# Work-in-Progress: Tight Response-time Analysis for Periodic Preemptive Tasks under Global Scheduling

Pourya Gohari, Jeroen Voeten, and Mitra Nasri
Eindhoven University of Technology (TU/e), Eindhoven, The Netherlands

***Abstract*—While multicore real-time systems are extensively employed in the industry, research gaps still exist in developing a scalable analysis to find tight bounds on the worst-case response time (WCRT) of tasks scheduled by global preemptive scheduling policies. Additionally, the presence of release jitter poses a challenge where examining the earliest and latest release times may not derive WCRT. The existing analyses either provide very conservative bounds or face challenges in scaling to systems with numerous cores and tasks. This work provides preliminary foundations to derive tight WCRT bounds for tasks scheduled by global preemptive job-level fixed-priority scheduling policies (e.g., EDF and FP) on homogeneous multicore platforms by performing a reachability analysis using time-label-transition systems. Our solution uses 2 orders of magnitude less memory than UPPAAL and identifies on average 12% (up to 39%) more schedulable task sets than sufficient schedulability analyses (e.g., for systems with 4 cores and 10 tasks).**

***Index Terms*—Response-time analysis, global scheduling, preemptive tasks, periodic tasks, real-time systems.**

## I. Introduction

While more than 80% of industrial real-time systems already use or plan to use multicore platforms by 2024 [1], there are still research gaps in finding tight bounds on the worst-case response time (WCRT) of tasks scheduled by global scheduling policies on these platforms. Due to the NP-hardness of the problem, the existing response-time analyses for global scheduling policies suffer from two pitfalls: either they provide loose bounds [2]–[4], or they do not scale (run out of time or memory) as the system size grows [5]–[8].

**Related Work.** The first exact schedulability test for a preemptive global policy was introduced by Baker and Cirinei [5], where they mapped the problem into a finite state machine to perform an exhaustive search of reachable system states. However, the time and space complexity of this analysis prevents it from scaling to larger systems. According to the paper, the method can handle tasks with period values from the set $\{3, 4, 5\}$. Later, Bonifaci and Marchetti-Spaccamela [7] and Burmyakov et al. [8] developed a more efficient exact test by focusing on constrained-deadline tasks and using aggressive space reduction techniques. However, despite multiple enhancements until 2022, this analysis still can only handle systems with up to 4 cores and 13 tasks whose parameters are not numerically larger than 255 due to its high computational and space complexity. Guan et al. [6] proposed a timed-automata model in UPPAAL for schedulability analysis of periodic tasks. As we will show in Sec. IV, this test does not scale well. Nasri et al. [9], [10] have proposed a scalable and accurate reachability-based response-time analysis called schedule-abstraction technique. However, their method does not yet account for preemptive execution. Cucu-Grosjean and Goossens [11] proposed an exact simulation-based test for task sets without release jitter and execution time variation. Their solution is not applicable to the problem we consider here. Bertogna et al. [2] and Guan et al. [3] propose sufficient schedulability tests for global fixed-priority scheduling. However, as shown in Sec. IV these tests are pessimistic, especially for a system with utilization higher than 50%.

**This paper.** The aim of our work is to enhance the scalability of response-time analysis for periodic preemptive tasks under a global fixed-priority scheduling policy by introducing a new exact response-time analysis using a timed-labeled transition system. Our preliminary results indicate on average 12% more schedulable task sets than the existing sufficient schedulability tests [2], [3], and 18% more schedulable than the existing tests made in UPPAAL [6] (which frequently runs out of time or memory).

## II. Models and assumptions

We consider the problem of scheduling a set of independent periodic tasks, denoted by $\Gamma = \{\tau_1, \ldots, \tau_n\}$ (where $n$ is the number of tasks), on a homogeneous multicore system with $m$ cores. Each task $\tau \in \Gamma$ is identified by its period $T(\tau)$, relative deadline $D(\tau)$ (which can be hard or soft), maximum release jitter $\sigma(\tau)$, worst-case execution time (WCET) $C(\tau)$. It's important to note that Ha and Liu [12] have proven that in global job-level fixed-priority (JLFP) scheduling policy, the WCRT occurs when jobs execute with their WCET. Therefore, we always assume that tasks are running with their WCET.

