Work-in-Progress: Partial-Order Reduction in Reachability-based Response-Time Analyses

Sayra Ranjha[1,2], Mitra Nasri[1], Geoffrey Nelissen[1]
[1] *Eindhoven University of Technology, The Netherlands* [2] *Delft University of Technology, The Netherlands*

***Abstract*—The temporal correctness of safety-critical systems is typically guaranteed via a response-time analysis (RTA). However, as the systems become complex (e.g., parallel tasks running on a multicore platform), most existing RTAs either become pessimistic or do not scale. To make a trade-off between accuracy and scalability, recently, a new reachability-based RTA called *schedule-abstraction graph* (SAG) has been introduced by Nasri et al. It explores the space of possible decisions that a *scheduling policy* can take while dispatching a set of tasks or jobs on processing resources. The analysis is at least three orders of magnitude faster than other exact RTAs and is able to identify many more schedulable task sets than the existing fixed-point iteration-based analyses.**

One fundamental limitation of the SAG analysis is that *in its reachability graph, each edge can only include one single scheduling decision*. Therefore, the graph grows exponentially when there are large uncertainties in the release time or execution time of the jobs. In this paper, we improve the scalability of the SAG analysis by introducing *partial-order reduction* (POR) rules that allow combining multiple scheduling decisions on one edge and hence avoiding combinatorial exploration of all possible orderings between jobs. An empirical evaluation shows that our solution is able to reduce the runtime by *five orders of magnitude* and the number of states by 98%.

***Index Terms*—Real-time systems, response-time analysis, partial-order reduction, reachability analysis**

I. Introduction

Temporal correctness of safety-critical real-time systems is typically guaranteed via a response-time analysis whose goal is to determine the *worst-case response time* (WCRT) of a set of input jobs/tasks when they are scheduled by a given scheduling policy on a computing resource. Ekberg [2] showed that most variations of the *response-time analysis* (RTA) problem to analyze periodic tasks scheduled by a *job-level fixed-priority* (JLFP) policy such as the *fixed-priority* (FP) or *earliest-deadline first* (EDF) are NP-hard even for a uniprocessor platform.

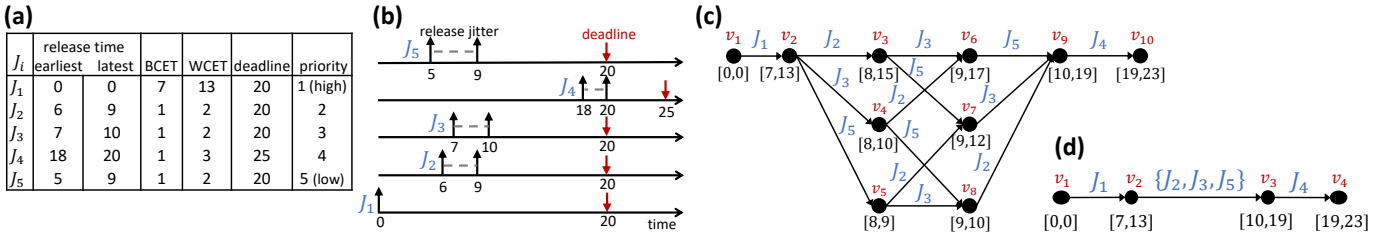
Related work. Known *exact* WCRT analyses fall into two general categories: (i) fixed-point iteration-based analyses [3,4] that have pseudo-polynomial time complexity and (ii) *reachability-based analysis* (RBA) [1,5]–[10]. Fixed-point iteration-based analyses are typically faster than RBAs, however, they are exact only in special cases, e.g., preemptive periodic or sporadic tasks scheduled by FP [3] or EDF [11] scheduling, or non-preemptive sporadic tasks scheduled by an FP [4] or EDF policy [12]. RBAs, on the other hand, are often exact in more general cases, e.g., preemptive [5] or non-preemptive [7] periodic and sporadic tasks scheduled by global FP scheduling policies. However, they are known to have a very poor scalability [5]–[7], for example, they can handle limited period values (e.g., smaller than 20 [5]) or limited number of tasks (e.g., fewer than 7 [6]).

