Schedulability Analysis of Real-time Tasks with Precedence Constraints

Rongfei Xu, Li Zhang
School of Computer Science and Engineering
Beihang University
Beijing, China

Ning Ge
School of Software
Beihang University
Beijing, China
gening@buaa.edu.cn

Xavier Blanc
LaBRI, UMR5800
University of Bordeaux, Bordeaux INP, CNRS
Talence, France

***Abstract*—The timing requirements of real-time systems can be guaranteed by the well-designed scheduling. The analysis of such scheduling inputs an abstract task model of the system and outputs a diagnostic regarding the practicability of the timing requirements. Task models have evolved from periodic models to more sophisticated graph-based ones, among which the digraph real-time (DRT) task model is the most applicable because of its good expressiveness and analysis efficiency. However, the DRT model can't support the precedence constraints within or between tasks. In this paper, we propose a new task model, called the DRTPC model, that extends the DRT model to support the precedence constraint. Further, based on our model, we present a uniprocessor schedulability analysis algorithm for the static priority scheduling, and introduce an optimization technique to improve the analysis efficiency. Our experiments show that, despite the high computational complexity of the problem, our approach scales very well for large sets of tasks with precedence constraints.**

***Index Terms*—real-time system, schedulability analysis, task model, precedence constraint, static priority scheduling**

I. Introduction

The timing requirements of real-time systems can be guaranteed by the well-designed scheduling. The analysis of such scheduling is to assess the schedulability regarding the practicability of the timing requirements [1], i.e. the system's tasks can complete by the deadline. In the design of real-time systems, an abstract task model is usually used as an input for the schedulability analysis, and specifies the tasks' timing constraints (duration, start constraint, etc.) [2]. The precedence constraints within or between tasks, which commonly exist in real-time systems [3], [4], directly affect the schedulability and need to be concerned in the task model.

Let us take a simple example of a robot controller system to explain the precedence constraint. This system consists of three periodic tasks: the *navigation* task is to go to the destination by continuous movements and to avoid the obstacles when finding them, the *detect* task is to detect the location of the obstacles, and the *balance* task is to keep the balance of the robot. The *navigation* task needs the obstacle's location to do the job of finding the obstacle (*FO*). The location is collected by the job of receiving the location of an obstacle (*RL*) in the *detect* task. The job of computing the adjustment for a balance (*CA*) in the *balance* task needs the data from the inclinometer, which is collected by the job of reading the inclinometer (*RI*) in the same task. Finally, the adjustment is used by the job of controlling balance (CB) to balance the robot. There are then three precedence constraints: *RL* precedes *FO*, *RI* precedes *CA*, and *CA* precedes *CB*.

Since the well-known Liu and Layland task model [5] appeared, a large number of task models, ranging from the relatively simple periodic and sporadic ones to the more complex graph-based ones, have been proposed [2]. There is a contradicting goal of expressiveness and analysis efficiency for these task models. For example, the Petri net [6] and the timed automata [7] are powerful to allow accurate modeling and easy to support the precedence constraint, but their analyses are based on the model checking, which can result in the exponential computational complexity. On the other hand, some works have specified the precedence constraint by extending the tractable task models, such as the recurring real-time task model [8], the sporadic task model [9], the dynamic offsets task model [10], [11], etc. Although these extended task models have a relatively good analysis efficiency, they are limited to the specific tasks and not for general-purpose ones. Therefore, the existing task models supporting the precedence constraint are inadequate. Due to good expressiveness and analysis efficiency, the currently proposed digraph real-time (DRT) task model [12] is promising according to a thorough survey [2]. Specifically, the DRT model can specify both the sporadic and periodic tasks, and is tractable (i.e. in pseudo-polynomial time) to be analyzed for large sets of tasks. However, the DRT model is based on the assumption that the tasks are independent of each other, so it is not yet to support the precedence constraint.