Each task $\tau \in \Gamma$ releases *jobs* (task instances) periodically with a period $T(\tau)$ during the lifetime of the system. Hence, the $k^{\text{th}}(1 \leq k)$ job of a task $\tau$ (referred to as job $j$ for the simplicity of notations), is released within the interval $\left[r^{\min}(j), r^{\max}(j)\right]$, where $r^{\min}(j) = (k-1) \times T(\tau)$ is the earliest release time (a.k.a. *arrival time* in Audsely's definitions [13]) and $r^{\max}(j) = r^{\min}(j) + \sigma(\tau)$ is the latest release time of the job. We assume that jobs are executed until completion (even when they are tardy).

For the sake of simplicity, we will create a finite set of jobs denoted by $\mathcal{J}$. This set will include all jobs that could potentially appear in a hyperperiod. For periodic tasks with constrained deadlines, the hyperperiod refers to the shortest time window after which the sequence of job arrivals repeats. The hyperperiod $H$ is calculated by finding the least common multiple (LCM) of the task periods

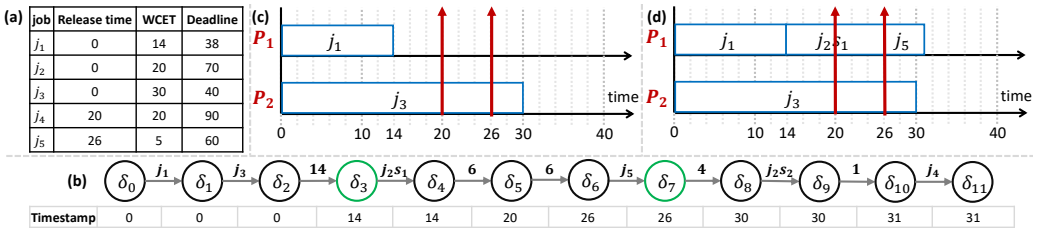


Fig. 1: The TLTS for a preemptive job set, **(a)** the job set, **(b)** TLTS, **(c)** schedule in state $\delta_3$, **(d)** schedule in state $\delta_7$.

$H = lcm(T(\tau_1), T(\tau_2), \ldots, T(\tau_n))$ [14]. The job set $\mathcal{J} = \{j_1, j_2, \ldots, j_z\}$ comprises all jobs that could be released by tasks in $\Gamma$ during the interval $[0, H)$. We denote the number of jobs in the job set by $z = |\mathcal{J}|$.

We assume the system is scheduled using a work-conserving job-level fixed-priority (JLFP) policy, which includes policies such as the earliest-deadline first (EDF) and fixed-priority (FP). Each job $j$ has an assigned priority $P(j)$ given by the JLFP scheduling policy. For example, in EDF, the absolute deadline of a job can denote its priority. We assume that a numerically smaller value of $P(j)$ implies higher priority. Any ties in priority are broken arbitrarily in a deterministic way.

## III. Response-time analysis

In our response-time analysis, we use a *timed-labeled transition system* (TLTS) to represent the operational semantics of the scheduling problem. TLTS is an abstract representation of the system in the form of a directed graph consisting of states and transitions (labeled edges) and is commonly used to describe the behavior of discrete systems [15]. In a TLTS, transitions represent the activation of functions/events and the passage of time.

### A. System state representation

We express a system state $\delta$ by a tuple $\big(TS(\delta), A(\delta), \mathcal{E}(\delta), \mathcal{J}^D(\delta), \mathcal{J}^P(\delta)\big)$, where $TS(\delta)$ is the time stamp of the state, $A(\delta)$ is the status of resources (described as a set of resource-availability times), $\mathcal{E}(\delta)$ is the set of scheduling events, $\mathcal{J}^D(\delta)$ is the set of jobs dispatched until the system state $\delta$, and $\mathcal{J}^P(\delta)$ is the set of the remaining segment of the preempted jobs. The set of scheduling events consists of times when the schedule may change. A schedule that follows a work-conserving JLFP scheduling policy can only change when a new job is released or a core becomes available (a task is completed). Hence, the scheduling event set for this scheduling policy contains the times that a new job is released or a core becomes available.

