Meeting Job-Level Dependencies by Task Merging

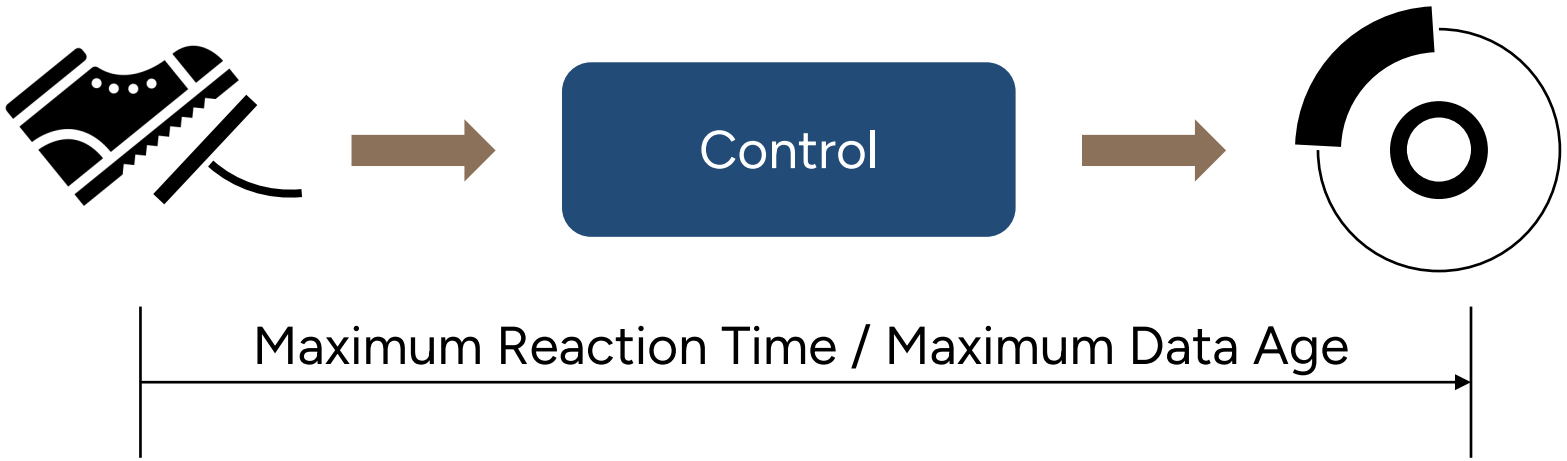
29th Asia and South Pacific Design Automation Conference (ASP-DAC) 2024

Matthias Becker

January 25, 2024

Motivating Example: Cause-Effect Chains

- Functionality of applications often emerges from the interplay of different system components that process data
- Often from sensor to actuation
- One such chain is called cause-effect chain
- Timing constraints are specified for cause-effect chains in the form of end-to-end delay constraints



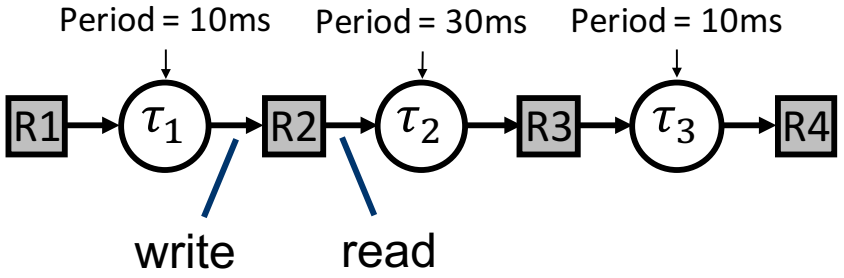
Motivating Example: Cause-Effect Chains

No signaling between tasks

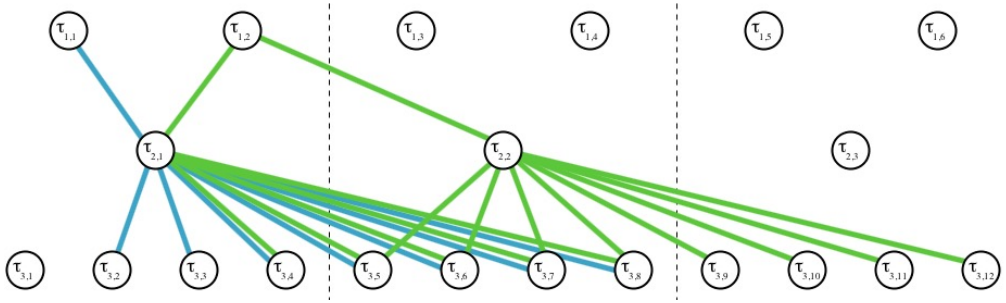
A value can be overwritten before it is read or read >1

- A cause-effect chain describes a chain of semantically related tasks
  - Can have different periods
  - Describes no precedence constraints

- Communication between tasks over shared memory
  - A sender task writes to the shared memory
  - A receiver task reads from the shared memory



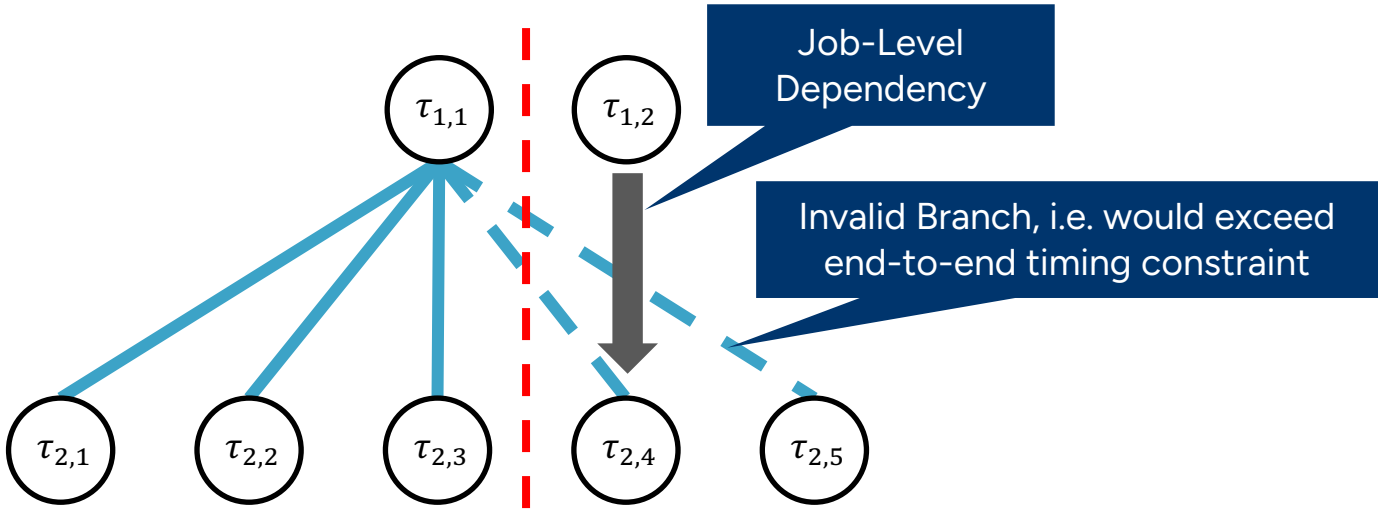
- Basen on task periods and WCET values we can determine all possible data propagation paths (independent of scheduling)
  - Some paths would lead to missed end-to-end deadlines.



How to make sure such paths can't be used at runtime?

Motivating Example: Cause-Effect Chains

- Order selected jobs in such a way that any schedule that respects this order satisfies the timing constraints.