In this paper, we extend the DRT model by specifying the precedence constraint to get a new task model called DRTPC (Digraph Real-Time task model with Precedence Constraint). Further, we present a uniprocessor schedulability analysis algorithm for the static priority scheduling in the DRTPC model. This technique is capable of analyzing the schedulability by considering the interferences caused by both the priority-based preemption and the precedence constraint. In addition, we introduce an optimization technique for the schedulability

This paper is supported by the National Natural Science Foundations of China (No. 61672078 and No. 61732019)
DOI:10.18293/SEKE2018-116

analysis to improve its efficiency. Our experiments show that, the proposed approach scales very well for large sets of tasks with precedence constraints.

In the remainder of this paper, we first discuss the related works in Sect. II. Then, we describe the DRTPC model in Sect. III and present the schedulability analysis algorithm for DRTPC in Sect. IV. Finally, the efficiency and scalability of our approach are evaluated in Sect. V, and Sect. VI gives some concluding remarks and perspectives.

II. Related Works

In the past decades, the abstract task models that specify the tasks' timing constraints have been studied intensively in the real-time scheduling [1], such as the multiframe (MF) task model [13], generalized multiframe (GMF) task model [14], recurring real-time (RRT) task model [15], etc. The DRT model [12] considered in this paper is a generalization of the above models. Specifically, the DRT model can specify the branching and loop structures in real-time systems, which makes it capable of modeling both the sporadic and periodic tasks. Besides, some efficient methods of schedulability analysis have been proposed for the DRT model [16]. However, the DRT model can only support the independent tasks.

Currently, some works have been done to concern the dependence between tasks in the DRT model, such dependencies include three classes: inter-release time constraint [17], shared resource [18], [19] and synchronization [20], [21]. Specifically, the works [17] and [20] extended the edge in the DRT model to specify the global inter-release separation constraint and the synchronous execution respectively. The works [18], [19] and [21] extended the vertex in the DRT model to specify the maximal duration of resource access, the semaphore that guards the shared resource, and the synchronization operation respectively. However, it lacks an approach to concern the dependence caused by the precedence constraint, which is considered in this paper for the DRT model.

In the context of formal modeling and analysis, the precedence constraint is supported by two classes of works. The first class is based on the more expressive task models. For example, the work [7] specified the timed automata to deal with the precedence and resource constraints between the real-time tasks; the work [6] identified the precedence constraint properties of Petri net and tested whether it was feasible to execute a workflow with the specified temporal constraints. Due to the analysis complexity of automata or Petri net, the above-mentioned works suffer from the state space-explosion problem. The second class extends a simple task model to support the precedence constraint. For example, the work [9] considered the real-time system with the implicit precedence constraints between the intra-task jobs based on the sporadic task model; the work [8] represented the recurrent precedence-constrained tasks to be executed on multiprocessor platforms, where each recurrent task was modeled by a directed acyclic graph (DAG); the works [10], [11] addressed the schedulability analysis of the tasks with precedence relations in the distributed real-time systems based on the model of tasks with dynamic offsets, which was specific for the distributed systems; the work [22] proposed an approach to scheduling the tasks with pipeline precedence constraints in the distributed real-time systems, which were described by the directed acyclic graph (DAG). Due to limited expressiveness of the task models to be extended, the above works can't support the general purpose tasks. Therefore, it is necessary to extend the DRT model to support the precedence constraint.

III. Task Model

In this section, we first introduce the DRT model, then define our DRTPC model to support the precedence constraint based on the DRT model, and explain its semantics.

A. DRT Model

A real-time system is usually designed as a set of tasks, each of which consists of a sequence of functional blocks (called jobs here) [23]. A task model characterizes a task by the execution sequences and the timing constraints of its jobs. According to the definition of DRT model in [12], each task is characterized by a directed graph G(T). The vertices of G(T) represent the jobs in the task. Each vertex is labeled by an ordered pair of (*WCET*, *RD*), which represents the worst-case execution time (WCET) demand and the relative deadline of the corresponding job respectively. The directed edges of G(T) represent the orders (from start to end) in which the jobs are released. Each edge is labeled by the inter-release interval time between two jobs. In such graph G(T), a vertice may have multiple edges or a loop edge, which makes the DRT model can specify the branching and loop structures.