### B. System state transition

A state $\delta$ evolves into a new state $\delta'$ via a state transition labeled by either a job or a segment (dispatch transition) or a time value (time transition) indicating the passage of time.

When a dispatch transition occurs, it may affects the availability of resources, dispatched jobs, and preempted jobs. In our analysis, we use a conservative lookahead method for dispatching a job. This is done by verifying if there are any jobs that are released while job $j$ is being executed, which have a higher priority compared to job $j$. Therefore, when a job $j$ is selected to dispatch as the highest priority job that has been released, we first check if this job can be preempted in the future or not. If a higher priority job is released at time $t$ during the execution of $j$, job $j$ is divided into two segments at time $t$. Then, only the first segment that executes before the higher-priority job is released is executed, while the second segment is added to the preempted set of the next system state. Otherwise, if there is no higher priority job that can preempt job $j$, the whole job $j$ will be dispatched.

On the other hand, if there is no job that can be dispatched in the current state, the state evolves into a new state by a time transition to the next scheduling event. A time transition affects only the state's time stamp; all other information in the new state remains the same as in the previous state.

**Example 1.** Fig. 1 illustrates an example of making TLTS for a system with two cores that uses the EDF scheduling policy. The system starts with state $\delta_0$ where all cores are available and the time stamp is 0. Among jobs $j_1$, $j_2$, and $j_3$ that are released in time zero, the highest-priority jobs, namely, $j_1$ and then $j_3$ are dispatched on the two cores. After these two actions, no other core is available at time 0, therefore state $\delta_2$ evolves to state $\delta_3$ with a time transition (to the first time that a core becomes available, i.e., at time 14). In state $\delta_3$, job $j_2$ has the highest priority. Before dispatching $j_2$, we will look ahead to check for how long it can be executed before it is certainly preempted by another job, at the upcoming potential release times of higher-priority jobs, i.e., jobs $j_4$ and $j_5$ (which will be released at times 20 and 26). Between $j_4$ and $j_5$, since job $j_5$ has a higher priority (smaller deadline) than job $j_2$, it can preempt the execution of $j_2$. Hence, we consider that $j_2$ will be preempted at time 26 (the release time of $j_5$) and break it into two segments $j_2s_1$ and $j_2s_2$. Now, the first segment $j_2s_1$ is dispatched and state $\delta_3$ evolves to state $\delta_4$.

In state $\delta_4$, no core is available, therefore, it evolves to a new state $\delta_5$ by a time transition (that progresses the time to the next scheduling event at time 20). At time 20, since both cores are occupied by $j_2s_1$ and $j_3$ and no pending higher priority job exists, state $\delta_5$ evolves to a new state by time transition to the next scheduling event at time 26. This exploration continues until all the jobs in the job set have been dispatched.

**Supporting release jitter.** Considering the release jitter of the tasks creates uncertainty in the time that a job could release and exist in a system state. As a result, we could have multiple possible transitions between system states. To the best of our knowledge, no previous work has considered the impact of

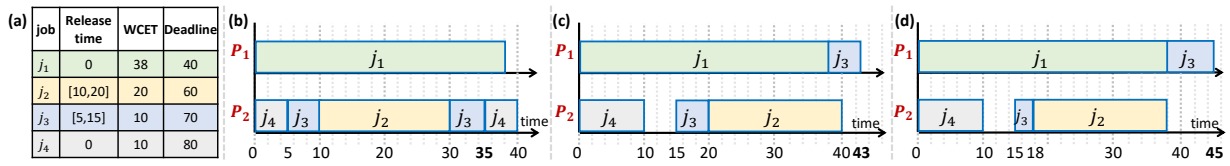


Fig. 2: The impact of release jitter on the response time, **(a)** the job set, **(b)** schedule when all the job release as early as possible, **(c)** schedule when all the jobs are released as late as possible, **(d)** schedule when $j_2$ is released at time 18 and $j_3$ is released as late as possible.

adding a release jitter to tasks in schedulability analysis.