Recently, Nasri et al. [1,8]–[10] have introduced a new reachability-based response-time analysis, called *schedule-abstraction graph* (SAG), that surpasses the scalability limitations of the existing RBAs by at least three orders of magnitude (see [7,9]). However, as we will show in Sec. II, this analysis still suffers from state-space explosion when there are large uncertainties in the timing parameters of the input jobs/tasks.

Schedule-abstraction graph. SAG explores the space of possible decisions that a JLFP scheduling policy can take when dispatching a set of jobs on processing resources. This decision space is explored by building a graph whose vertices represent the *state* of the platform/resource *after the execution of a set of jobs* and whose edges represent possible scheduling decisions that evolve the system states. SAG has been designed for non-preemptive jobs [1,8]–[10], hence, a scheduling decision is to determine "a next job that can possibly be dispatched" on the platform after a state v_i. Dispatching of this job will change the platform state from v_i to v_j. An example job set along with its schedule are provided in Figs. 1(a) and 1(b), respectively. The graph made by the SAG for this job set is shown in Fig. 1(c). While building the graph, the response time of each job that is being dispatched on an edge is tracked and hence when the graph is fully constructed, the method outputs the smallest and largest response time of each job in all scenarios (edges) that involve that job.

To defer the state-space explosion, Nasri et al. [1] have introduced two main techniques: (i) powerful *interval-based abstractions* to aggregate similar system states (or equivalently, schedules that have a similar impact on the final system state), and (ii) *state-merging rules* to combine system states whose future can be explored together. These techniques allowed their solution to be at least 3000 times faster than other exact RTAs that are based on generic formal verification tools such as UPPAAL [7] and to scale to large system sizes.

Current limitation of SAG. Despite the current success in scalability, the schedule-abstraction-based analysis still faces one big fundamental limitation: *each edge can only include one single scheduling decision* (i.e., dispatching of one job) [1,8]–[10]. As a result, as soon as there are large uncertainties in the release time or execution time of the jobs in the input job set, the number of states generated by the SAG grows exponentially because the analysis will try to explore all (valid) combinations of ordering between jobs.



J_i	release time earliest	release time latest	BCET	WCET	deadline	priority
J_1	0	0	7	13	20	1 (high)
J_2	6	9	1	2	20	2
J_3	7	10	1	2	20	3
J_4	18	20	1	3	25	4
J_5	5	9	1	2	20	5 (low)

Fig. 1: An example showing the difference between the SAG constructed by the original analysis and our new POR analysis. (a) An example job set with release jitter. (b) Schedule showing the job set visually. (c) SAG constructed by the analysis from [1] for the given job priorities. (d) SAG constructed by our POR for the same system.

This paper. The goal of our work is to improve the scalability of the schedule-abstraction-based analysis by introducing *partial-order reduction* (POR) rules that allow combining multiple scheduling decisions on one edge and hence avoiding combinatorial exploration of all possible orderings between jobs in cases where there are large uncertainties.

As our goal is to demonstrate how to apply POR on SAG-based analyses, in this work we will focus on a simpler (yet NP-hard) schedulability analysis problem, i.e., analyzing a set of non-preemptive jobs (or periodic tasks) scheduled by a work-conserving JLFP scheduling policy on a uniprocessor platform. We show that our solution maintains the *exactness* of the original analysis [1] in terms of schedulability while reducing the runtime of the original analysis by *five orders of magnitude* and the number of explored states by 98%.

II. Motivation

When a given job set has timing uncertainties, e.g., due to release jitters or execution time variations of the jobs, the exact release or execution times are not known. Since an online JLFP scheduler takes its decisions by looking at the *released jobs* in the ready queue when the processor is available, depending on the order in which the jobs are released and the completion time of already running jobs (called *execution scenarios*), multiple schedules might be generated under different circumstances. A system is *schedulable* if no job misses its deadline under any schedule that the underlying scheduler can generate for those jobs at runtime.