Example 1: For the example of the robot controller, it consists of three tasks. For the *navigation* task, it includes three jobs (go to the destination *GD*, find an obstacle *FO* and avoid the obstacle *AO*) and four release orders (from *FO* to *AO*, from *AO* to *GD*, from *GD* to *FO* and to itself). For the *detect* task, it includes two jobs (send a detection *SD* and receive the location of the obstacle *RL*) and two release orders (from *SD* to *RL*, and from *RL* to *SD*). For the *balance* task, it includes three jobs (read the inclinometer *RI*, compute the adjustment *CA*, and control the balance *CB*) and three release orders (from *RI* to *CA*, from *CA* to *CB*, and from *CB* to *RI*). If we don't consider the precedence constraints, the DRT model of this example is characterized as shown in Fig. 1. For example, the job *AO* in the *navigation* task is set as having a WCET demand of 1 time unit and a relative deadline of 3 time units. The job *AO* is set as being released at 35 time units later than the release time of *FO*.

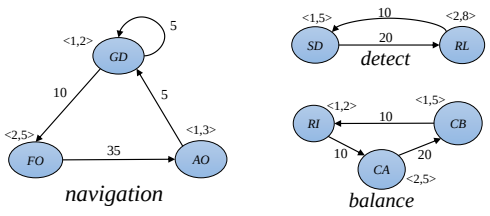


Fig. 1. DRT model of the robot controller.

B. DRTPC Model

The DRTPC model is defined based on the DRT model to support the precedence constraint, which constrains that the execution of a job shouldn't start until all of its precedent jobs (if exist) are finished. In the DRTPC model, each task is expressed by a directed graph $G'(T)$, which describes the timing constraints of each job and the precedence constraints between jobs in this task. Each vertex of $G'(T)$ is labeled by a triple (*PJ*, *WCET*, *RD*), which represents a set of precedent jobs, the worst-case execution time, and the relative deadline of the corresponding job respectively. The directed edge of $G'(T)$ has the same meaning as the DRT model, i.e. the inter-release interval time between jobs. It should be noted that the precedent job of a job may be in the same task model or in a different task model, which is called intra-task or inter-task precedence constraint respectively.

Example 2: The DRTPC model of the robot controller considering the precedence constraints is shown in Fig. 2.

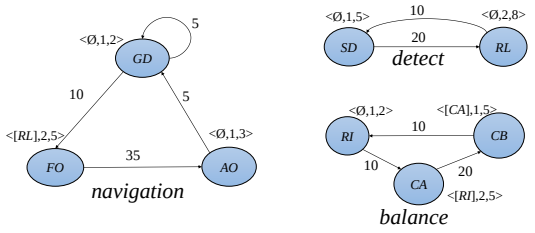


Fig. 2. DRTPC model of the robot controller.

The semantics of the DRTPC model is defined based on the execution paths generated by the task model. Such execution path is specified by a job sequence, where each job is considered from two parts: timing constraint and precedence constraint. Formally, we use a 3-tuple (*RT*, *DT*, *AD*) to denote a job that is released at the time *RT*, with the duration *DT* and the absolute deadline *AD*.

A job sequence is said to be valid, if and only if an arbitrary job in the sequence satisfies the following two conditions:

Condition 1: For the timing constraint, it should satisfy the three sub-conditions [12]: The duration is equal to the WCET of the job; the absolute deadline is equal to the release time plus the relative deadline of the job; the time between the release time of the job and that of an other job is greater than the inter-release interval time between them.

Condition 2: For the precedence constraint, it should satisfy the two sub-conditions: For the intra-task precedence constraint, all of the precedent jobs (if exist) of the job should be included in the same job sequence; For the inter-task precedence constraint, all of the precedent jobs (if exist) of the job should be included in the job sequences of other tasks, which contain these precedent jobs.