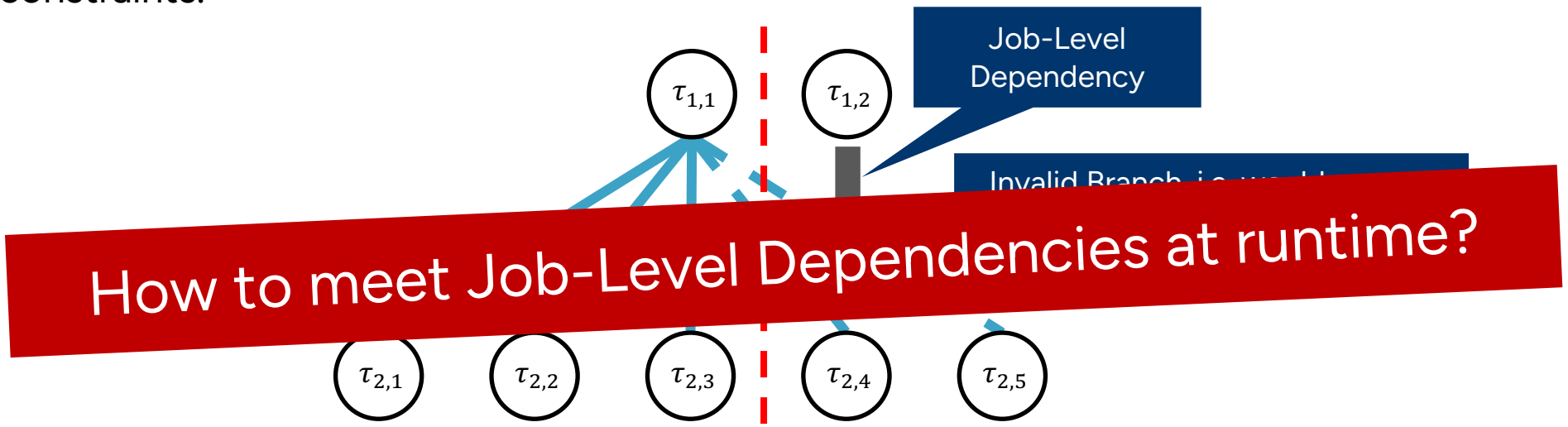
- A heuristic method exists to specify such job-level dependencies [1] for a set of cause-effect chains

End-to-end latency constraints are guaranteed to be met as long as the job-level dependencies are respected at runtim. I.e. a translation of data propagation delay constraints to precedence constraints on job-level.

[1] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *Proceedings of the 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016.

Motivating Example: Cause-Effect Chains

- Order selected jobs in such a way that any schedule that respects this order satisfies the timing constraints.



- A heuristic method exists to specify such job-level dependencies [1] for a set of cause-effect chains

End-to-end latency constraints are guaranteed to be met as long as the job-level dependencies are respected at runtim. I.e. a translation of data propagation delay constraints to precedence constraints on job-level.

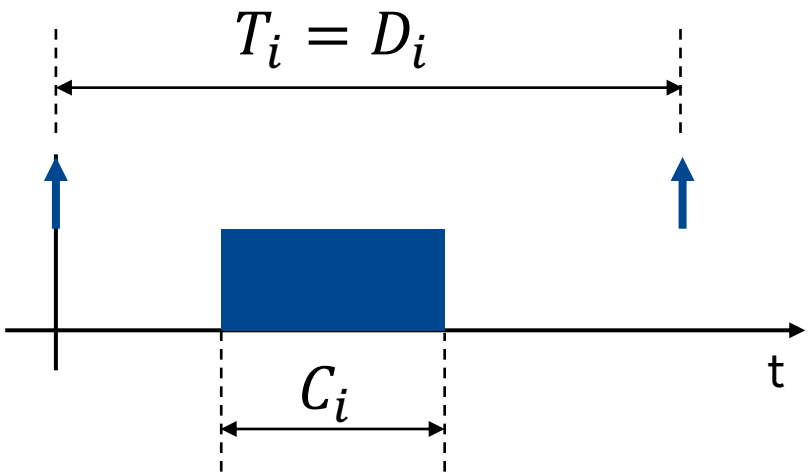
[1] M. Becker, D. Dasari, S. Mubeen, M. Behnam, and T. Nolte, "Synthesizing job-level dependencies for automotive multi-rate effect chains," in *Proceedings of the 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2016.

Outline

- System Model
- Background
- Approach Overview
- Merging Job-Level Dependencies as Optimization Problem
- Evaluation
- Conclusions

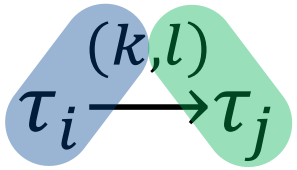
System Model

- A set of periodic tasks Γ
- All tasks are released synchronously at $t = 0$
- A task τ_i is characterized by:
  - Activation period T_i
  - Worst-Case Execution Time C_i
  - Deadlines are equal to periods $D_i = T_i$
- A set of job-level dependencies $\mathcal{D}$
  - For example generated by the approach of [1]

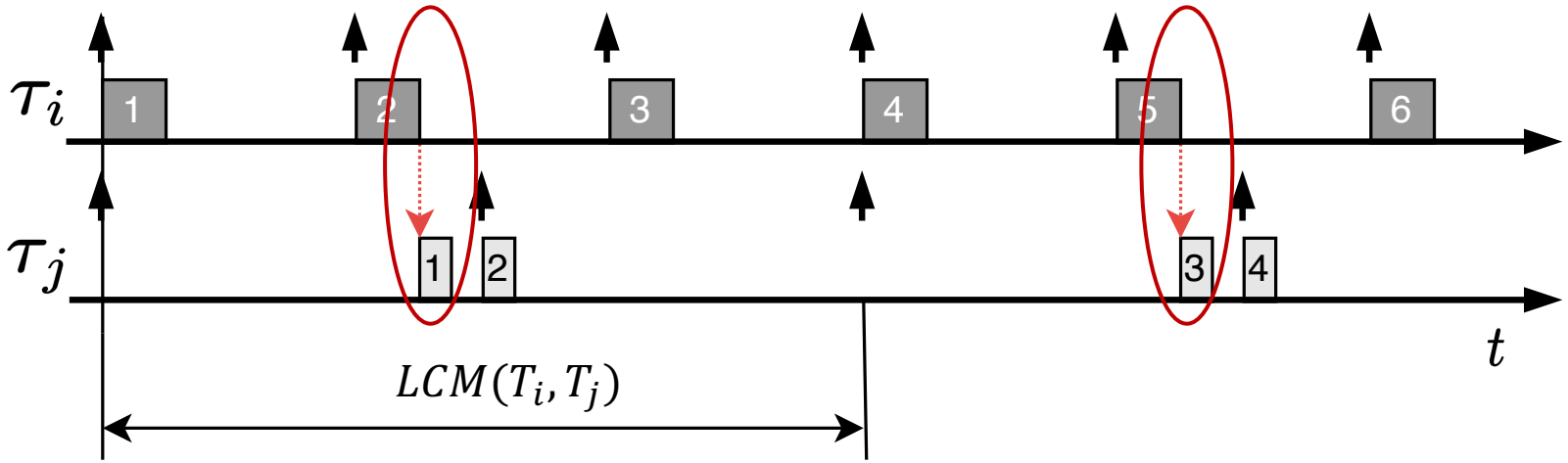


Job-Level Dependencies

- A job-level dependency (JLD) describes a partial ordering of tasks job's:
  - Every k[th] instance of τ_i must finish before every l[th] instance of τ_j.
  - A JLD repeats with its hyperperiod, i.e. $LCM(T_i, T_j)$



- Example:
  - $\tau_i \xrightarrow{(2,1)} \tau_j$, where $3T_i = 2T_j$



Job-Level Dependencies (JLD) at Runtime

- Time-Triggered Scheduling
  - Very easy to integrate in time-triggered scheduling
- Online Scheduling
  - Implementation using semaphores, but high overheads
  - By manipulating task offsets and deadlines, under single-core fixed-priority scheduling [2]
  - By manipulating task deadlines under Earliest Deadline First [3]
  - A combination can be applied to consider multi-core platforms with partitioned Earliest Deadline First scheduling [4]

Example:
FreeRTOS with 3 tasks and 2 JLDs
→ System Utilization increases from 59,54% to 60,49% (+0,95%)!

