly submitted short-deadline task with late finish times may be scheduled last

Only for 1 processor



Developed into a slack-focused heuristic: *Greedy-Slack*

Slack of T1

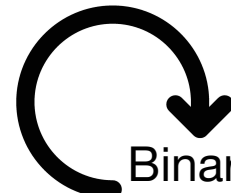


- Sort tasks by **deadline slack**:
“amount of time a task can remain in queue before it must be immediately processed in order to meet its deadline”

EDF order: 1,2,3,4,5 / Greedy-Slack order: 4,2,1,3,5

Developed into a slack-focused heuristic: *Greedy-Slack*

- 1: Assign a deadline to non-critical tasks as $r_j + F_{target}$
- 2: Sort \mathbb{T} by decreasing value of deadline slack
- 3: **for each** $P_i \in \mathbb{P}$ **do**
- 4: $t \leftarrow \text{current_time}$
- 5: **for each** $T_j \in \mathbb{T}$ **do**
- 6: $EST \leftarrow \max(t, r_j)$
- 7: **if** $EST + p_j \leq d_j$ **then**
- 8: Schedule T_j on P_i
- 9: $t \leftarrow EST + p_j$



Binary
search on
 F_{target}

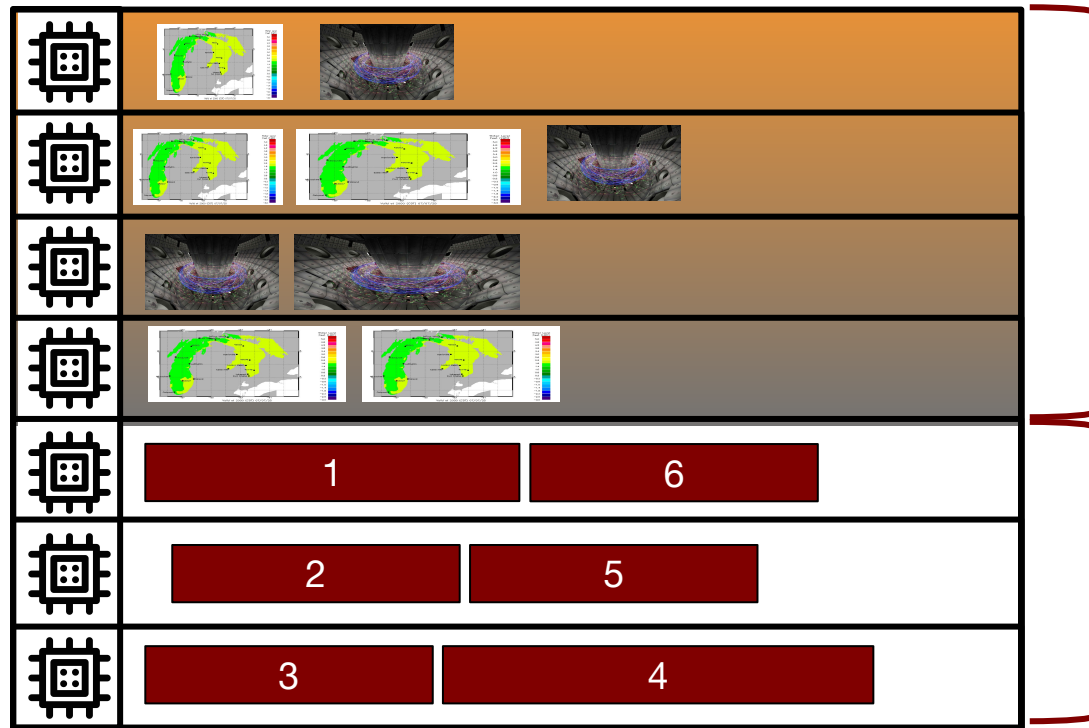
- Schedule tasks by order of **deadline slack**
- Keep intuition from approximation:
schedule processor by processor &

Table of contents

1. Which problem are we trying to solve?
2. Theoretical lower bound
3. Approximation & proposed heuristic
4. **State-of-the-art competitors**
5. Evaluation