Example 3: We take the task model in Fig. 2 as an example to illustrate the valid job sequences within one period. For the *balance* task, the job sequence of (*RI*, *CA*, *CB*) with the timing constraints of (0, 1, 2), (10, 2, 15), and (30, 1, 35) respectively is valid, because the precedent job of *CA* (i.e. *RI*) is included in this job sequence. For the *detect* task, a valid job sequence is (*SD*, *RL*) with the timing constraints of (0, 1, 5) and (20, 2, 28) respectively. For the *navigation* task, the job sequence of (*FO*, *AO*) with the timing constraints of (0, 2, 5) and (35, 1, 38) respectively is valid, because the precedent job of *FO* (i.e. *RL*) is included in the job sequence of the *detect* task.

In the next section, we will take an example of the three valid job sequences to introduce the schedulability analysis for the DRTPC model.

IV. Schedulability Analysis

In this section, we present the schedulability analysis for the static priority scheduling in the DRTPC model. For such scheduling, each task in the real-time system is assigned a unique priority. The jobs have the same priority as their task. For each job in the execution paths of the task, the job can be executed only if no job with a higher priority exists in the system. When all jobs in the execution paths meet their deadline after the scheduling, this task is considered as schedulable.

As the DRTPC model (DRTPC in short) is based on the DRT model (DRT in short), we first introduce the schedulability analysis for the static priority scheduling in the DRT, which only concerns the interference on the scheduling caused by the priority-based preemption. Then, we propose our schedulability analysis algorithm for the DRTPC, which extra concerns the interference caused by the precedence constraint.

A. Schedulability Analysis for DRT

The schedulability analysis for the DRT is based on evaluating the request function [16], which represents the maximum accumulated workload of all jobs that the job sequence may generate during a time interval. The request function is defined as follows.

Definition 1 (Request Function [16]). For the job sequence $\sigma = (v_0, v_1, ..., v_n)$ of an arbitrary execution path π in the task model, its request function before time t is defined as

$$rf_\pi(t) = max(dt(\pi')|\pi' \text{ is prefix of } \pi \text{ and } g(\pi') < t) \quad (1)$$

where $dt(\pi) = \sum_{i=0}^{n} dt(v_i)$, $g(\pi) = \sum_{i=1}^{n} g(v_{i-1}, v_i)$, $dt(v_i)$ is the duration of job v_i, and $g(v_{i-1},v_i)$ is the inter-release interval time between job v_{i-1} and job v_i.

The schedulability of a job with respect to a set of interfering tasks is specified based on the request function. To analyze the schedulability, we first define Π_T as the set of all execution paths for an arbitrary task T in the task model. Then, for a set of tasks $\Gamma = (T_1, T_2, \ldots, T_n)$, let $\Pi(\Gamma) = \Pi_{T1} \times \Pi_{T2} \times \ldots \times \Pi_{Tn}$ be the set of all path combinations, namely $\Pi(\Gamma) = \{ (\pi_1, ..., \pi_n) \mid \pi_1 \in \Pi_{T1}, \ldots, \pi_n \in \Pi_{Tn} \}$. Finally, the schedulability is judged based on the following theorem.

Theorem 1 ([16]): A job with duration dt and absolute deadline ad is schedulable under a set of interfering tasks Γ if and only if

$$\forall(\pi_1, ..., \pi_n) \in \Pi(\Gamma) : \exists t < ad : dt + \sum_{T_i \in \Gamma} rf_{\pi_i}(t) \leq t \quad (2)$$

This theorem shows that, a necessary and sufficient condition for the schedulability of a job is that during its release time and (absolute) deadline, there exists a time instant t at which this job and all of the interfering jobs released before t are finished. For the static priority scheduling in the DRT, the interfering jobs are the jobs with a higher priority.

B. Schedulability Analysis for DRTPC

As the DRTPC extends the DRT to support the precedence constraint, the interference caused by such precedence constraint needs to be extra considered in the schedulability analysis for the DRTPC. We first take an example of the execution of the robot controller to present the interferences in the DRTPC.

Example 4: We give a scenario that the tasks in the robot controller have a priority relationship as $P(balance) > P(detect) > P(navigation)$. The sample execution of the tasks with the execution paths in *Example 3* is shown in Fig.3.

