2016 IEEE 22nd International Conference on Embedded and Real-Time Computing Systems and Applications

Transforming Real-Time Task Graphs to Improve Schedulability

Chuancai Gu[1], Nan Guan[2], Zhiwei Feng[3], Qingxu Deng[3], Xiaobo Sharon Hu[1] and Wang Yi[3,4]

[1]University of Notre Dame, US [2]Hong Kong Polytechnic University, Hong Kong
[3]Northeastern University, China [4]Uppsala University, Sweden

***Abstract*—Real-time task graphs are used to describe complex real-time systems with non-cyclic timing behaviors. The workload of such systems are typically bursty, which may degrade their schedulability even with sufficient resource in the long term. In this paper, we propose to use task graph transformation to improve system schedulability. The idea is to insert artificial delays to the release times of certain vertices of a task graph to get a new graph with a smoother workload, while still meeting the timing constraints of the original task graph. Delaying the release time of a vertex may smoothen the workload of some paths of the task graph, but at the same time make the workload of other paths even more bursty. We developed efficient techniques to search for an appropriate release time delay for each vertex. Experiments with randomly generated task systems show that the proposed transformation method can make a significant number of task systems that was originally unschedulable to become schedulable, and the transformation procedure is very efficient and can easily handle large-scale task graph systems in very short computation time.**

I. Introduction

Traditionally, real-time task systems are modeled as collections of periodically repeating computational requests [15]. Behaviors that are not entirely periodic cannot be expressed accurately with this simple periodic task model. Instead, a natural representation of these processes is a task graph: a directed graph in which each vertex represents a code block and each edge represents a control flow. Over years, there have been many efforts to study more and more general graph-based real-time task models to precisely represent complex embedded real-time systems [4], [2], [17], [3], [23], [24], [25]. These graph-based task models can accurately express timing characterizations of expressive computation models such as Finite State Machines (FSM) [7], [19], [31], which is adopted in common modeling and code synthesis tools like Simulink Stateflow [5].

The system designer must guarantee a real-time system to be *schedulable*, i.e., at run time the timing constraints are respected under any circumstance. Complex real-time systems typically exhibit bursty behaviors, in some short time periods incurring workload much higher than the average. Therefore, timing constraints may still be violated even though in the long term the available resource is enough to process all the computation requests.

In this paper, we propose a method to improve schedulability of task graph systems with *static-priority scheduling*, by transforming task graphs to new ones with smoother workload. Graph transformation is done by adding extra delay to activations of certain vertices in the graph, and adjusting the parameters of related vertices/edges to guarantee that the resulting new graph fully complies with the timing behavior specification of the original graph.

The number of possible run-time activation sequences incurred by a task (corresponding to the paths in the graph) explodes exponentially. Delaying the activation of a vertex may smoothen the workload of certain paths, at the cost of making the workload of other paths potentially more bursty. It is computationally intractable to explicitly enumerate all the paths of a graph to decide how to transform a graph to get smoother workload and better system schedulability.

We develop techniques to *efficiently* transform an unschedulable task graph system into a schedulable one. Using our techniques, task set transformation is performed by modifying the parameters related to each vertex in task graphs step by step. We explore interesting properties and use proper abstractions to guide an efficient yet effective transformation procedure. Our transformation technique provides monotonic schedulability improvement guarantees at each step of the transformation procedure, in the sense that it can only make individual unschedulable tasks to become schedulable, but will not cause any task that was originally schedulable to become unschedulable. This property can guide the overall transformation procedure to quickly converge to a high-quality solution. Although our efficient technique in general does not guarantee to find the optimal solution, in practice it is very effective in successfully transforming unschedulable task systems to schedulable ones.

We evaluate the proposed technique by experiments using randomly generated real-time task graph systems. Experiment results show that our proposed method can significantly improve system schedulability: a significant number of task systems that were originally unschedulable becomes schedulable after the transformation. On the other hand, the transformation procedure is very efficient and can easily handle realistic-size task graph systems in very short time.

This work is presented in the context of the Digraph Real-Time (DRT) task model [23], which is a generalization of most existing graph-based real-time task models, such as GMF [17], RRT [2], and non-cyclic RRT [3]. All the results in this paper are directly applicable to these more restricted models as well.