All approaches require knowledge of the platform and a specific scheduling algorithm

Goal of this paper: A method to realize JLDs independent of scheduling algorithm and OS-level synchronization

[2] J. Forget, F. Boniol, E. Grolleau, D. Lesens, and C. Pagetti, "Scheduling dependent periodic tasks without synchronization mechanisms," in 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010.

[3] H. Chetto, M. Silly, and T. Bouchentouf, "Dynamic scheduling of realtime tasks under precedence constraints," Real-Time Systems, vol. 2, no. 3, pp. 181–194, 1990.

[4] Klaus T, Becker M, Schröder-Preikschat W, Ulbrich P. Constrained data-age with job-level dependencies: How to reconcile tight bounds and overheads. In2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS), 2021.

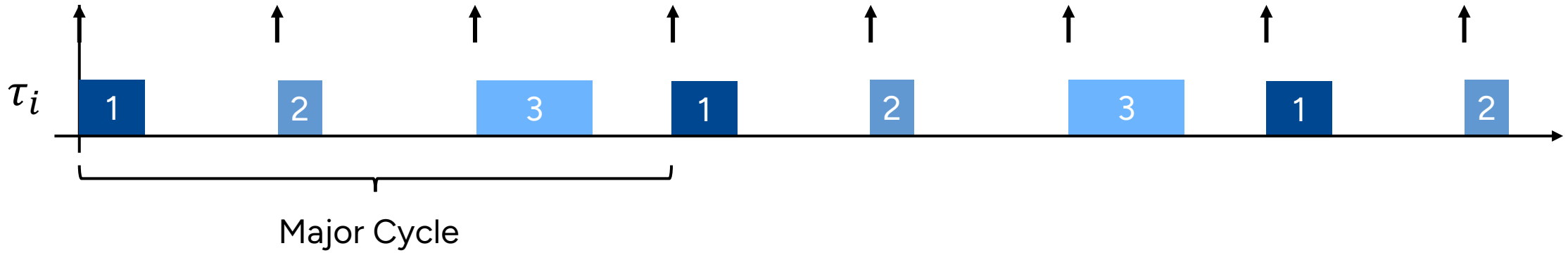
Background: Multi-Frame Task Model

- The multiframe task model [5] is a generalization of the periodic task model
- Instead of fixed WCET the execution time is described by an array of k WCET estimates
- Each execution is called a frame
- Frames repeat cyclically, described as the major cycle

$$\tau_i = (T_i, D_i, \{C_i^1, \dots, C_i^k\})$$

- For analysis, we can approximate a multiframe task by a periodic task assuming that $C = \max(C_i^1, \dots, C_i^k)$

- Example with $k = 3$



[5] A. K. Mok and D. Chen, “A multiframe model for real-time tasks,” IEEE transactions on Software Engineering, vol. 23, no. 10, 1997.

Background: Multi-Frame Task Model

- The multiframe task model [5] is a generalization of the periodic task model
- Instead of fixed WCET the execution time is described by an array of k WCET estimates
- Each execution is called a frame
- Frames repeat cyclically, described as the major cycle

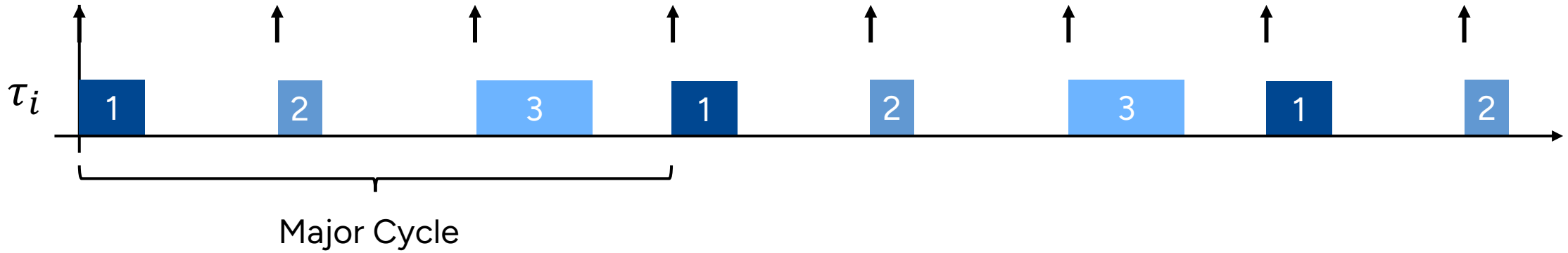
WCET estimates for all frames

$$\tau_i = (T_i, D_i, \{C_i^1, \dots, C_i^k\})$$

Activation Period

Deadline

- For analysis, we can ap me task by sk assuming that $C = \max(C_i^1, \dots, C_i^k)$
- Example with $k = 3$

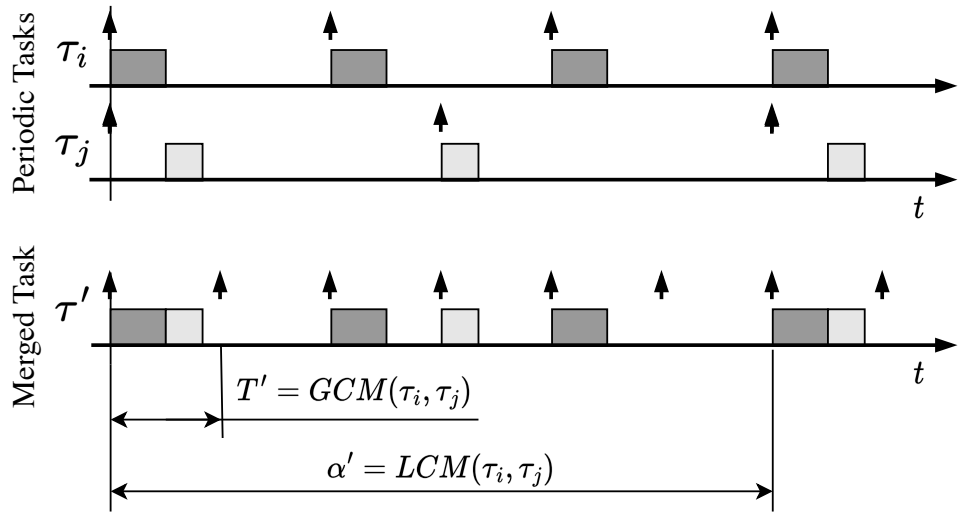


[5] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," IEEE transactions on Software Engineering, vol. 23, no. 10, 1997.

Background: APS for Task Merging

- The Arbitrary Periodic Solution (APS) [6] approach was introduced to assign AUTOSAR runnables (the elementary unit of computation in AUTOSAR systems) of different periods to the same task

- If two tasks are merged, they can be represented by a multiframe task τ'
- The period T' of τ' is the greatest common diviser (GCD) of all merged task periods
- The major cycle of τ' is the least common multiple (LCM) of all merged task periods
- In this work, we assume that the multiframe task has an arbitrary deadline D'



[6] F. Khenfri, K. Chaaban, and M. Chetto, “Efficient mapping of runnables to tasks for embedded autosar applications,” Journal of Systems Architecture, vol. 110, 2020.