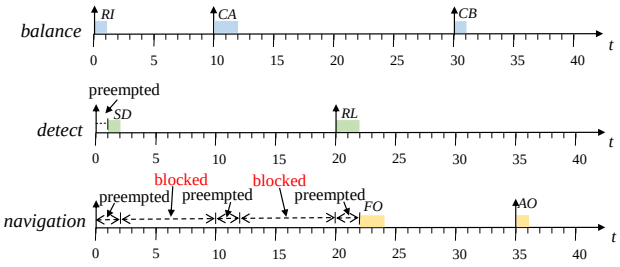


Fig. 3. A sample execution of the execution paths in *Example 3*

It can be found from the example that there are two interferences during the scheduling in the DRTPC: preemption interference caused by the jobs with a higher priority and block interference caused by the precedent jobs. The schedulability analysis for the DRT only considers the interference workload caused by the preemption. Here, we additionally study the workload resulting from the precedence constraint.

During the scheduling, if the precedent jobs of a job haven't been all finished, then this job will be blocked. The block relationship between a job j and its precedent job k is summarized as the following three cases:

- The finish time of job k is before the release time of job j. In this case, job j will not be blocked by such a precedent job.
- The finish time of job k is after the absolute deadline of job j. In this case, job j will be blocked until its deadline, which makes the block time infinite.
- The finish time of job k is between the release time and the absolute deadline of job j. This case includes two sub-cases:
  - If the release time of job k is before that of job j, then the block time is the time interval between the release time of job j and the finish time of job k.
  - If the release time of job k is after that of job j, then the block time is the time interval between the release time and finish time of job k.

Then, the workload of an arbitrary job under a set of interfering tasks in the DRTPC consists of three parts: duration of the job, preemption interference, and block interference. Based on *Theorem 1*, an arbitrary job with duration dt and absolute deadline ad is schedulable under a set of interfering tasks Γ if and only if

$$\forall(\pi_1, ..., \pi_n) \in \Pi(\Gamma) : \exists t < ad : \\ dt + \sum_{T_i \in \Gamma} rf\_P_{\pi_i}(t) + \sum_{T_i \in \Gamma} rf\_B_{\pi_i}(t) \leq t \quad (3)$$

Where $\sum_{T_i \in \Gamma} rf\_P_{\pi_i}(t)$ and $\sum_{T_i \in \Gamma} rf\_B_{\pi_i}(t)$ represent the accumulated workload caused by the preemption interference and the block interference respectively. This formula shows that, a necessary and sufficient condition for the schedulability of a job in the DRTPC is that during its release time and (absolute) deadline, there exists a time instant t at which this job and all of its precedent jobs together with the jobs with a higher priority released before t are finished.

Finally, we introduce the schedulability analysis procedure for a task T with an arbitrary execution path JS (i.e. a job sequence), which is shown in Fig.4.

function schedulability_analysis (*T*, *JS*)
1: **for each** job $j \in JS$ of T **do**
2: compute $\sum_{T_i \in \Gamma} rf\_P_{\pi_i}(t)$ for j
3: compute $\sum_{T_i \in \Gamma} rf\_B_{\pi_i}(t)$ for j
4: **if** $\forall(\pi_1, ..., \pi_n) \in \Pi(\Gamma)$:
5: $\exists t < ad: dt + \sum_{T_i \in \Gamma} rf\_P_{\pi_i}(t) + \sum_{T_i \in \Gamma} rf\_B_{\pi_i}(t) \leq t$ **then**
6: isSchedulable(j) = *true*
7: **else**
8: isSchedulable(j) = *false*
9: **end if**
10: **end for**
11: **for each** job $k \in JS$ *of* T **do**
12: **if** isSchedulable(k) == *false* **then**
13: **return false**
14: **end if**
15: **end for**
16: **return true**

Fig. 4. Procedure of schedulability analysis.