The current SAG analysis by Nasri et al. [1] processes one scheduling decision at a time as can be seen on the label of the edges in Fig. 1(c). It starts from a idle system (state v_1) and then looks for all *next* possible scheduling decisions. Here, job J_1 has no release jitter, thus, as the first and only job in the ready queue in state v_1 and under a work-conserving scheduling policy, J_1 will always be dispatched before other jobs (hence, there is only one edge from v_1 to v_2). However, due to the non-deterministic execution time of J_1, the processor may become available for other jobs at any time from 7 to 13 (note that 7 is the best- and 13 is the worst-case execution time of J_1). The SAG abstracts these uncertainties in an *uncertainty interval* $[7, 13]$ on the state v_2 which means that the processor may *possibly* be available for other jobs at time 7 and will *certainly* be available at time 13.

Representing a system state using an uncertainty interval allows SAG to fairly reduce the number of states needed to keep track of different schedules. However, it also forces the analysis to work with uncertainties. Namely, now to decide how v_2 will change after the next scheduling decision, SAG must consider any job that can be *dispatched next* after this state. In our example, due to the release jitter, either J_2, J_3, or J_5 can be dispatched next (or can be released next), after which any of the two remaining jobs can be dispatched. For example, if J_5 is released at time 5 and J_2 and J_3 are released at time 9 and 8, respectively, and the execution time of J_1 is not more than 7, then the first job that will be dispatched by the scheduler is J_5 and then job J_3 and finally job J_2. Alternatively, we could see a scenario in which J_2 is dispatched first (e.g., when it is released at time 6 but J_3 and J_5 are released after time 7). The SAG captures all these scenarios (and possible orderings between the jobs) in the graph by adding more out-going edges to the system states when a different job ordering is possible (shown in Fig. 1(c)).

However, when there are large uncertainties in the timing parameters, the original analysis [1] which processes only one scheduling decision at a time, will face a state-space explosion due to the combinatorial increase in the number of possible orderings between jobs. Our key observation to improve performance of the SAG analysis is that *the exploration of many of these job execution orderings is not relevant to schedulability of the job set, as none of these orderings may lead to a deadline miss*. For example, the original SAG graph in Fig. 1(c), has 10 nodes and 14 edges. Yet, none of the possible job execution orderings of J_2, J_3, and J_5 considered in the graph may lead to a deadline miss (for any of these jobs or the future jobs, e.g., J_4). Ideally, we would want to skip over these three jobs without exploring every individual job execution ordering. By avoiding the exploration of such scenarios, we could get a performance gain in the analysis and have a much smaller graph like the one shown in Fig. 1(d).

III. Partial-order reduction

Our key idea is to identify subsets of jobs for which the combinatorial exploration of all orderings is *irrelevant* to the schedulability of the job set. Exploring these combinations is irrelevant when (i) all scenarios lead to the same system state and (ii) none of the jobs observe a deadline miss. Dispatching such jobs can be considered in a single step (that combines

all those scheduling decisions), which further defers the state-space explosion. The POR technique proposed in this work allows us to identify such job orderings and treat them as a *batch of scheduling decisions on a single edge in the SAG*.

Fig. 1(d) shows the SAG constructed using our proposed POR technique, where J_2, J_3, and J_5 are put on a single edge, removing the need to enumerate all possible scenarios between these jobs and shrinking the resulting schedule-abstraction graph. Note that the interval (i.e., label) recorded in v_9 in Fig. 1(c) is the same as the interval recorded in v_3 in Fig. 1(d), even though the latter did not explore all different scenarios.