*Corresponding author: Nan Guan, email: csguannan@comp.polyu.edu.hk

978-1-5090-2479-7/16 $31.00 © 2016 IEEE
DOI 10.1109/RTCSA.2016.13

A. Related Work

Much work has been done on the schedulability analysis of various graph-based real-time task models, including the multiframe (MF) task model [16], generalized multiframe (GMF) task model [4], non-cyclic GMF task model [17], recurring branching (RB) task model [1], recurring real-time (RRT) task model [2] et al. A generalization of the above models is the Digraph Real-Time (DRT) task model, which allows to model task release patterns by arbitrary directed graphs. It has been proved that the static-priority schedulability analysis problem of these graph-based models are strongly *co*NP-hard [26]. While previous work focuses on how to *analyze* the schedulability of task graph systems, this paper is the first work to study how to *improve* their schedulability to the best of our knowledge.

Shaping is a well-known technique in the area of networking, which delays datagrams to bring them into compliance with a desired traffic profile [8], [21]. By appropriate shaping one can smoothen the bursty traffic flows, to optimize the buffer requirement, improve latency, and/or increase usable bandwidth for some kinds of packets by delaying other kinds. The idea of shaping has been applied to the design of real-time embedded systems. Wandeler et al. [29], [30] extended the greedy shaper from network calculus [14] to modular performance analysis of real-time systems. Richter et al. [22] introduce a restricted kind of traffic shaping through so-called event adaption functions (EAFs). Phan and Lee [20] designed a new shaper for periodic tasks with jitters to smoothen the workload and improve schedulability. At a high level, task graph transforming has the same aim as shaping: they both smoothen the workload by forcing the original workload sequence to comply with certain extra regulations. However, in the problem of shaping real-time workload that has been studied in previous work, all the computation requests are identical (in terms of their worst-case execution times and relative deadlines), while in our problem a task releases different types of jobs corresponding to different vertices in the graph. Therefore, previous shaping techniques are not applicable to the task graph model considered in this paper.

II. Problem Model

In this section, we introduce the task model considered in this paper and some basic notions. The digraph real-time (DRT) task model [23] describes the workload of a system by a task set $\tau = \{T_1, T_2, \cdots, T_N\}$ of N independent tasks. Each task T is depicted by a directed graph $G(T)$ which contains vertex set $\{v_1, v_2, \cdots, v_n\}$ characterizing the various run-time job types and edge labels denoting the minimum job inter-release separation time. Each vertex v is labeled with an ordered pair $\langle e(v), d(v)\rangle$ characterizing the worst-case execution-time $e(v)$ and the relative deadline $d(v)$ of the corresponding job, respectively. Both values are defined in the domain of non-negative integers. The directed edges of $G(T)$ indicate the release order of jobs generated by T. Each directed edge (u,v) is labeled with a non-negative integer $p(u,v)$ denoting the minimum job inter-release separation time from u to v. In particular, we assume *constrained deadlines*, i.e., for each vertex u, its relative deadline $d(u)$ is no greater than the minimal $p(u,v)$ among all edges outgoing from u. We use $Prod(v)$ and $Succ(v)$ to denote the set of predecessor and successor vertices of v, respectively. Further, we define $Prod^-(v) \triangleq Prod(v) \setminus \{v\}$ and $Succ^-(v) \triangleq Succ(v) \setminus \{v\}$ to denote the predecessor and successor sets excluding the vertex v itself if included.

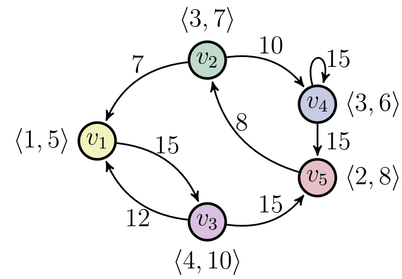


Figure 1: An example task containing five different jobs.

Semantics: The actual run-time behavior of each task T corresponds to a potentially infinite path through $G(T)$. Each visit to a vertex along that path causes a run-time job released with parameters labeled on the vertex. The inter-release separation times between successively released jobs through that path are constrained by the edge labels. Formally, we use a 3-tuple (r, e, d) to denote a job that is released at (absolute) time r, with execution time e and absolute deadline at time d. We assume dense time, i.e., $r, e, d \in \mathbb{R}_{\geq 0}$. A job sequence $\phi = [(r_1, e_1, d_1), (r_2, e_2, d_2), \cdots]$ is generated by T, if and only if there is a path $\pi = (v_1, v_2, \cdots)$ in $G(T)$ satisfying for all i:

1) $r_{i+1} - r_i \geq p(v_i, v_{i+1})$,
2) $e_i \leq e(v_i)$,
3) $d_i = r_i + d(v_i)$.

For a task set τ, a job sequence ϕ is generated by τ, if it is a composition of sequences ϕ_T, which are individually generated by tasks T of τ.