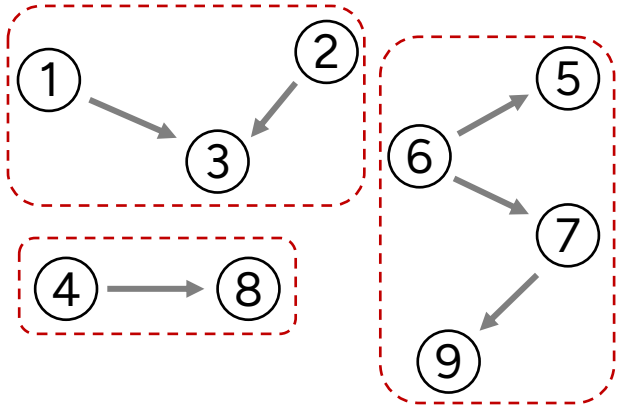
Approach Overview

Find Clusters of tasks connected by JLDs

Merge each task cluster to a new task

- Jobs of constraint tasks must appear in the right order by being placed either in different frames or in the right sequence in the same frame
- Balance the execution load in different frames
- Set the deadline of the multiframe task such that all original job deadlines are met

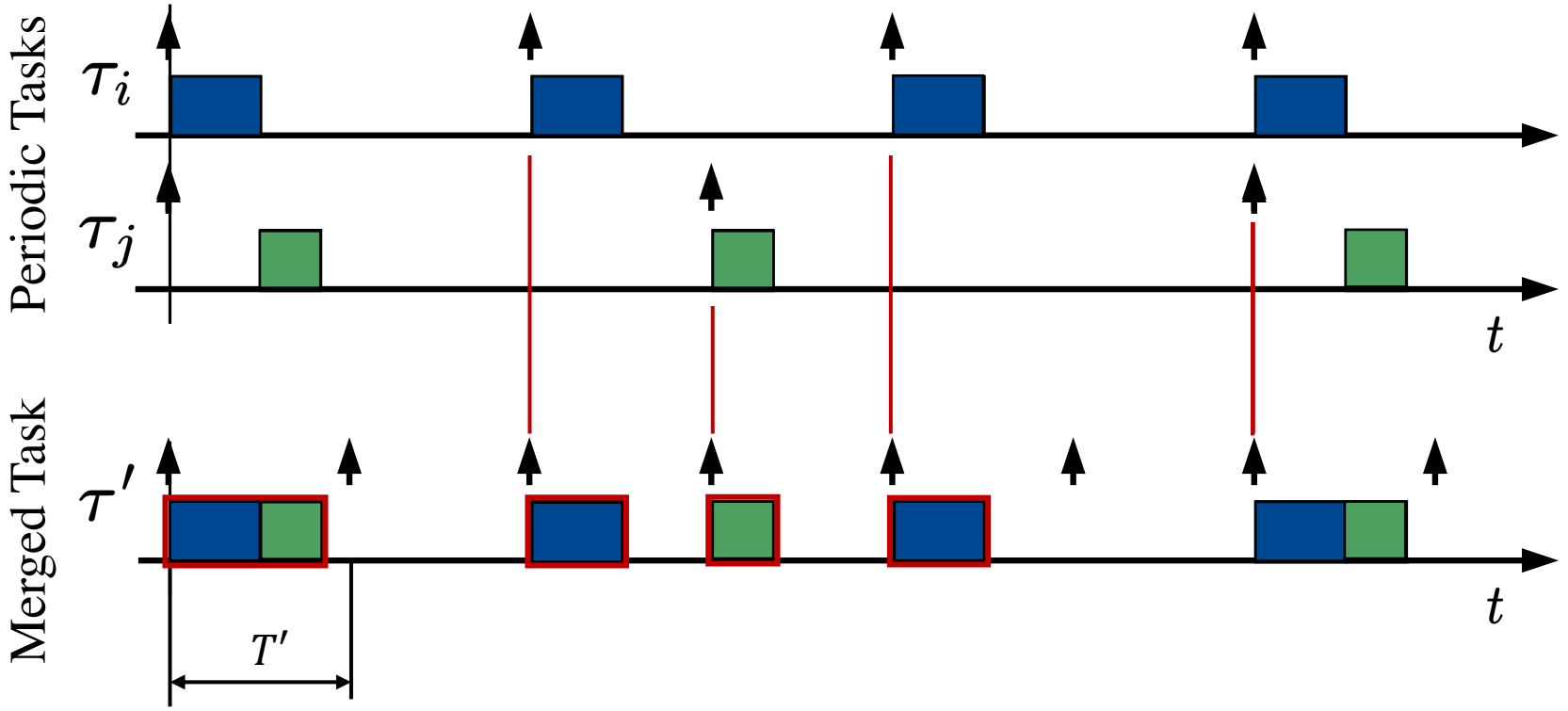
Approximate multiframe tasks as periodic task for analysis



Any scheduling algorithm / schedulability test can be used that supports periodic tasks with arbitrary deadlines

Deadline Assignment

- The deadline of the merged task is set such that all original job deadlines are met



$$D'^1 = 2\mathrm{T}'$$
$$D'^3 = 2\mathrm{T}'$$
$$D'^4 = 3\mathrm{T}'$$
$$D'^4 = 2\mathrm{T}'$$

$$D' = 2\mathrm{T}'$$

Task merging as optimization problem

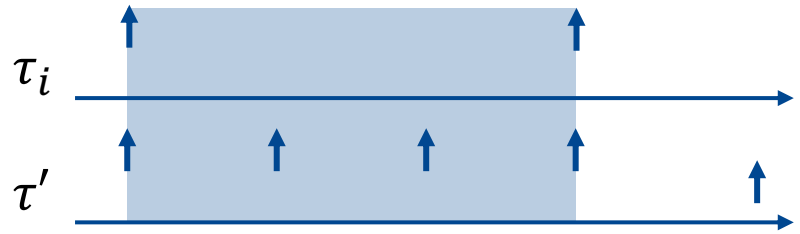
- We can formulate the task merging problem as Constraint Programming (CP) problem
  - Assign all jobs of merged tasks that are released during the major cycle of the multiframe task to frames of the multiframe task.

- Two types of decision variables
  - $x_{i,j} \in [1, \dots, f']$ is an integer that indicates the frame index of job $\tau_{i,j}$, where f' is the number of frames in a major cycle
  - $\phi_{i,j} \in [1, \dots, |\mathcal{J}'|]$ is an integer that represents the position of the job $\tau_{i,j}$ in its frame, where $\mathcal{J}'$ is the set of jobs assigned to all frames of the multiframe task.

- Helper formulations:
  - C'_i is the maximum cummulative execution time of a frame → to approximate the multiframe task as periodic task
  - D'_i is the resulting deadline of the multiframe task

Constraints

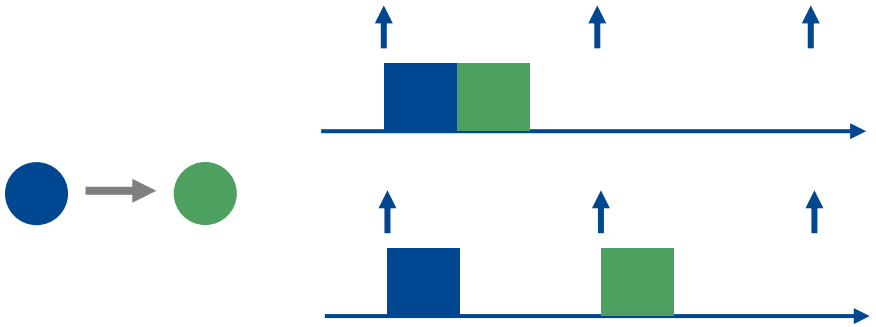
1. Each job must be allocated such that it executes during its original execution window



2. Execution time of a frame can not be larger than the period of the multiframe task

$$C' \leq T'$$

3. Jobs with precedence constraint must either be in different frames (in the right order) or are in the same frame in the right sequence

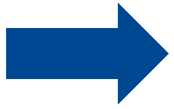


4. All decision variables $\phi_{i,j}$ have different values

→ Each job has a unique position.