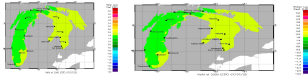

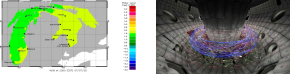





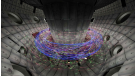


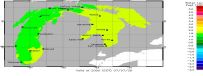


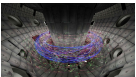


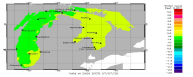

State of the art competitor 1: static provisioning



Min number of nodes required
Provisioned offline w/ binary search

FIFO optimal to minimize
 F_{max} with 1 processor
→ use FIFO to schedule
non-critical task

State of the art competitor 2: combination of optimal algorithms for subproblems

- EDF optimal in the sense that “if a valid schedule exists, it will be found in an online single processor setting” & “if processors are $(2-1)/m$ faster, it will find a solution in a preemptive case with m processors”
→ EDF is a reasonable approach
- FIFO optimal to minimize F_{max} with 1 processor

→ first use EDF for critical tasks then use FIFO for non-critical tasks

Table of contents

1. Which problem are we trying to solve?
2. Theoretical lower bound
3. Approximation & proposed heuristic
4. State-of-the-art competitors
5. **Evaluation**

Goal:

Complete **ALL**
critical task in time

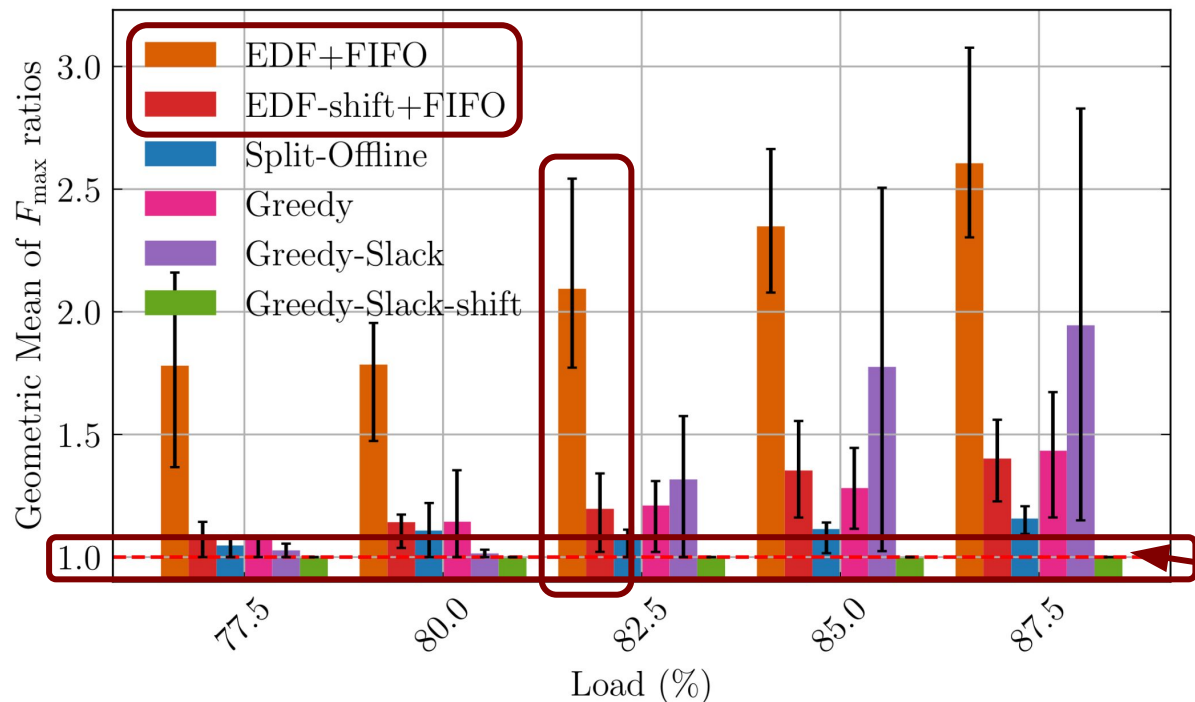
&

minimize F_{max} of
non-critical tasks



Geometric mean of ratio between F_{max} and lower bound - offline

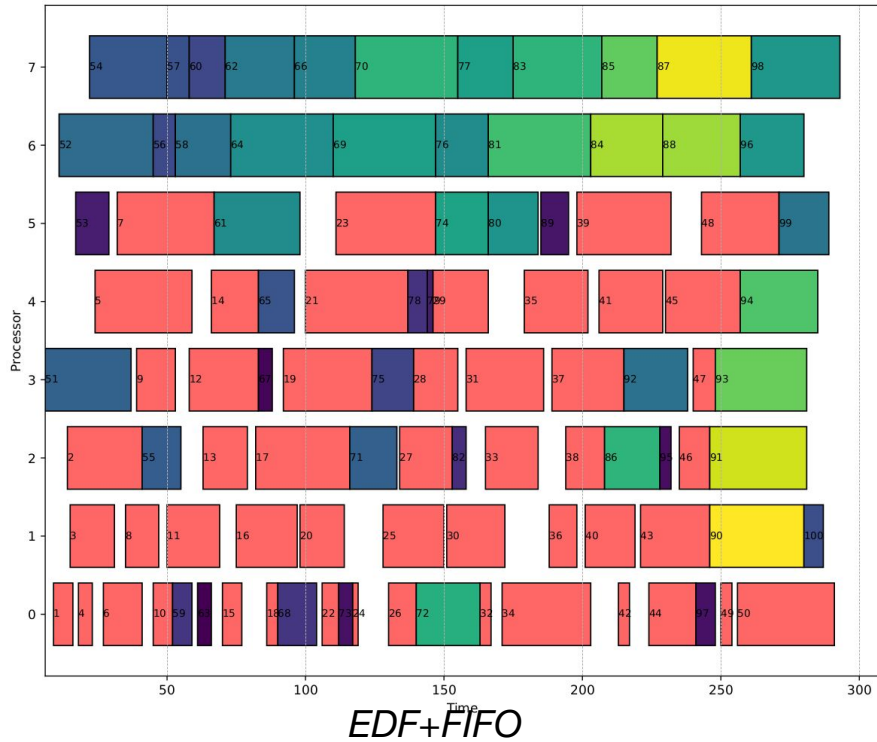
Lower
better



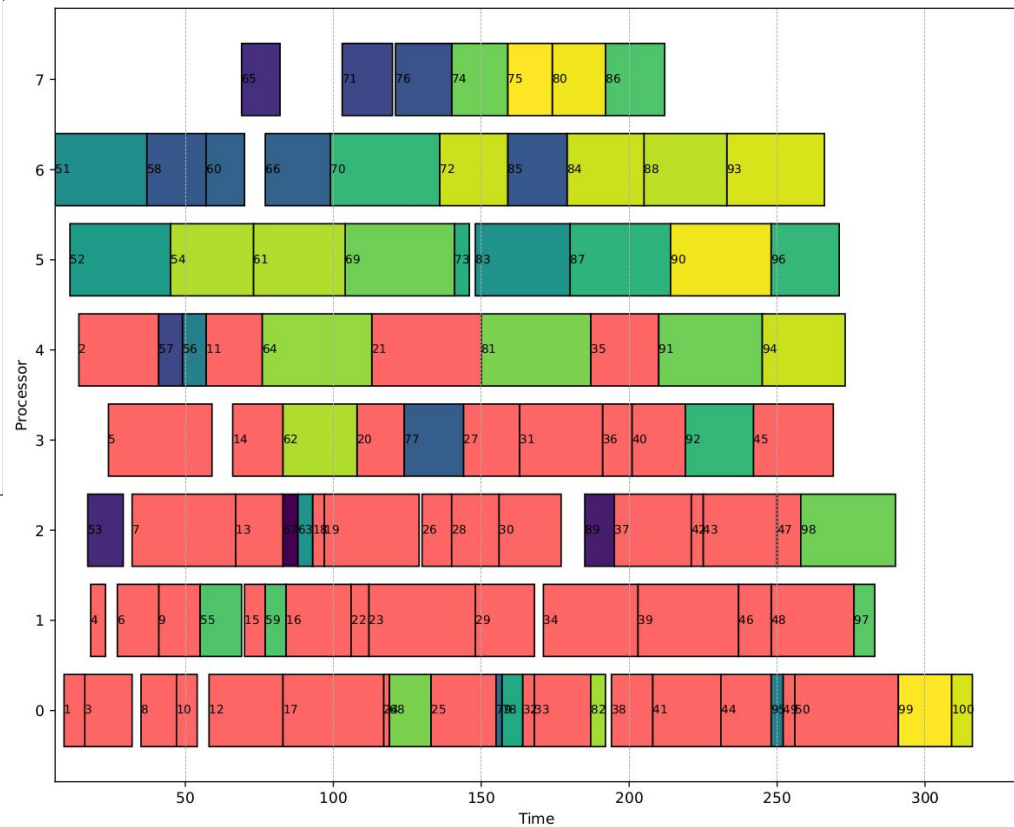
- 1 \rightarrow lower bound
- 30 random instances