**Example 2 (timing anomaly with release jitter).** Coping with release jitter presents a challenge because merely considering the minimum or maximum release jitter does not provide safe response time bounds for the jobs. Fig. 2 illustrates a job set with release jitter scheduled using the EDF policy. Fig. 2b and Fig. 2c show the schedule where all the jobs are released as early and as late as possible, respectively. Furthermore, Fig. 2d depicts the schedule for the scenario where job $j_2$ is released at time 18 and job $j_3$ is released as late as possible (at time 15). It is clear that in the last scenario, the response time of job $j_3$ exceeds that of the other two scenarios.

In our work, to address this challenge, we consider all possible job release times in the scheduling event set. We also introduce rules to select the possible highest-priority jobs in a state that can be dispatched next. In summary, to determine potential next transitions following the state $\delta$, we use the following rules if a job $j$ is certainly released in $\delta$ (namely, $r^{\max}(j) < TS(\delta)$):

R1) We find the highest priority certainly released job $j^c$. This job can certainly be dispatched after state $\delta$;

R2) every other possibly released job $j^p$ ($r^{\min}(j^p) \leq TS(\delta) < r^{\max}(j^p)$) that has a higher priority than $j^c$ can also be dispatched after state $\delta$.

On the other hand, if there is no certainly released job in $\delta$:

R3) Every possibly released job $j^p$ ($r^{\min}(j^p) \leq TS(\delta) < r^{\max}(j^p)$) can be dispatched after state $\delta$ and make a new dispatch transition;

R4) a time transition to the next state is also possible.

**Example 3 (handling release jitter).** Fig. 3 shows an example of a system with two cores that uses the EDF scheduling policy where the jobs have release jitter. As shown in Fig. 3a, all jobs have a release jitter, therefore, we have to consider all possible release scenarios. As an example, in the first state of the TLTS (at time 0), any combination of jobs $j_1$, $j_2$, and $j_3$ could be released and every one of them could be the highest-priority job and dispatch after $\delta_0$ (Rule R3).

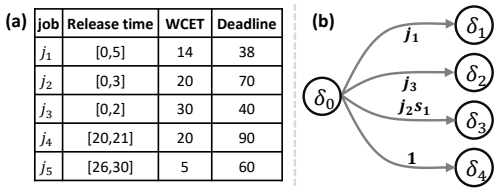


Fig. 3: The example of a system with release jitter, **(a)** the job set, **(b)** possible transition from the first system state of TLTS.

Also if no job is released at time 0, the state evolves to a new state by a time transition to the next scheduling event at time 1 (Rule R4). Fig. 3b depicts the possible transition from $\delta_0$.

**Merging.** As demonstrated in the previous example, timing uncertainty has the potential to generate multiple branches within the TLTS. To mitigate the rapid expansion of the state space, we introduce a merging rule to combine similar states and reduce the branches. In our analysis, two states merge into one state if and only if they have (i) the same timestamp, (ii) the same dispatched jobs, and (iii) the same preempted jobs.

**Next steps.** Our first next step is to investigate a smarter look-ahead method to avoid unnecessarily splitting jobs into segments and reduce state generation. Second, we intend to add a fast-forward mechanism to the time transition to ignore the event times that cannot change the schedule. We believe these steps can help to prone the unnecessary states and significantly improve the scalability of our analysis.

## IV. Empirical Evaluation

This section compares the runtime and schedulability ratio of our analysis against state of the art.