We use the example in Figure 1 to illustrate the semantics of DRT task systems. When the system starts, T releases its first run-time job by an arbitrary vertex. Then the released sequence corresponds to a particular directed path through $G(T)$. Consider the job sequence $\phi = [(5, 2, 13,), (15, 3, 22), (25, 3, 31)]$ which corresponds to path $\pi = (v_5, v_2, v_4)$ in $G(T)$. Note that this example demonstrates the "sporadic" behavior allowed by the semantics of the DRT model. The first job in ϕ (corresponding to v_5) is released at time 5, and the second job in ϕ (corresponding to v_2) is released 2 time units later than its earliest possible release time, while v_4 is released as early as possible after v_2.

We assume that the run-time job sequences are executed on a uniprocessor system and scheduled by a static-priority (SP) preemptive scheduler. Given a task set τ, a static priority assignment $\mathcal{P} : \tau \to \mathbb{N}$ assigns a unique priority value to each task T, denoted by $\mathcal{P}(T)$. Following the convention in real-time scheduling literatures, a smaller value represents a higher priority. For implicity we also use $\mathcal{P}(v)$ to denote the priority of the task containing vertex v. At run-time, the SP scheduler allocates the processor only to the job with the highest priority (the smallest $\mathcal{P}(v)$), among all the active jobs (those have been released by not finished yet).

A task set is *schedulable* if in all of its possible job sequences, all the released jobs finish execution before their absolute deadlines.

III. Basic Idea of Task Graph Transformation

In the following we explain the basic idea of the method proposed in this paper, namely, how can a DRT task be transformed to reduce its interference to lower-priority tasks and thus improve the schedulability. First consider the DRT task in Figure 2-(a) and a particular job sequence corresponding to path $\pi = (v_2, v_3, v_2, v_3, \cdots)$ as shown in Figure 2-(c). If we artificially postpone the release time of each instance of v_2 by one time unit, the resulting job sequence is shown in Figure 2-(d). The workload of the resulting job sequence becomes "smoother", which is potentially beneficial to the schedulability of lower-priority tasks.

The above modification equals to transforming the DRT task in Figure 2-(a) into the form of Figure 2-(b). The release separation of edge (v_2, v_3) is increased by 1. Since v_3 still has to meet its original deadlines, the distance between its delayed release time and its absolute deadline should be decreased by 1. The release separations on the edges outgoing from $v_3$2 are both decreased by 1 to comply with the original job release time separation constraints.

Now we define formal notations to describe the above transformation of a DRT task. We introduce a non-negative parameter *release delay* $\delta(v)$ for each vertex v. For each transformed vertex v, the release separation of each incoming edge is increased by $\delta(v)$, and its relative deadline $d(v)$ and the edge length of each outgoing edge is decreased by $\delta(v)$. In order to distinguish the release separations and relative deadlines before and after the transformation, we define the following notations

Definition 1. The *transformed inter-release separation*

$$pp(u, v) \triangleq p(u, v) - \delta(u) + \delta(v) \tag{1}$$

denotes the inter-release separation from u to v after transformation, and the *transformed relative deadline*

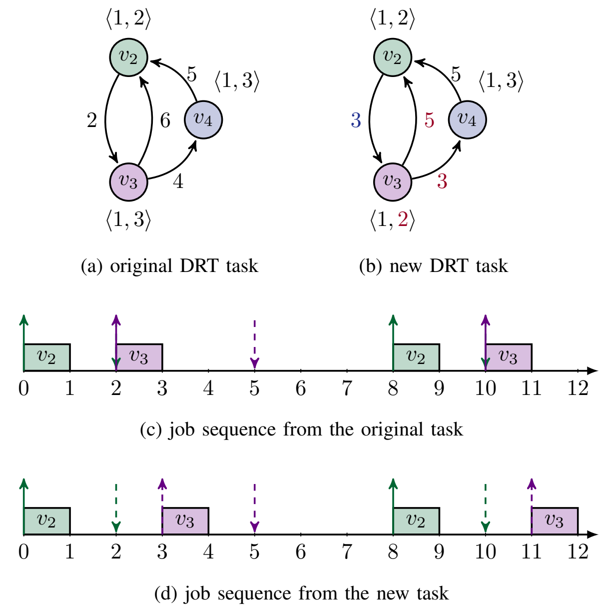
$$dd(v) \triangleq d(v) - \delta(v) \tag{2}$$

denotes the adjusted relative deadline due to release delay of the transformed vertex v.