- *shift* \rightarrow temporal shifting, i.e. “a critical task may be postponed as long as its deadline is met”

- *Greedy* performs worse than *Split-Offline*
- *Greedy-Slack* with temporal shifting matches lower bound

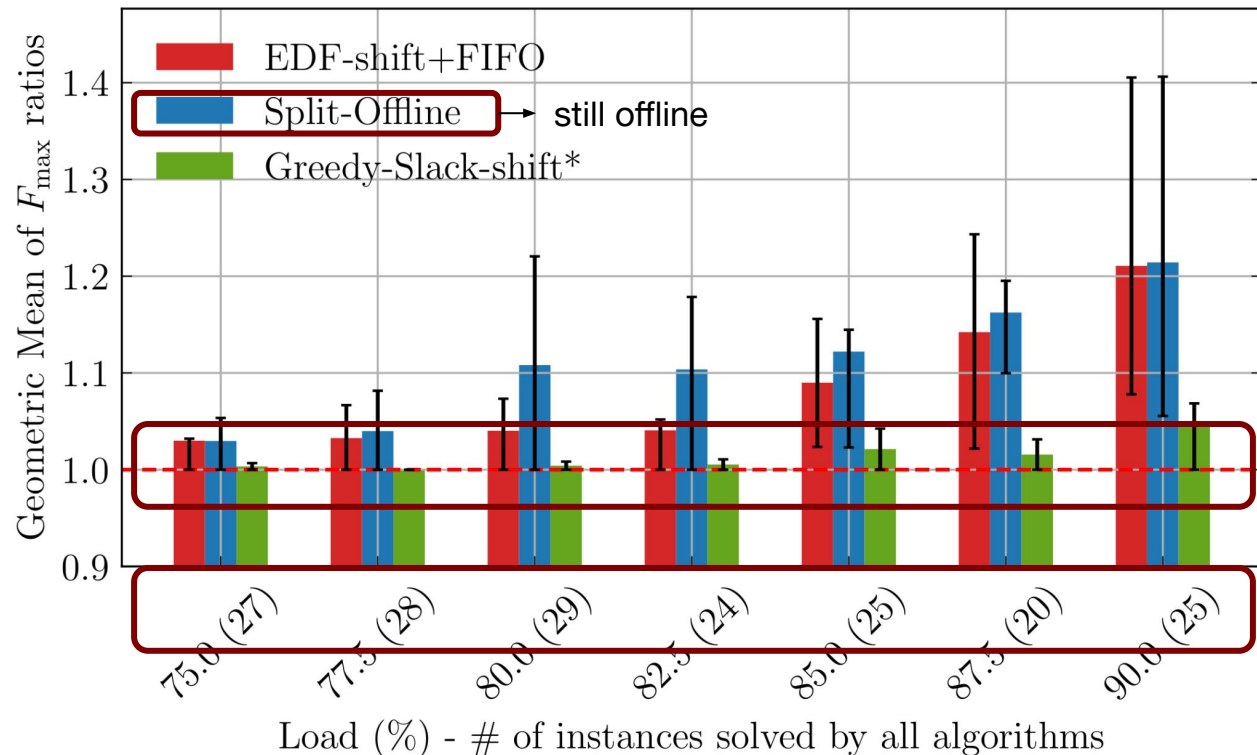


Greedy-Slack



Geometric mean of ratio between F_{max} and lower bound - online

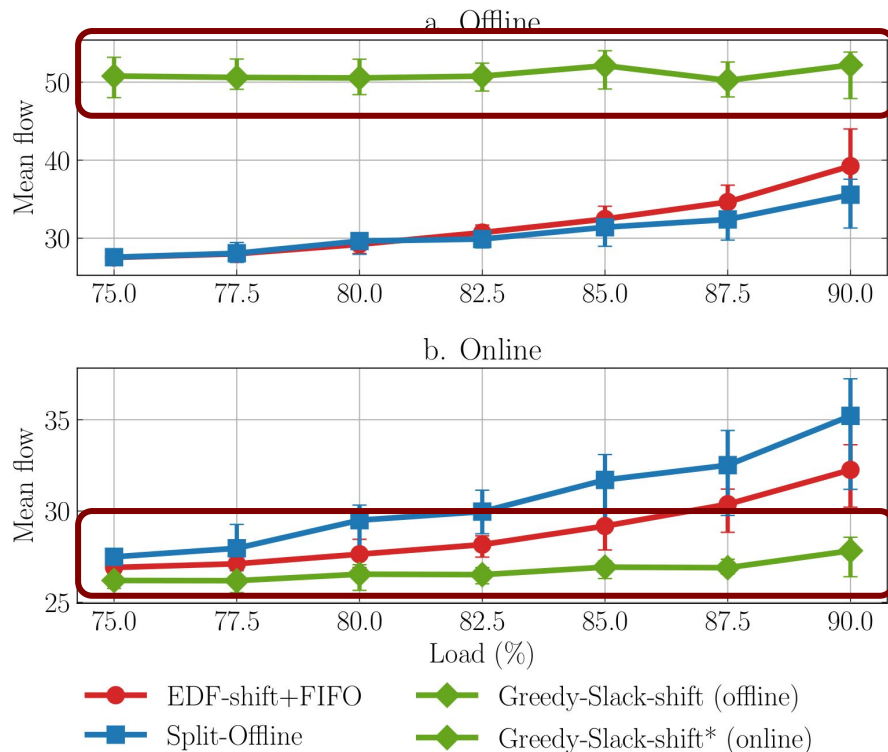
Lower
better



- Some heuristics unable to find solution for some instances → only compare results in instances where all heuristics reach a valid schedule (= all deadline satisfied)
- Greedy-Slack* still close to lower bound
- Scheduling overhead of *Greedy-Slack*: 36ms per task

Absolute values of mean flow w/ 8 processors

Lower
better



- Represent average delay
- Not our main goal, service quality metric
- **Offline:** many tasks available at once
→ Greedy-slack will slow down all tasks to minimize F_{max}
- **Online:** limited amount of tasks at once
→ Greedy-slack schedules non-critical tasks earlier

Proportion of successful instances (out of 30) under various workload constraints - online



EDF-SHIFT+FIFO

Deadline slack	0.0	0	0	0	0	0	0
2.0	100	73	70	63	47	43	27
4.0	100	90	93	87	93	87	83
6.0	100	97	100	93	97	100	100