Objective Function

- We want to minimize the effect merging has on system schedulability through the periodic abstraction of the multiframe task when approximated by a periodic task (i.e. $C = \max(C_i^1, \dots, C_i^k)$)
- The periodic task is then described by 3 parameter:
  - Period
  - Arbitrary Deadline
  - Worst-Case Execution Time



Maximize the slack of the task, i.e. $D' - C'$

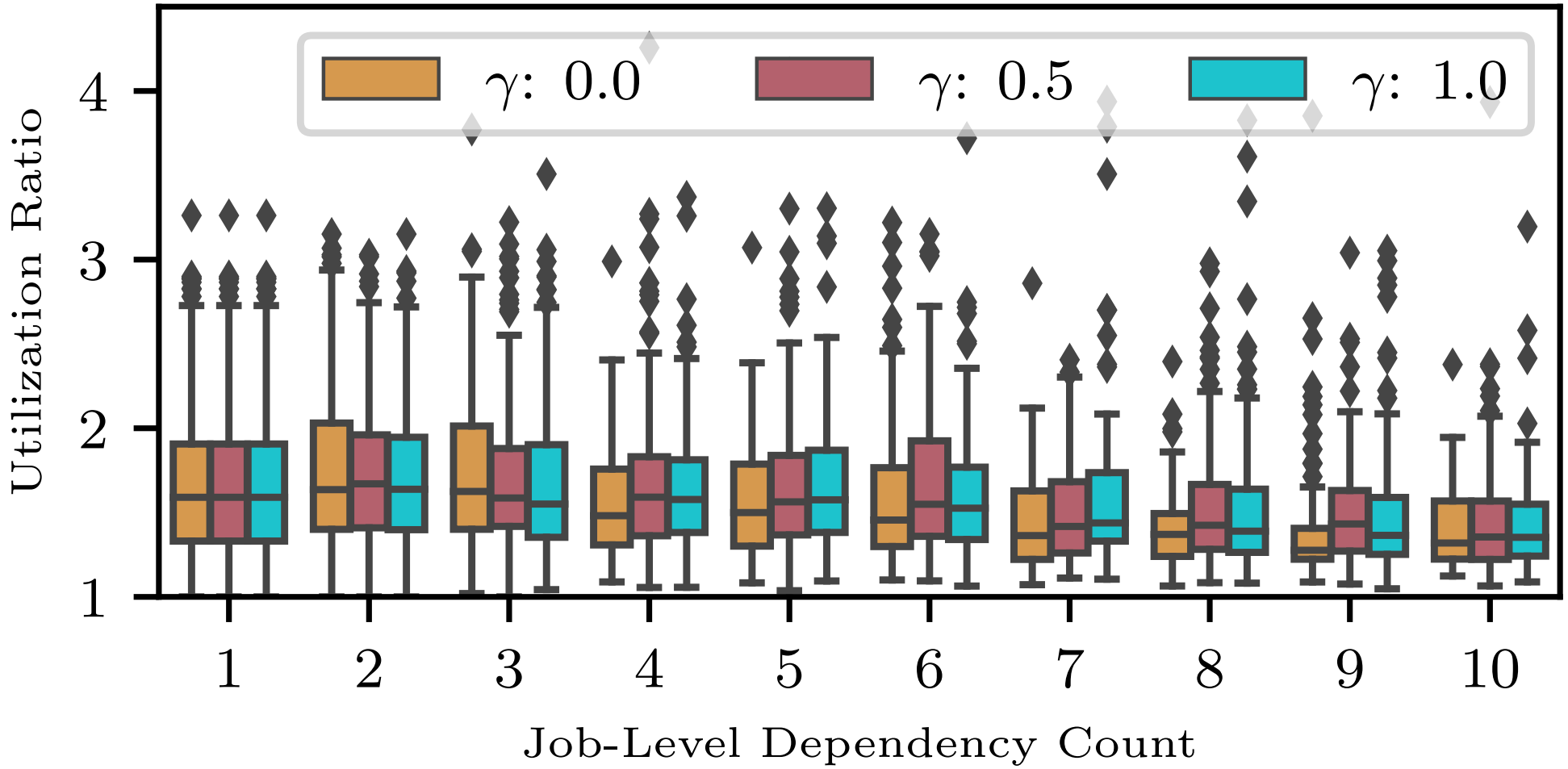
- This will lead to:
  - Large deadlines, i.e. jobs appear in early frames
  - Small approximated periodic execution times, i.e. frames are evenly loaded

Evaluation – Setup

- Two types of experiments
- Randomly generated JLD cluster where γ indicates branching of clusters
  - γ = 0 → Sequential dependencies, tasks in the cluster can't be part of a new dependency
  - γ = 1 → All tasks can be selected during generation, i.e. most branching
  - Periods: {1, 2, 5, 10, 20, 50, 100, 200} ms
  - WCET: [50, 150] μs
  - 200 systems per data point
- Existing task sets with JLDs generated to meet end-to-end delays
  - 838 publically available systems [7]
- For each experiment, timeout 60 seconds

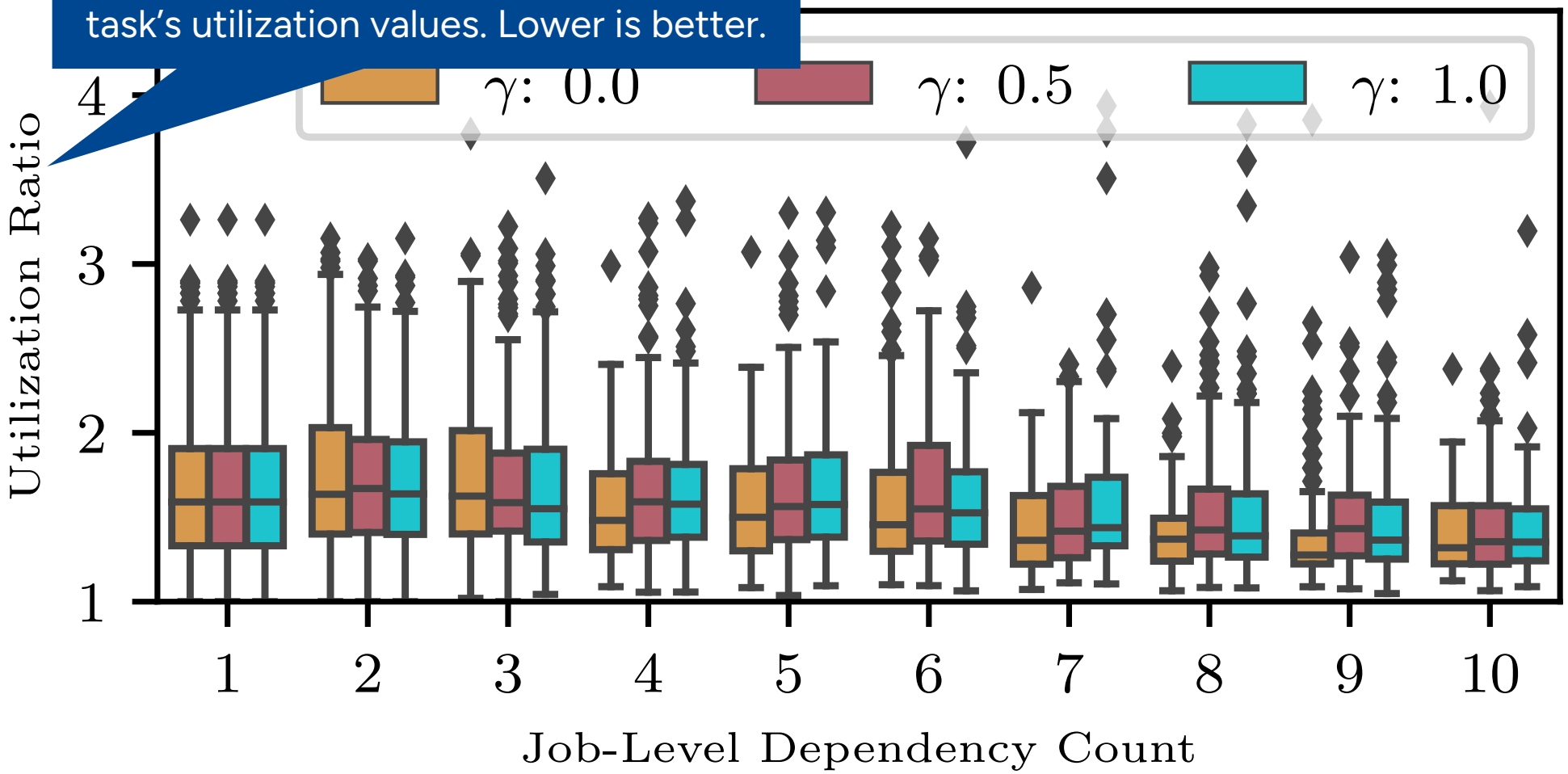
[7] Klaus, Tobias, Florian Franzmann, Matthias Becker, and Peter Ulbrich. "Data propagation delay constraints in multi-rate systems: Deadlines vs. job-level dependencies." In *Proceedings of the 26th International Conference on Real-Time Networks and Systems*, pp. 93-103. 2018.