Procedure 1: First, compute the accumulated workload caused by the preemption interference $\sum_{T_i \in \Gamma} rf\_P_{\pi_i}(t)$ and the block interference $\sum_{T_i \in \Gamma} rf\_B_{\pi_i}(t)$ for every job j in the job sequence JS of the task T (L. 2, 3). Then, sum the duration of the job j, the $\sum_{T_i \in \Gamma} rf\_P_{\pi_i}(t)$, and the $\sum_{T_i \in \Gamma} rf\_B_{\pi_i}(t)$ to check whether the condition of the request function in (3) is satisfied (L. 4). If there exists some time t that makes the condition satisfied, then the job j is considered as schedulable (L. 5). Only all of the jobs in JS are schedulable can the execution path JS be considered as schedulable (L. 10-15).

C. Optimization for Analysis

The schedulability analysis proposed in Sect. IV-B considers all path combinations of the interfering tasks Γ, which leads to a high computational complexity. Here, we optimize this

issue by checking each path π_i (i.e. job sequence) in the path combinations Π_Γ before the schedulability analysis, to distinguish the path combinations that must be tested.

The check follows two principles: if the path π_i is checked to make the task T (to be analyzed) unschedulable, then all of the path combinations needn't be analyzed anymore, and this task is considered as unschedulable directly; if the job in path π_i, which is a precedent job of some job in the job sequence of task T, is checked to cause no block, then remove this precedence constraint when analyzing the accumulated workload caused by the precedence constraints. The check only involves such jobs in path π_i that are the precedent jobs of some jobs in the job sequence of task T. Here, we assume an arbitrary job of a job sequence of task T and its precedent job in path π_i as the job j and the job k respectively. Then, the condition of removing the precedence constraint is that the priority of k is higher than that of j, and the release time of k is before that of j; the condition of unschedulable is that the priority of k is lower than that of j, and the release time of k is after that of j.

V. Evaluation

It has been proven that the DRT method can have the pseudo-polynomial complexity and good expressiveness comparing with existing methods. Here, we only need to evaluate that whether our based-DRT approach also has a tractable complexity. We first analyze the efficiency and scalability to evaluate whether it is practical to be used for large sets of tasks with precedence constraint. Besides, we also investigate the relationship between the schedulability and the number of precedence constraints in the task set, which is helpful to design the precedence constraints in real-time systems.

A. Experimental Setup

We implement our approach using the Javascript programming language running on a standard desktop computer with the 3.3GHz CPU and 8 GB RAM. For the random task set generation, we consider three types of tasks (namely small, medium, and large tasks) referring to [16], which have the parameter ranges as shown in Table I. For each task, one of the three types is randomly selected, then the task parameters are chosen randomly from the corresponding intervals.

TABLE I
Task Parameter Ranges

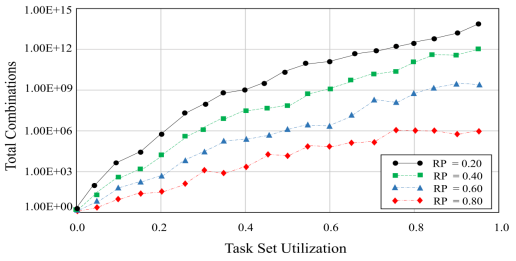
Task Type	Small	Medium	Large
Vertices	[3,5]	[5,9]	[7,13]
Branching degree	[1,3]	[1,4]	[1,5]
p	[50,100]	[100,200]	[200,400]
e	[1,2]	[1,4]	[1,8]
d	[25,100]	[50,200]	[100,400]

The workload generated by the tasks (instead of their numbers) has a direct relationship to the experimental results. Here we use the utilization to represent such workload, which is defined as the ratio of the sum of duration over the sum of inter-release separation time in the tasks [12]. Our experiment is implemented under a given task set utilization. In order to generate a task set with a desired utilization, random tasks are generated and added to the task set until the desired utilization is achieved. Besides, we use the ratio of precedence (RP) to represent the ratio of the jobs with the precedence constraint over the whole jobs in the task set, and use the acceptance ratio to represent the ratio of the schedulable tasks over the whole tasks in this experiment.