The target of graph transformation is to assign the $\delta(v)$ value for each vertex v of each task, to make the task set schedulable if it was not originally. In the following we illustrate why this is not a trivial problem by a small task set of three simple DRT tasks, with the priority order $\mathcal{P}(T_1) < \mathcal{P}(T_2) < \mathcal{P}(T_3)$.

1) Consider the original task set and a job sequence in Figure 3-(a).

All tasks release their first run-time jobs (corresponding to v_1, v_2, v_5 respectively) at time 0 simultaneously. Then v_3 releases a job at 2. The accumulated workload during $[0, 3)$ is $e(v_1) + e(v_2) + e(v_5) + e(v_3) = 4$ which is greater than $d(v_5) = 3$, so job v_5 misses its deadline at time 3. The task set τ shown in Figure 3 is not schedulable by SP scheduling algorithm with the given priority order.

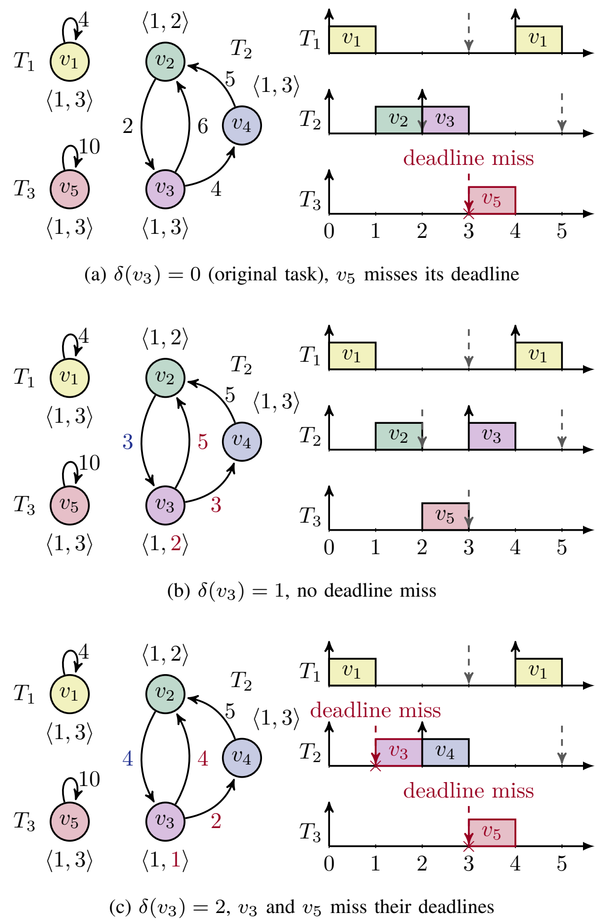


(a) original DRT task (b) new DRT task

(c) job sequence from the original task

(d) job sequence from the new task

Figure 2: Illustration of DRT task transformation



(a) $\delta(v_3) = 0$ (original task), v_5 misses its deadline

(b) $\delta(v_3) = 1$, no deadline miss

(c) $\delta(v_3) = 2$, v_3 and v_5 miss their deadlines

Figure 3: Schedulability with different values of $\delta(v_3)$

2) Transform task T_2 as shown in Figure 3-(b).

We set $\delta(v_3) = 1$, i.e., postpone the release of v_3 by one time unit. The resulting transformed relative deadline of v_3 and the transformed released separation of edges connected to v_3 is computed according to Definition 1. After the transformation, the workload released by path $(v_2, v_3, \cdots)$ becomes "smoother". The total released workload during $[0, 3)$ is $e(v_1) + e(v_2) + e(v_5) = 3$, so v_5 can finishes its execution by its deadline 3. On the other hand, although v_3's relative deadline is decreased by 1, it is still schedulable as $e(v_1) + e(v_3) = 2 = dd(v_3)$.

3) Transform task T_2 as shown in Figure 3-(c).

If we set $\delta(v_3) = 2$, then $pp(v_2, v_3) = pp(v_3, v_2)$. The workload released by v_2 and v_3 becomes even "smoother" than the above case. However, now v_3 becomes unschedulable since its own relative deadline becomes too short ($dd(v_3) = 3 - 2 = 1$). Moreover, setting $\delta(v_3) = 2$ not only makes v_3 unschedulable, but also causes T_3 to misses its deadline. This is because, although setting $\delta(v_3) = 2$ makes the workload alternatively released by v_2 and v_3 "smoother", it makes the workload of another path $(v_3, v_4, v_2, \cdots)$ even more busty and causes v_5 to misses its deadline, as shown in Figure 3-(c).