8.0	100	100	100	97	100	100	100
10.0	100	100	100	97	100	100	100
Load (%)	75.0	77.5	80.0	82.5	85.0	87.5	90.0

More valid schedule under constrained workloads at the cost of higher F_{max}

SPLIT-OFFLINE

Deadline slack	0.0	97	80	73	67	57	33	27
2.0	100	100	100	100	100	100	100	100
4.0	100	100	100	100	100	100	100	100
6.0	100	100	100	100	100	100	100	100
8.0	100	100	100	100	100	100	100	100
10.0	100	100	100	100	100	100	100	100
Load (%)	75.0	77.5	80.0	82.5	85.0	87.5	90.0	

Offline provisioning → will get the most valid schedule

GREEDY-SLACK-SHIFT*

Deadline slack	0.0	0	0	0	0	0	0
2.0	70	70	50	37	37	17	13
4.0	90	97	93	83	87	80	60
6.0	100	100	97	97	93	93	83
8.0	100	100	100	93	100	100	100
10.0	100	100	100	100	100	100	97
Load (%)	75.0	77.5	80.0	82.5	85.0	87.5	90.0

Conclusion on mixed-criticality scheduling of non-preemptive tasks on homogeneous processors

- Derived a $\frac{1}{2}$ -approximation algorithm and a lower bound
- Designed a slack-focused heuristic from the approximation
- **Reduces** online **max flow** by:
 - 14% vs. static provisioning
 - 13% vs. EDF+FIFO
- **Near theoretical lower bound** (offline & online)
- Similar on real-world traces

Next:

- Task failures
- Parallel tasks