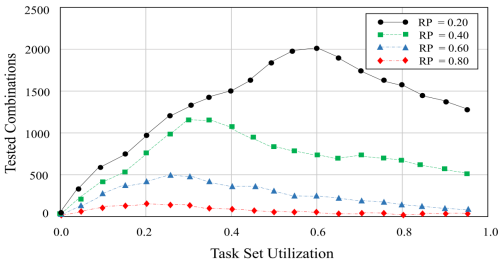
B. Experimental Results

(1) *Efficiency and scalability*

First, we explore how much our optimization can help in the complexity reduction. For four representative ratios of precedence in the task set, the comparison of the total path combinations and the combinations that must be tested after the optimization is depicted in Fig. 5. As seen, the reduction obtained by the optimization is considerable compared to the number of total combinations. It is also observed that the task set with a higher ratio of precedence (RP) has the fewer total combinations, as well as the tested combinations. This is because a larger number of precedence constraints in the task set makes fewer paths (i.e. job sequences) valid (see *valid job sequence* in Sect. III-B).



A. Total combinations



B. Tested combinations

Fig. 5. Reduction of the path combinations need to be tested

Then, we evaluate the efficiency and scalability of our approach through varying the ratio of precedence in the task set with a step of 0.05. Figure 6 shows the comparison of average run-time for the two analyses (with and without the optimization) under the task set utilization of 0.5. As seen, increasing the ratio of precedence in the task set causes that the analysis without the optimization becomes very lengthy. This is because that although the higher ratio of precedence means the fewer valid path combinations, the path combinations

with more precedence constraints can sharply increase the computation. In contrast, the analysis with the optimization has a good efficiency, and scales very well with the increasing ratio of precedence. This is because that with the ratio of precedence increasing, more and more tasks can directly be checked as unschedulable in advance, and do not cost the analysis run-time anymore.

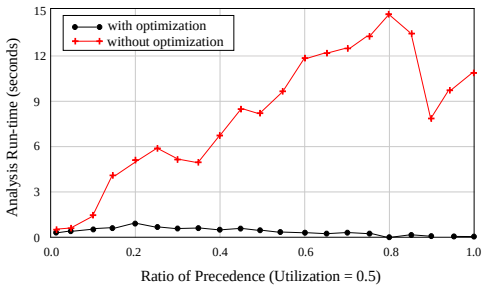


Fig. 6. Average run-time of the proposed methods

(2) *Schedulability*

Here, we investigate the relationship between the schedulability and the ratio of precedence in the task set. We find that a high ratio of precedence makes most of the tasks unschedulable, so we only present the results of four low ratios of precedence as shown in Fig. 7, i.e. RP = 0.00, 0.05, 0.10 and 0.20. As seen, with the utilization increasing, the task set under the four ratios of precedence all exhibit a lower and lower acceptance ratio. This is because that the higher utilization makes the tasks more difficult to be arranged. It is also observed that a higher ratio of precedence causes the decline curve of acceptance ratio steeper. This is because more precedence constraints in the task set lead to more blocks, which makes more tasks unschedulable.

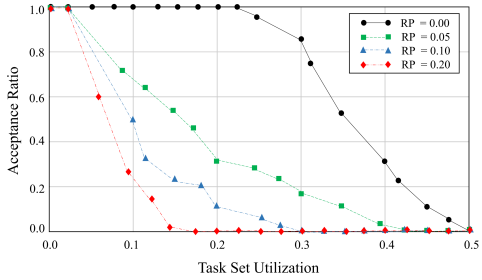


Fig. 7. Schedulability under different ratios of precedence

VI. Conclusion and Perspectives

With real-time systems are becoming more and more complex, the precedence constraints between or within their tasks directly impact the schedulability. In this paper, we propose an extension of the DRT task model to specify the precedence constraints, then propose a uniprocessor schedulability analysis algorithm for the static priority scheduling in our model. In addition, we introduce an optimization method for the analysis to improve its efficiency. Our experiments show that, the proposed approach can scale well for large sets of tasks with precedence constraint. Except for the precedence constraint, there are various constraints between the tasks affecting their schedulability, next we will extend our approach to more constraints. Besides, we will also extend our approach to support the multiprocessor in the future.