The above example demonstrates that postponing the release time of a vertex v may have both positive and negative effects to the system schedulability:

- On the *positive* side, it may smoothen the workload released by some paths of the task graph and improve the schedulability of lower-priority tasks.
- On the *negative* side, it may make the workload released by some paths more bursty and impair the schedulability of lower-priority tasks.
- On the *negative* side, it decreases the vertex v's own relative deadline and makes itself more difficult to be schedulable.

Given a non-trivial task graph, it is not clear whether delaying the release time of a vertex by a certain amount is beneficial to the system schedulability or not. The problem becomes even more complex when setting $\delta(v)$ for multiple vertices. The general complexity of deciding whether there exists an assignment of $\delta(v)$ for each v to make the task set schedulable is *co*NP-hard in the strong sense[1]. A naive solution would be enumerating all the possible combinations of all the possible candidate values of $\delta(v)$ for each vertex v and check the schedulability, which is computationally intractable. The target of this paper is to develop *efficient* techniques to do task graph transformation to improve system schedulability. Although our techniques do not guarantee to find the optimal assignments of all $\delta(v)$, they can quickly come to high-quality solutions, which make a significant portion of task sets that was originally unschedulable to become schedulable.

[1]Even the simpler problem of verifying the schedulability of a task set with a particular assignment of $\delta(v)$ for each v is strongly *co*NP-hard [26]

IV. Efficient Transformation Algorithm

A. Algorithm Overview

Since the exact schedulability test of a fixed DRT task set is already highly intractable, it is not affordable to use exact schedulability tests in the transformation procedure, which needs to repetitively analyze the system schedulability (explicitly or implicitly) to guarantee that the transformation is towards the right direction of improving system schedulability.

Instead, we will use the abstraction *request bound function*, $rbf_T(t)$, to perform sufficient schedulability tests efficiently. Section IV-B will introduce the definition of *request bound function* and the method to efficiently compute it. Intuitively, *request bound function* $rbf_T(t)$ bounds from above the interference of task T to lower-priority tasks in any time interval of length t.

Our algorithm transforms individual tasks from the highest to the lowest priority, i.e., the task with the smallest $\mathcal{P}$ first. With each task, the target is to decrease its interference to lower-priority tasks as much as possible, while not violating its own schedulability. If the transformation procedure returns **true**, it implies that the transformed task set is schedulable. However, if the algorithm returns **false**, it does *not* mean the task set is unschedulable. Instead, it only means the transformation procedure, which employs sufficient schedulability tests based on the request bound function abstractions, cannot guarantee the schedualbility of the task set. So after the transformation procedure, we shall apply the exact analysis in [27] to finally decide the schedulability of the task set.

Figure 4 shows the pseudo-code of the transformation algorithm. In the original DRT task set τ, the release delay $\delta(v)$ of each vertex v is 0. Transformation is performed from the highest- to the lowest-priority tasks. For each vertex v in the current task, it first calculates an upper bound for v's release delay, denoted by Δ_{slf}, such that setting $\delta(v)$ by any value below Δ_{slf} will not make v itself to become unschedulable if it was originally schedulable. Δ_{slf} is computed by routine Slf_Bound, which uses information of v itself and the request functions of all higher-priority tasks. Section IV-C will introduce Slf_Bound in detail. If Slf_Bound returns a negative value, then v may be unschedulable (could be a pessimistic decision), and the algorithm will not transform such vertices.

If Slf_Bound returns a positive Δ_{slf}, we can delay v's release time for up to Δ_{slf} time units to decrease the interference from the current task T to lower-priority tasks, without violating v's own deadline. Then the algorithm will decide another bound Δ_{itf} for $\delta(v)$ by routine Itf_Bound, to as much as possible decrease the interference to lower-priority tasks. Section IV-D will introduce Itf_Bound in detail. The smaller one between the above two bounds is the final choice of $\delta(v)$. After processing all the vertices of a task T, we calculate its interference function rbf_T, which will be used in the transformation of lower-priority tasks in following iterations. Finally, the algorithm is successful if it manages to set $\delta(v)$ for all the vertices of all tasks, with which the transformed task set is guaranteed to be schedulable.