**Baseline.** We considered (i) the simulation-based analysis using SimSo [16] (SimSo-EDF-P) which is only a necessary schedulability test, (ii) exact UPPAAL-based schedulability test of Guan et al. [6] (UPPAAL-RM-P) for global fixed-priority scheduling of periodic tasks, (iii) the exact test of Burmyakov et al. [8] for global fixed-priority scheduling of sporadic tasks (Burmyakov-RM-SP), and (iv) the sufficient tests of Guan et al. [3] (Guan-RM-SP) and (v) Bertogna et al. [2] (Bertogna-RM-SP) for sporadic tasks.

**Task set generation.** We generated 200 random synthetic task sets for each data point with $n$=10 periodic tasks and a total utilization of $U$ for a system with $m$=4 cores, using the UUnifast method [17]. Due to UPPAAL's variable limitation to integer, the periods are selected from the set $\{x \times 10^3 \mid 1 \leq x \leq 32\}$ with the log-uniform distribution. We discarded any task set that had more than 100,000 jobs per hyperperiod.

**Experimental setup.** We implemented all methods as a single-threaded C++ program with access to a maximum of 256GB RAM. Our experiments were conducted on a cluster of AMD Rome 7H12 processors clocked at 2.6GHz.

**Results.** Fig. 4a illustrates a significant difference in the schedulability ratios between our solution and state-of-the-art tests (particularly for utilization values larger than 50%). More specifically, with 70% utilization, our analysis Our-RM (for periodic tasks scheduled by the rate-monotonic scheduling) identifies 74% of schedulable task sets while Guan's test and

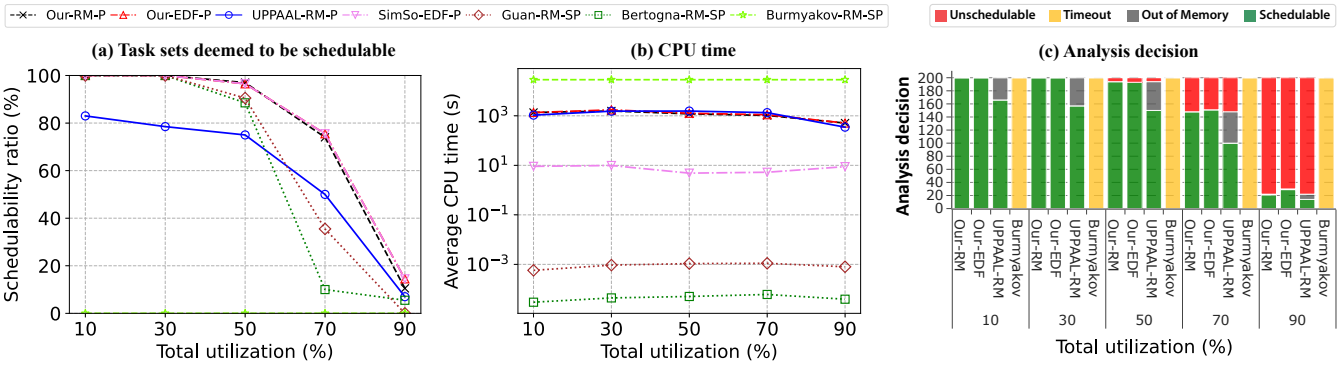


Fig. 4: Experimental results for synthetic task sets, **(a)** task sets deemed to be schedulable, **(b)** CPU time, **(c)** analysis decision.

Bertogna's test can only detect 35% and 10%, respectively. Also, the exact test (UPPAAL) can only identify 50% of schedulable task sets (because it often runs out of time or memory as shown in Fig. 4c). Furthermore, Burmyakov's exact test [8] failed to complete any test within 8 hours of the time budget and it was unable to detect any schedulable task set at any system utilization level.

Fig. 4b shows the average CPU time of each test. As expected, the two sufficient tests (Guan-RM-SP and Bertogna-RM-SP) are significantly faster than other tests. The average runtime of our analysis (for EDF) is 1197 seconds (about 20 minutes) and it manages to analyze almost all task sets without timeout while the UPPAL-based test runs out of memory for the majority of schedulable yet high-utilization task sets (which are often the hardest to analyze). Moreover, as expected, the simulation-based analysis (SimSo-EDF-P) is faster than ours because it only looks at one scenario and therefore provides only a necessary test. Finally, our analysis revealed that the gap between schedulable tasks and those that pass a simulation base test (SimSo-EDF-P) is extremely small, prompting that the outcome of a simulation-based test for task sets that do not have release jitter could be a good indicator of the true schedulability ratio.