Evaluation – Synthetic Tasksets



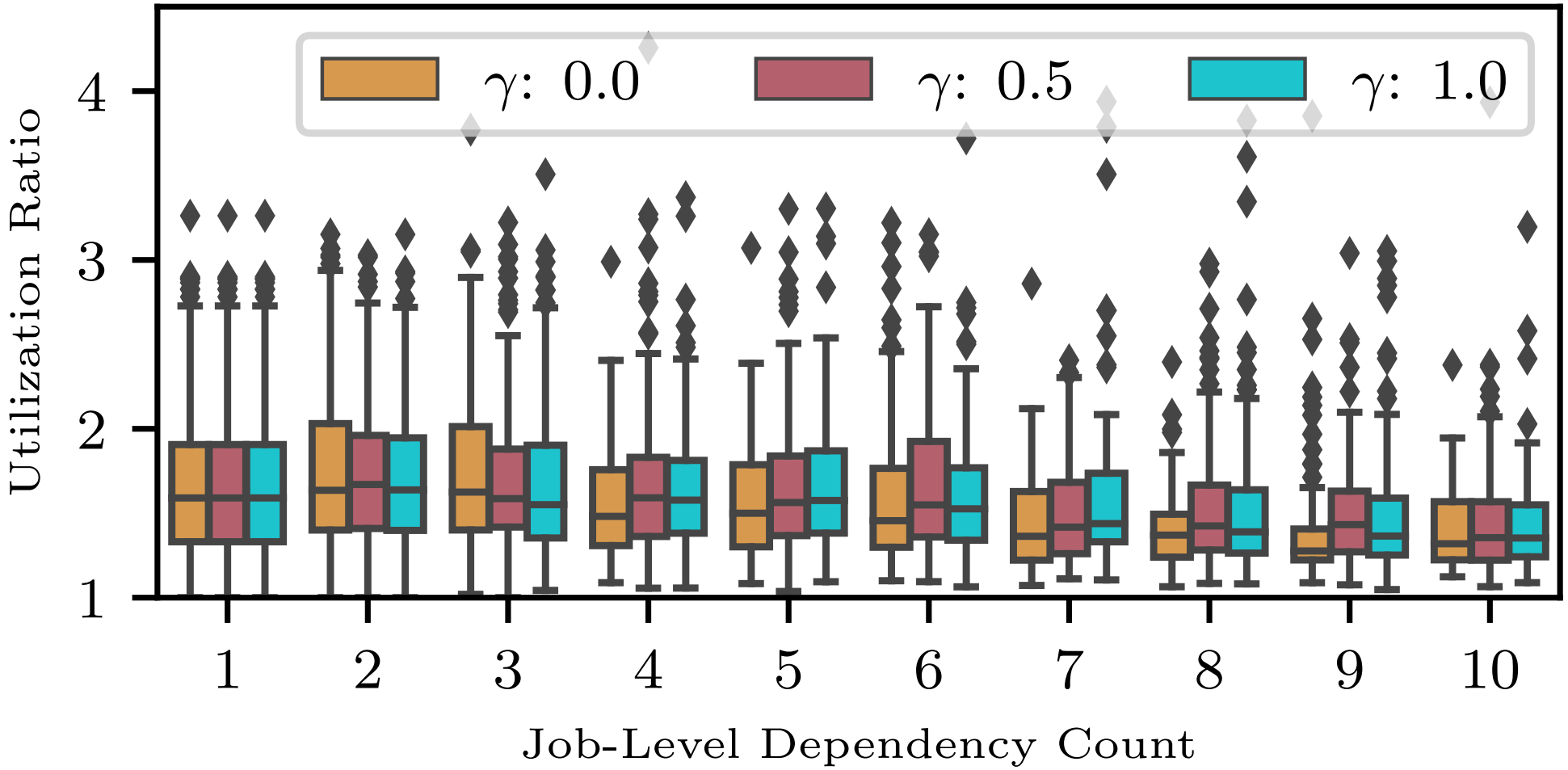
Evaluation – Synthetic Tasksets

Utilization of the periodic approximation of τ' compared to the sum of all merged task's utilization values. Lower is better.



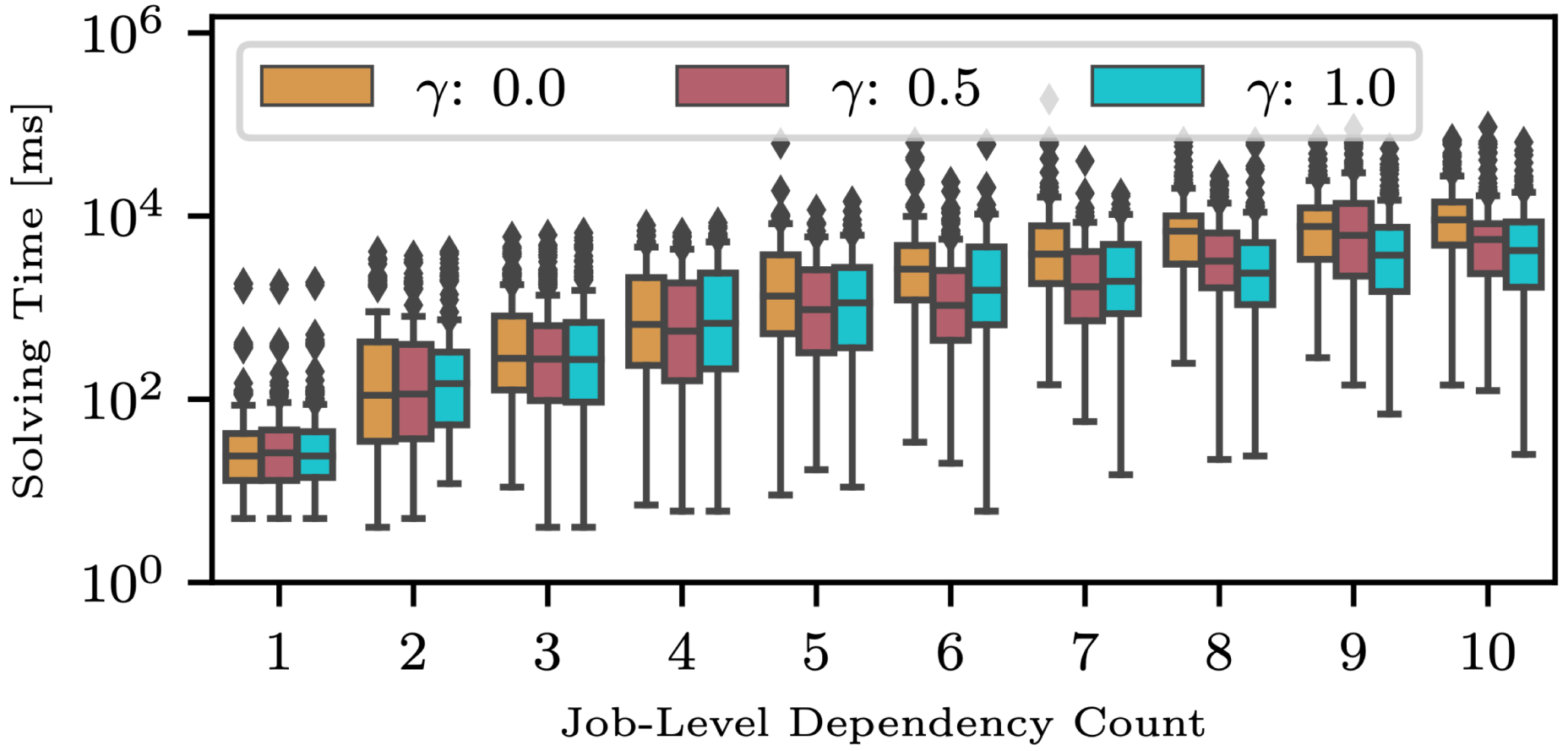
Evaluation – Synthetic Tasksets

A larger number of merged tasks results in a better utilization ratio as workload can be more evenly assigned to frames.