References

[1] L. Sha, T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, and A. K. Mok, "Real time scheduling theory: A historical perspective," *Real-time systems*, vol. 28, no. 2-3, pp. 101–155, 2004.

[2] M. Stigge and W. Yi, "Graph-based models for real-time workload: a survey," *Real-time systems*, vol. 51, no. 5, pp. 602–636, 2015.

[3] J. Schlatow and R. Ernst, "Response-time analysis for task chains with complex precedence and blocking relations," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, p. 172, 2017.

[4] D. Prot and O. Bellenguez-Morineau, "How the structure of precedence constraints may change the complexity class of scheduling problems," *arXiv preprint arXiv:1510.04833*, 2015.

[5] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[6] N. R. Adam, V. Atluri, and W.-K. Huang, "Modeling and analysis of workflows using petri nets," *Journal of Intelligent Information Systems*, vol. 10, no. 2, pp. 131–158, 1998.

[7] E. Fersman and W. Yi, "A generic approach to schedulability analysis of real-time tasks." *Nordic Journal of Computing*, vol. 11, no. 2, pp. 129–147, 2004.

[8] S. Baruah, V. Bonifaci, A. Marchettispaccamela, L. Stougie, and A. Wiese, "A generalized parallel task model for recurrent real-time processes," in *IEEE Real-Time Systems Symposium*, 2012, pp. 63–72.

[9] T. P. Baker and S. K. Baruah, "An analysis of global edf schedulability for arbitrary-deadline sporadic task systems," *Real-Time Systems*, vol. 43, no. 1, pp. 3–24, 2009.

[10] M. G. Harbour, "Exploiting precedence relations in the schedulability analysis of distributed real-time systems," in *Real-Time Systems Symposium, 1999. Proceedings. the IEEE*, 1999, pp. 328–339.

[11] R. Pellizzoni and G. Lipari, "Improved schedulability analysis of real-time transactions with earliest deadline scheduling," in *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS*, 2005, pp. 66–75.

[12] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "The digraph real-time task model," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2011 17th IEEE*. IEEE, 2011, pp. 71–80.

[13] A. K. Mok and D. Chen, "A multiframe model for real-time tasks," in *Proc. Real-Time Systems Symposium, Dec*, 1996, pp. 22–29.

[14] S. Baruah, D. Chen, S. Gorinsky, and A. Mok, "Generalized multiframe tasks," *Real-Time Systems*, vol. 17, no. 1, pp. 5–22, 1999.

[15] S. K. Baruah, "Dynamic- and static-priority scheduling of recurring real-time tasks," *Real-Time Systems*, vol. 24, no. 1, pp. 93–128, 2003.

[16] M. Stigge and W. Yi, "Combinatorial abstraction refinement for feasibility analysis," *Real-Time Systems*, vol. 51, no. 6, pp. 1–36, 2013.

[17] M. Stigge, P. Ekberg, N. Guan, and W. Yi, "On the tractability of digraph-based task models," in *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*. IEEE, 2011, pp. 162–171.

[18] N. Guan, P. Ekberg, M. Stigge, and W. Yi, "Resource sharing protocols for real-time task graph systems," in *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*. IEEE, 2011, pp. 272–281.

[19] Y. Zhuo, "Static priority schedulability analysis of graph-based real-time task models with resource sharing," 2014.

[20] M. Mohaqeqi, J. Abdullah, N. Guan, and W. Yi, "Schedulability analysis of synchronous digraph real-time tasks," in *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*. IEEE, 2016, pp. 176–186.

[21] N. Guan, Y. Tang, J. Abdullah, M. Stigge, and W. Yi, "Scalable timing analysis with refinement." in *TACAS*, 2015, pp. 3–18.

[22] P. Jayachandran and T. Abdelzaher, "Transforming distributed acyclic systems into equivalent uniprocessors under preemptive and non-preemptive scheduling," in *Euromicro Conference on Real-Time Systems*, 2007, pp. 233–242.

[23] C. M. Krishna, *Real-Time Systems*. Wiley Online Library, 1999.