## V. Conclusion and Future Work

In this work, we used timed-labeled transition systems to develop a new response-time analysis for preemptive periodic tasks under global JLFP scheduling. Our empirical evaluation shows that on average, our analysis identifies 18% more schedulable task sets compared to exact analysis in UPPAAL (which frequently runs out of time or memory). It also finds 12% more schedulable task sets than sufficient analyses (e.g., for systems with 4 cores and 10 tasks). Our future work includes providing formal proofs of soundness, adding support for precedence constraints, and further improving the scalability by applying space reduction techniques such as pruning and symmetry reduction.

## Acknowledgment

This work was supported by the Dutch national e-infrastructure (grant no. EINF-5183) and the EU ECSEL project TRANSACT (grant no. 101007260).

## References

[1] B. Akesson, M. Nasri, G. Nelissen, S. Altmeyer, and R. I. Davis, "A comprehensive survey of industry practice in real-time systems," *Real-Time Systems*, vol. 58, no. 3, pp. 358–398, 2022.

[2] M. Bertogna, M. Cirinei, and G. Lipari, "Schedulability analysis of global scheduling algorithms on multiprocessor platforms," *IEEE Transactions on parallel and distributed systems*, vol. 20, no. 4, pp. 553–566, 2008.

[3] N. Guan, M. Stigge, W. Yi, and G. Yu, "New response time bounds for fixed-priority multiprocessor scheduling," in *Real-Time Systems Symposium (RTSS)*, 2009, pp. 387–397.

[4] Y. Sun and M. Di Natale, "Assessing the pessimism of current multicore global fixed-priority schedulability analysis," in *ACM Symposium on Applied Computing*, 2018, pp. 575–583.

[5] T. P. Baker and M. Cirinei, "Brute-force determination of multiprocessor schedulability for sets of sporadic hard-deadline tasks," in *International Conference On Principles Of Distributed Systems*, 2007, pp. 62–75.

[6] N. Guan, Z. Gu, Q. Deng, S. Gao, and G. Yu, "Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking," in *IFIP International Workshop on Software Technologies for Embedded and Ubiquitous Systems*, 2007, pp. 263–272.

[7] V. Bonifaci and A. Marchetti-Spaccamela, "Feasibility analysis of sporadic real-time multiprocessor task systems," *Algorithmica*, vol. 63, pp. 763–780, 2012.

[8] A. Burmyakov, E. Bini, and C.-G. Lee, "Towards a tractable exact test for global multiprocessor fixed priority scheduling," *IEEE Transactions on Computers*, vol. 71, no. 11, pp. 2955–2967, 2022.

[9] M. Nasri, G. Nelissen, and B. B. Brandenburg, "Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling," in *Euromicro Conference on Real-Time Systems (ECRTS)*, 2019, pp. 21–1.

[10] S. Ranjha, G. Nelissen, and M. Nasri, "Partial-order reduction for schedule-abstraction-based response-time analyses of non-preemptive tasks," in *Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2022, pp. 121–132.

[11] L. Cucu-Grosjean and J. Goossens, "Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms," *Journal of systems architecture*, vol. 57, no. 5, pp. 561–569, 2011.

[12] R. Ha and J. W. Liu, "Validating timing constraints in multiprocessor and distributed real-time systems," in *International conference on distributed computing systems*, 1994, pp. 162–171.

[13] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software engineering journal*, vol. 8, no. 5, pp. 284–292, 1993.

[14] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[15] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[16] M. Chéramy, P.-E. Hladik, and A.-M. Déplanche, "SimSo: A simulation tool to evaluate real-time multiprocessor scheduling algorithms," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2014, pp. 37–42.

[17] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-time systems*, vol. 30, no. 1-2, pp. 129–154, 2005.