Evaluation – Synthetic Tasksets

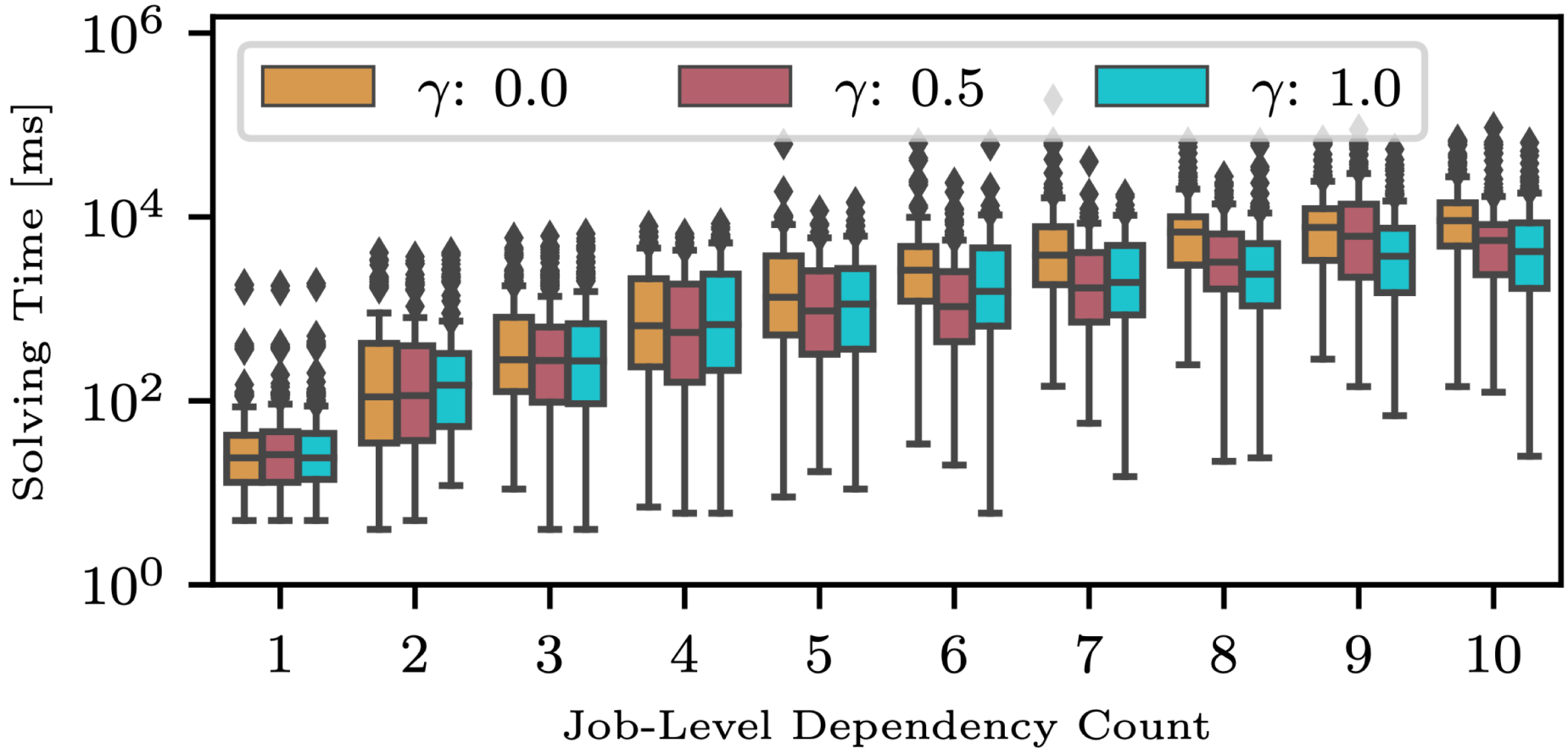
- Among the 6000 evaluated systems, only 31 reached a timeout and 3 of those could not find any solution before the timeout.
- All other systems could be optimally solved or are proven unfeasible.



Evaluation – Synthetic Tasksets

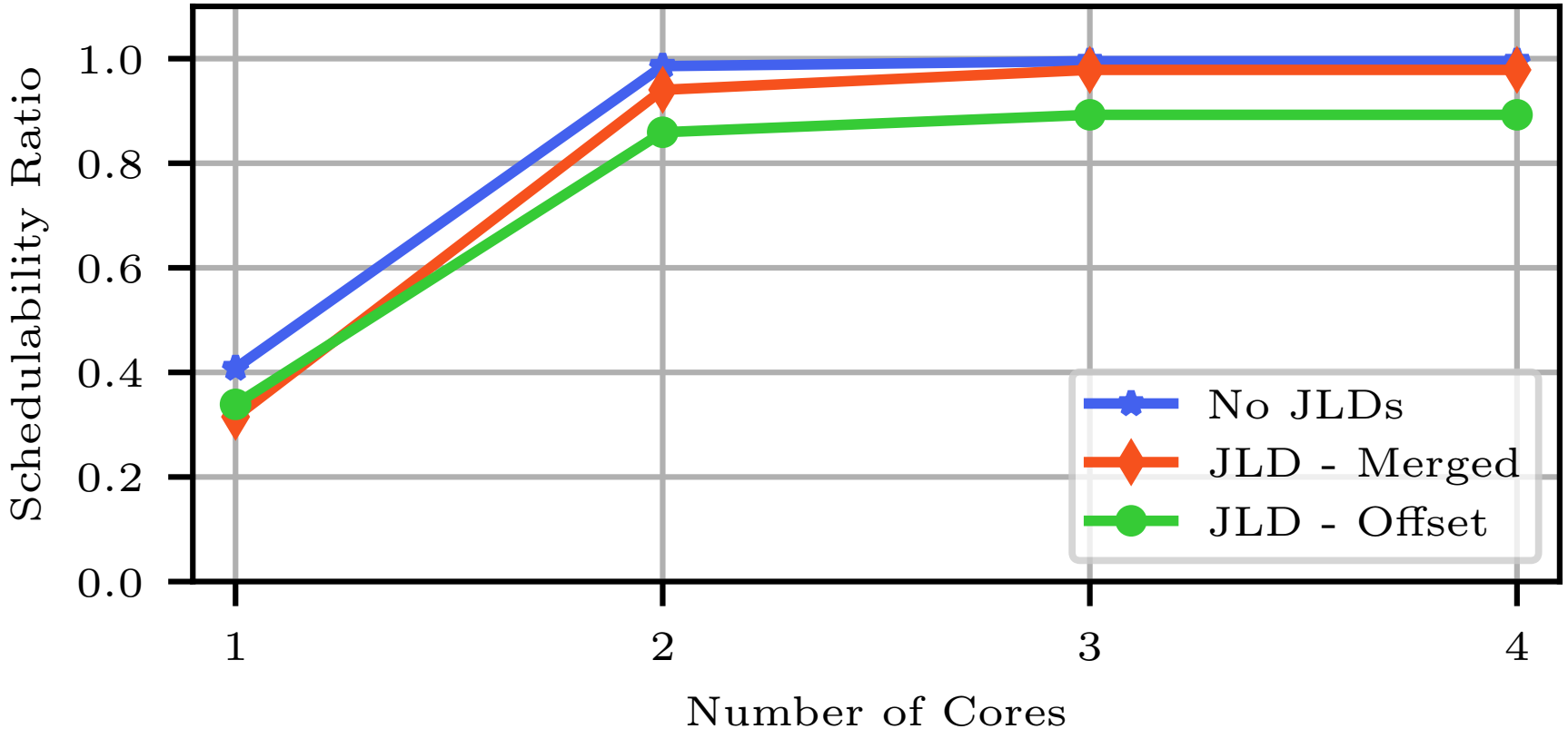
- Among the 6000 evaluated systems, only 31 reached a timeout and 3 of those could not find any solution before the timeout.
- All other systems could be optimally solved or are proven unfeasible.