In this work, we extend the SAG algorithm with the idea of POR sketched above. When expanding a state v_p in the SAG, the original algorithm by Nasri et al. [1] simply adds a new vertex for each *direct successor* (i.e., a job that can be dispatched after v_p before any other job). Instead, we perform POR to determine whether a set of future (i.e., not yet dispatched) jobs can be "reduced" to a single edge that encompasses all orderings of the original jobs without explicitly exploring all these orderings. We refer to the set of candidate jobs $\mathcal{J}^S$ that is being considered for the reduction to a single edge as the *candidate reduction set*. If the set of jobs meets the to-be-determined conditions for a valid POR, it is added to the graph using a single edge and vertex. If these conditions are not met, the graph is expanded with a new vertex for each direct successor as per the original SAG.

The original schedulability analysis by Nasri et al. [1] is an *exact schedulability analysis* for the problem being considered. Namely, any job set deemed schedulable (resp. unschedulable) by the analysis is indeed schedulable (resp. unschedulable). Hence, when reducing a set of jobs to one batch of scheduling decisions, the schedulability analysis should remain exact by including the information of the possible job execution orders *without exploring each order explicitly*.

The key to having an exact schedulability analysis with POR is that the reduction does not affect the analysis of the execution of jobs not contained in the reduction set. Therefore, to have an exact schedulability analysis using POR, we must maintain two properties: (P1) the schedulability analysis of all jobs not contained in the reduction set should remain exact, and (P2) the schedulability analysis of all jobs in the reduction set should remain exact. (P1) is maintained if the intervals with and without POR are the same (like v_9 in Fig. 1(c) and v_3 in Fig. 1(d)) and (P2) is maintained if there is no deadline miss when the jobs in the reduction set can have any ordering. Satisfying (P1) requires (P3): no job outside of a reduction set ever *interferes* with the reduction set. To satisfy (P2) we need (P3) and (P4): no deadline miss happens for jobs in the reduction set. We ensure (P3) by our way of iteratively generating the candidate reduction set, and (P4) by using a sufficient schedulability test.

We create a candidate reduction set $\mathcal{J}^S$ for state v_p in an iterative manner while ensuring there is no interfering job for $\mathcal{J}^S$ (P3). We call a job J_x that can execute between two arbitrary jobs in $\mathcal{J}^S$ but is itself not in $\mathcal{J}^S$ an *interfering job*. The conditions under which a job J_x can be an interfering job for $\mathcal{J}^S$ depend on the scheduling policy used for the job set under analysis. For example, under a work-conserving scheduling policy, a job J_x can interfere with $\mathcal{J}^S$ if it can be released before or during a time instant where the processor is idle between the execution of two jobs in $\mathcal{J}^S$. Under a priority-driven policy, a job J_x can interfere with $\mathcal{J}^S$ if it can be released before a lower-priority job in $\mathcal{J}^S$ starts to execute.

A candidate reduction set $\mathcal{J}^S$ is initialized with the direct successors of v_p, since these jobs interfere with each other. If (P3) holds for $\mathcal{J}^S$, we can compute an accurate uncertainty interval (i.e., label) for the vertex that receives a set of scheduling decisions (for the reduction set), e.g., for v_3 in Fig. 1(d), such that this interval guarantees that the schedulability analysis of all jobs not contained in $\mathcal{J}^S$ remains exact. If (P3) does not hold, i.e., there is an interfering job J_x for $\mathcal{J}^S$, we add J_x to $\mathcal{J}^S$ and repeat the process of checking for interfering jobs.

In order to efficiently determine whether a deadline miss may happen within the jobs in a candidate reduction set (and therefore whether (P4) holds), we use a fixed-point-iteration based sufficient schedulability test that works on a set of jobs. If the reduction set passes the sufficient test, i.e., is certainly schedulable, we reduce the set to one batch of scheduling decisions in the SAG. Otherwise, we explore all possible job execution orderings as per the original analysis [1].

IV. Empirical evaluation

To evaluate our POR solution in terms of the state-space reduction and runtime improvement in comparison to the original SAG analysis of Nasri et al. [1] (referred to as 'original'), we considered the problem of analyzing schedulability of a set of non-preemptive periodic task sets with implicit deadlines on a uniprocessor platform scheduled by a fixed-priority scheduling policy with rate-monotonic priorities.