In the following we will first introduce the formal definition of request bound function $rbf_T(t)$ and its efficient computation method in Section IV-B, and routines Slf_Bound and

Transform(τ)

$result \leftarrow$ **true**
$\forall v \in \tau : \delta(v) \leftarrow 0$
for each $T \in \tau$ in increasing order of $\mathcal{P}(T)$ **do**
  for each $v \in G(T)$ **do**
    $\Delta_{slf} = Slf\_Bound(v, \{rbf_{T'} | \mathcal{P}(T') < \mathcal{P}(T)\})$
    if $\Delta_{slf} \geq 0$ **then**
      $\Delta_{itf} = Itf\_Bound(v, \Delta_{slf})$
      $\delta(v) \leftarrow \min(\Delta_{slf}, \Delta_{itf})$
    else
      $result \leftarrow$ **false**
    end if
  end for
  Compute rbf_T
end for
return $result$

Figure 4: Pseudo-code of the Transformation Algorithm.

Itf_Bound in Section IV-C and IV-D, respectively.

B. Request Bond Function rbf_T

We first define the *request function* of a path in a task:

Definition 2 (Request Function of a Path)**.** Given a DRT task T, for a path $\pi = (v_0, \cdots, v_l)$ in graph $G(T)$, we define its *request function*, denoted by rf_π, as:

$$rf_\pi(t) \triangleq \max\{e(\pi') \mid \pi' \textit{ is prefix of } \pi \textit{ and } p(\pi') < t\}$$
$$where\ e(\pi) \triangleq \sum_{i=0}^{l} e(v_i) \textit{ and } p(\pi) \triangleq \sum_{i=0}^{l-1} pp(v_i, v_{i+1}).$$

$rf_\pi(t)$ is a non-decreasing staircase function with respect to t. Each horizontal segment is left-open and right-closed. In particular, $rf_\pi(0) = 0$.

Definition 3 (Request Bound Function)**.** For a path set $S = \{\pi_1, \cdots, \pi_n\}$, we define its *request bound function* rbf_S as:

$$rbf_S(t) \triangleq \max_{\pi_i \in S}\left\{rf_{\pi_i}(t)\right\}.$$

In particular, given a DRT task T, we define its *request bound function*, denoted by rbf_T, as:

$$rbf_T(t) \triangleq \max_{\pi \in G(T)}\left\{rf_\pi(t)\right\}.$$

A request bound function is also a staircase function and has the similar properties as the request functions. $rbf_T(t)$ can be computed in pseudo-polynomial time [9].

C. Routine $Slf\_Bound()$

The idea of $Slf\_Bound()$ is to estimate an upper bound R_v for the analyzed vertex v's response time. Then $d(v) - R_v$ is a safe upper bound for the values of v's release delay without violating v's own deadline, as stated in the following Lemma.

Lemma 1. *A DRT task T is guaranteed to be schedulable with release delay $\delta(v)$ satisfying $\delta(v) \leq d(v) - R_v$, where*

$$R_v \triangleq \min\left\{ t \,\middle|\, e(v) + \sum_{\mathcal{P}(T') < \mathcal{P}(T)} rbf_{T'}(t) \leq t \right\} \tag{3}$$

Proof: We prove the lemma by contradiction, assuming a job of vertex v is released at time t_r and misses its deadline at time t_d with a release delay $\delta(v)$. So $t_d - t_r = dd(v) = d(v) - \delta(v)$. We use $H(T)$ to denote the set of tasks with priority higher than T. By [26] we know the synchronous release pattern leads to the *critical instant* [15] in SP scheduling of DRT tasks. In other words, if there exists a job sequence where a job of v misses its deadline, then one can construct a job sequence where each task in $H(T)$ releases a jobs exactly at t_r. Therefore, without loss of generality, we assume in the considered job sequence each of the tasks in $H(T)$ releases a job at t_r, and use $\pi^{T'}$ to denote the path of a task $T' \in H(T)$ corresponds to the job sequence released by T' in $[t_r, t_d]$.

Since the job of v misses its deadline, we know for each time instant $t_1 \in [t_r, t_d]$ there are unfinished active jobs of tasks in $H(T)$ or task T itself. So we know that for any $t \in [0, d(v) - \delta(v)]$

$$e(v) + \sum_{T' \in H(T)} rf_{\pi^{T'}}(t) > t$$

By the definition of request bound functions, we know that $\forall t : rf_{\pi^{T'}}(t) \leq rbf_{T'}(t)$, so the above can be rewritten as

$$e(v) + \sum_{T' \in H(T)} rbf_{T'}(t) > t$$

since $\delta(v) \leq d(v) - R_v$, the above inequality implies that

$$e(v) + \sum_{T' \in H(T)} rbf_{\pi^{T'}}(R_v) > R_v$$

which contradicts (3). ■

The computation of R_v can be finished in pseudo-polynomial time since only the values in the range $[e(v), d(v)]$ need to be checked. If R_v does not exist in the range $[e(v), d(v)]$, then Slf_Bound returns a negative value, which means that v may not be schedulable (by sufficient tests) even with $\delta(v) = 0$ and the algorithm in Figure 4 will discard such vertex for further calculation of the release delay time. The fixed-point iteration technique in standard response time analysis [13] can be applied to further improve the efficiency for computing R_v.