With sequential dependencies more time is needed to find a solution



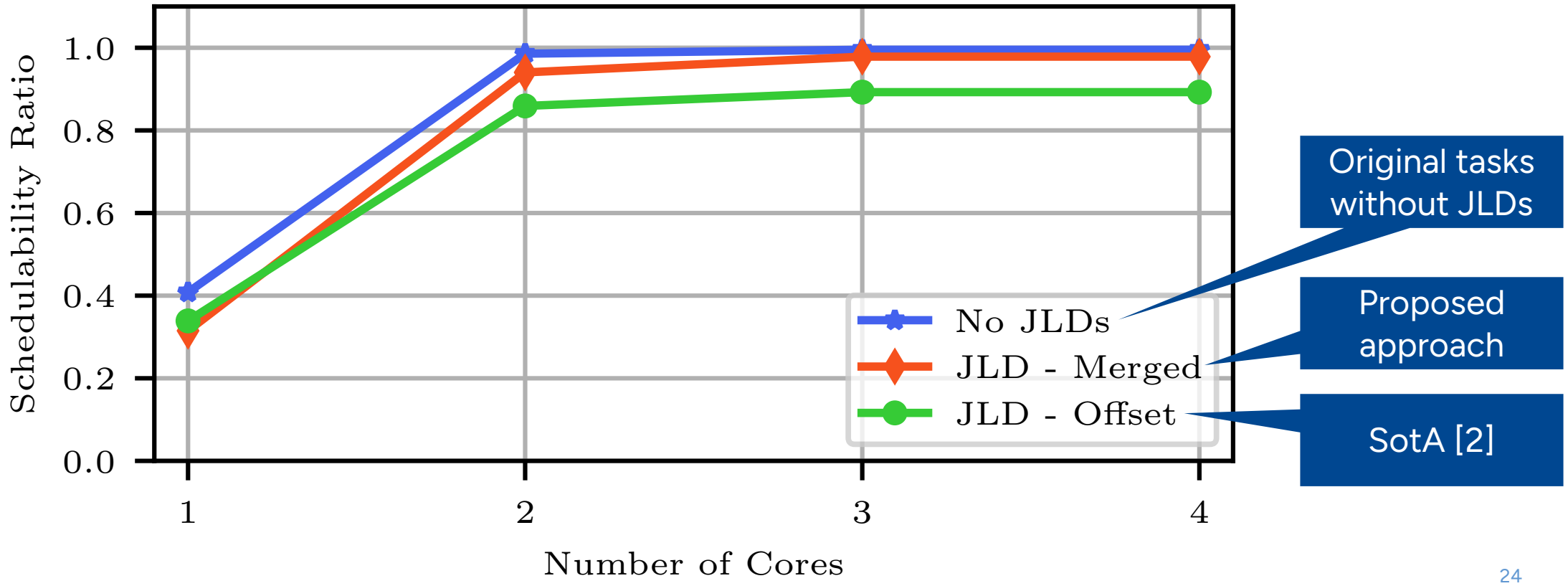
Evaluation – System-Level Evaluation

- Evaluate schedulability of complete taskset
- Partitioned Fixed Priority Preemptive Scheduling
- 1-4 Cores
- Task allocation “worst fit” heuristic



Evaluation – System-Level Evaluation

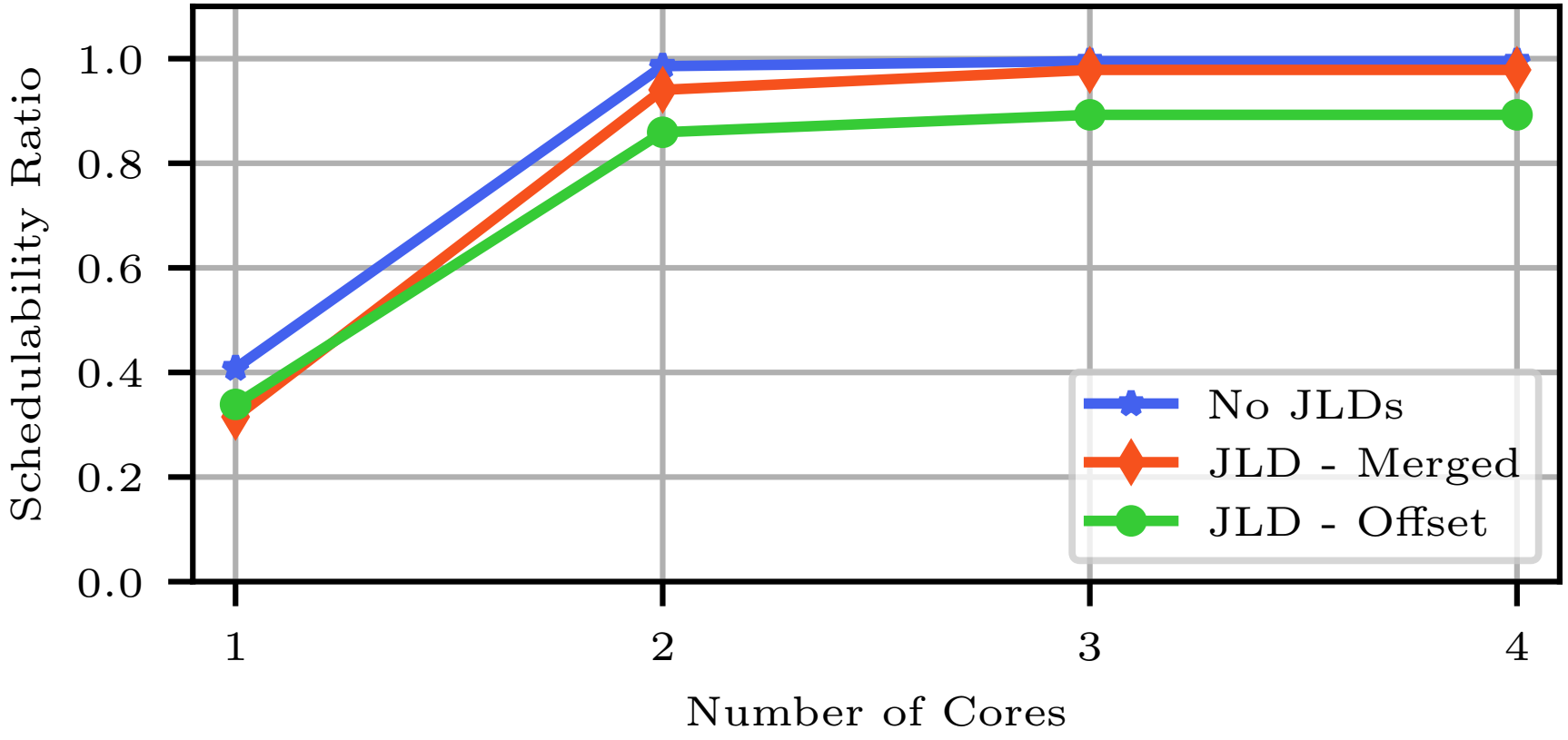
- Evaluate schedulability of complete taskset
- Partitioned Fixed Priority Preemptive Scheduling
- 1-4 Cores
- Task allocation “worst fit” heuristic



Evaluation – System-Level Evaluation

- Evaluate schedulability of complete taskset
- Partitioned Fixed Priority Preemptive Scheduling
- 1-4 Cores
- Task allocation “worst fit” heuristic

Proposed approach outperforms SotA once more than one core is available.



Conclusions and Future Work

- Realising job-level dependencies at runtime is challenging
- Merging clusters of dependent tasks is an effective way to meet job-level dependencies
- The approach is independent of the scheduling algorithm
- The problem can be efficiently solved using Constraint Programming

- Heuristic approach to construct merged tasks
- Avoid the periodic approximation and analyze the multiframe model directly

Thank you!