Metrics. Our performance metrics are as follows. **State-reduction ratio** is defined as $1 - N^P/N^O$, where N^P is the number of states explored by our POR analysis for a job set $\mathcal{J}$ and N^O is the number of states explored by the *original* schedule-abstraction-based analysis for the same job set. Namely, the closer the state reduction ratio is to 1, the more states the analysis is able to *remove* from the state-space in comparison to the states explored by the original analysis. **Speedup** of our analysis on a job set $\mathcal{J}$ is defined as the CPU time required by the original analysis to analyze $\mathcal{J}$ divided by the CPU time required by our analysis for $\mathcal{J}$.

Experimental setup. The experiments were performed on a Dutch national cluster called SURFsara Cartesius cluster, with nodes equipped with two Intel Xeon E5-2690 v3 processors clocked at 2.6 GHz and 64 GB RAM. The analysis was implemented as a single-threaded C++ program.

To construct random synthetic task sets with n periodic tasks and a total utilization of U, we first use the method of Emberson et al. [13] to generate periods following a log-uniform distribution in the range $[10, 100]$ms with a granularity of 5ms. Next, we generate n task-utilization values that sum to target utilization U using the RandFixedSum algorithm.The periods and task-utilization values are combined to obtain the

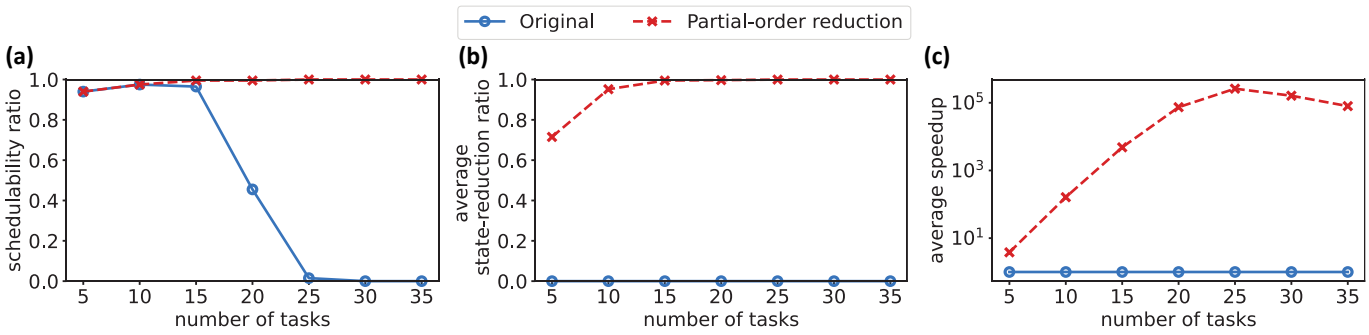


Fig. 2: Experimental results for synthetic task sets. (a) Schedulability ratio. (b) State-reduction ratio. (c) Speedup.

worst-case execution time (WCET) of each task. The best-case execution time (BCET) was set to be 0 for all tasks. Finally, we assumed a release jitter of 100µs for each job of a periodic task in the hyperperiod. In the following experiments, we fixed the utilization to 0.3 and varied the number of tasks per task set to evaluate the impact of the number of tasks (scalability) on the proposed solution. Choosing $U = 0.3$ ensures having sufficient number of schedulable task sets. We expect the effectiveness of our POR solution to decrease for high utilizations, as the increase in unschedulable task sets will mean a decrease in the number of reduction sets satisfying (P4). Finally, for each parameter value, we generated 200 random task sets.

A task set was deemed unschedulable as soon as either an execution scenario containing a deadline miss was encountered, or when a timeout of four hours was reached. We continued the analysis until either the end of the hyperperiod or a timeout was reached. Task sets with more than 50,000 jobs per hyperperiod were discarded to limit the runtimes (as otherwise we would not be able to compare our solution against SAG that was susceptible to state-space explosions).