Using Lemma 1 we can easily conclude the following theorem:

Theorem 1. *Given a DRT task set τ, if the transform algorithm in Figure 4 returns* ***true****, then the resulting new task set τ is schedulable.*

If the transformation algorithm returns **false**, it does *not* necessarily mean that the resulting new task set is unschedulable, since the schedulablility based on request bound functions is not exact. In that case we use the exact analysis in [27] to make final decision of its SP schedulablility.

D. Routine $Itf\_Bound()$

Assigning $\delta(v)$ by any value below the bound derived by routine $Slf\_Bound()$ in last section guarantees that the considered vertex v itself is still schedulable. However, as we discussed in Section IV-C, it is not clear which value is the best for the schedulability of other tasks (with lower priority).

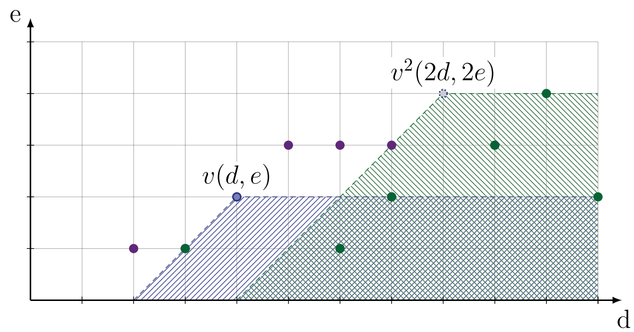


Figure 5: Illustration of vertex domination.

In this section, we introduce routine $Itf\_Bound()$, which decides such a proper value for $\delta(v)$. The bound returned by $Itf\_Bound()$ is not always optimal, but in most cases can significantly improve lower-priority tasks's schedulability.

Loosely speaking, the target of $Itf\_Bound()$ is to find a $\delta(v)$ value to decrease the *request bound function* $rbf_T(t)$ as much as possible. Clearly, it only makes sense to decrease $rbf_T(t)$ up to a certain t that is relevant to lower-priority tasks' schedulability. To find such an upper bound of t, a straightforward way is to get the maximal relative deadline of all lower-priority vertices. Actually, it is only necessary to consider the deadlines of the vertices that are "difficult to schedule", which can be formally defined based on the concept of *vertex domination*:

Definition 4 (Vertex Domination)**.** For two vertices v and v' of the same task T, we say that v *dominates* v', denoted by $v \succcurlyeq v'$, if at least one of the following condition holds:

$$(dd(v) - e(v)) \cdot \lceil e(v')/e(v) \rceil \le dd(v') - e(v') \qquad (4)$$

We say that v *strictly dominates* v', denoted by $v \succ v'$, if and only if $v \succcurlyeq v' \ \wedge \ v' \not\succcurlyeq v$.

Graphically, Figure 5 reveals the relationship under (4): For each vertex $v(d,e)$, we can draw series of regions $\{A^i \mid i \in \mathbb{N}^+\}$ controlled by $v^i(i \cdot d, i \cdot e)$ respectively, such as the shadow regions labeled by $v(d,e)$ and $(2d, 2e)$ in Figure 5. Note that for $i > 1$, v^i may be virtual vertices which don't belong to $G(T)$. If a vertex v' satisfying (4) with v, it must belong to some region controlled by v^i. In other words, the all vertices located in the union of regions $\cup A^i$ are dominated by the vertex $v(d,e)$.

With the domination relation, now we can formally define the vertices that are "difficult to schedule":

Definition 5 (Critical Vertex)**.** A vertex v that is *not strictly* dominated by any other vertices is called a *critical vertex*. The maximal set of critical vertices of a DRT task set τ is called the *critical vertex set* of τ, denoted by $CS(\tau)$.

We have the following property of critical vertices:

Lemma 2. *A DRT task set τ is SP-schedulable if and only if each of its critical vertices is schedulable.*

Proof: It is clear that if τ is SP-schedulable, all of its vertices (including the critical vertices) must be SP-schedulable.