Results. Fig. 2(a) shows a large difference between the schedulability ratio of the original analysis and the POR analysis. The difference is caused by the original analysis reaching the timeout of four hours, thus reporting task sets as unschedulable, whereas the POR analysis was able to explore the entire state-space and conclude these same task sets were schedulable within the time limit. This shows that POR allows task sets with more uncertainties and more tasks to be analyzed before the timeout is reached. Fig. 2(b) shows that the state-reduction ratio for POR increases as n increases. The average state-reduction ratio is 98.53%. This means that on average, POR is able to remove the vast majority of states from the state-space compared to the state-space explored by the original SAG. Fig. 2(c) shows that on average, POR provides a significant speedup over the original analysis for all values of n, with an average speedup of 1.12×10^5.

V. Conclusion

In this work, we explored the possibility of applying partial-order reduction (POR) on one of the recent and successful reachability-based response-time analyses called schedule-abstraction graph (SAG). We showed how to avoid a combinatorial exploration of multiple scheduling decisions of individual jobs by aggregating those decisions when the internal order of those decisions has no impact on the schedulability. Our empirical evaluation shows that our POR solution was able to reduce the runtime by five orders of magnitude and the number of explored states by 98% compared to the original SAG analysis of Nasri et al. [1]. Our work is a promising proof of concept showing that POR can be used in the future to improve the scalability of reachability-based analyses for a wider variety of scheduling policies and systems, including multiprocessor platforms.

Acknowledgment

This work was carried out on the Dutch national e-infrastructure with the support of SURF Cooperative.

References

[1] M. Nasri and B. B. Brandenburg, "An Exact and Sustainable Analysis of Non-preemptive Scheduling," in *RTSS*, 2017, pp. 12–23.
[2] P. Ekberg, "Rate-Monotonic Schedulability of Implicit-Deadline Tasks is NP-hard Beyond Liu and Layland's Bound," in *RTSS*, 2020, pp. 308–318.
[3] 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.
[4] R. Davis, A. Burns, R. Bril, and J. Lukkien, "Controller Area Network (CAN) schedulability analysis : refuted, revisited and revised," *Real-Time Systems*, vol. 35, no. 3, pp. 239–272, 2007.
[5] N. Guan, Z. Gu, Q. Deng, S. Gao, and G. Yu, "Exact Schedulability Analysis for Static-Priority Global Multiprocessor Scheduling Using Model-Checking," in *SEUS*, 2007, p. 263–272.
[6] Y. Sun and G. Lipari, "A Pre-Order Relation for Exact Schedulability Test of Sporadic Tasks on Multiprocessor Global Fixed-Priority Scheduling," *Real-Time Syst.*, vol. 52, no. 3, p. 323–355, 2016.
[7] B. Yalcinkaya, M. Nasri, and B. B. Brandenburg, "An Exact Schedulability Test for Non-Preemptive Self-Suspending Real-Time Tasks," in *DATE*, 2019, pp. 1228–1233.
[8] M. Nasri, G. Nelissen, and B. B. Brandenburg, "A Response-Time Analysis for Non-Preemptive Job Sets under Global Scheduling," in *ECRTS*, 2018, pp. 9:1–9:23.
[9] M. Nasri, G. Nelissen, and B. B. Brandenburg, "Response-time analysis of limited-preemptive parallel DAG tasks under global scheduling," in *ECRTS*, 2019, pp. 21:1–21:23.
[10] S. Nogd, G. Nelissen, M. Nasri, and B. B. Brandenburg, "Response-Time Analysis for Non-Preemptive Global Scheduling with FIFO Spin Locks," in *RTSS*, 2020, pp. 115–127.
[11] S. Baruah, A. Mok, and L. Rosier, "Preemptively scheduling hard-real-time sporadic tasks on one processor," in *RTSS*, 1990, pp. 182–190.
[12] K. Jeffay, D. Stanat, and C. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *RTSS*, 1991, pp. 129–139.
[13] P. Emberson, R. Stafford, and R. I. Davis, "Techniques for the synthesis of multiprocessor tasksets," in *WATERS*, 2010, pp. 6–11.