For each DRT task $T \in \tau$, by Definition 4, we can prove that for any $v \in G(T)$ satisfying $v \notin CS(\tau)$, $\exists u \in CS(\tau)$ and $\exists v_1, \cdots, v_n \notin CS(\tau)$ hold that $u \succ v_1 \succ \cdots \succ v_n \succ v$. It is sufficient to show that, given a DRT task $T \in \tau$, and two vertices $v, v' \in G(T)$ satisfying $v \succcurlyeq v'$, v' is schedulable as long as v is schedulable. We prove it by contradictions.

Since $v \succcurlyeq v'$, one of the conditions listed in Definition 4 must be satisfied. Assume v is schedulable but v' is not schedulable.

We use $\beta(l)$ to denote the minimal accumulated amount of time during which the processor is available for T to execute, and we know that for any $t_1 \le t_2$ it holds

$$0 \le \beta(t_2) - \beta(t_1) \le t_2 - t_1 \qquad (5)$$

For simplicity of presentation, we let $d = dd(v)$, $d' = dd(v')$, $e = e(v)$ and $e' = e(v')$.

Since v is schedulable but v' is not, we have $\beta(d') < e'$ and $\beta(d) \ge e$. Aocording to (4), we have that

$$\begin{aligned}
& (d-e) \cdot \lceil e'/e \rceil \le d' - e' \\
\Rightarrow \quad & d \le \frac{d'}{\lceil e'/e \rceil} + e - \frac{e'}{\lceil e'/e \rceil} \\
\Rightarrow \quad & \beta(d) \le \beta\left(\frac{d'}{\lceil e'/e \rceil} + e - \frac{e'}{\lceil e'/e \rceil}\right) \\
\Rightarrow \quad & \beta(d) \le \beta\left(\frac{d'}{\lceil e'/e \rceil}\right) + e - \frac{e'}{\lceil e'/e \rceil} \\
\Rightarrow \quad & \beta(d) \le \frac{\beta(d')}{\lceil e'/e \rceil} + e - \frac{e'}{\lceil e'/e \rceil} \\
\Rightarrow \quad & \beta(d) < \frac{e'}{\lceil e'/e \rceil} + e - \frac{e'}{\lceil e'/e \rceil} \\
\Rightarrow \quad & \beta(d) < e
\end{aligned}$$

It contradicts with the assumption that the verticle (d,e) is schedulable. The contradiction proves the lemma. ∎

Since the schedulability of a task is fully determined by its critical vertices, when we choose $\delta(v)$ for a vertex v in a higher-priority task T, we only need to decrease $itf_T(t)$ for t up to the maximal deadline of critical vertices with priority lower than T, called *critical window size*:

Definition 6 (Critical Window Size)**.** The *critical window size* of a task T is defined as:

$$\rho_T \triangleq \max\left\{dd(v) | v \in CS(\tau), \mathcal{P}(v) > \mathcal{P}(T)\right\}. \qquad (6)$$

Note that it is possible that when we decrease $rbf_T(t)$ for $t \le \rho_T$, $rbf_T(t)$ may increase for some $t > \rho_T$. However, by Lemma 2 we know this does not affect the schedulability of any lower-priority vertices that are not in $CS(\tau)$, as long as we can guarantee that the ones in $CS(\tau)$ are all schedulable.

Now we introduce another important concept, the domination relation among paths:

Definition 7 (Path Domination)**.** Given two paths π and π' derived from task graph $G(T)$ of a DRT task T, we say π *dominates* π' up to x, denoted by $\pi \succcurlyeq_x \pi'$ if and only if

$$\forall t \in [0, x] : rf_\pi(t) \ge rf_{\pi'}(t)$$

We say two paths π and π' are *incomparable* if and only if neither $\pi \succcurlyeq \pi'$ nor $\pi' \succcurlyeq \pi$ holds.

As we discussed in Section III, delaying the release time of a vertex v may decrease the interference to lower-priority tasks along a path π, but increase the interference along another path π'. However, if we can guarantee that after the transformation it holds $\pi \succcurlyeq_{\rho_T} \pi'$, then the increase of $rf_{\pi'}(t)$ will not cause the worst-case interference of the task T', $rf_{\pi'}(T)$, to increase with any $t \in [0, \rho_T]$, and thus it will not hurt the schedulability of any lower-priority task.

More specifically, delaying the release time of v will potentially decrease (but not increase) $rf_{\pi}(t)$ for all t if π does not start with v, and will potentially increase (but not decrease) $rf_{\pi}(t)$ for all t if π starts with v. Therefore, we shall choose an as-large-as-possible value for $\delta(v)$ to decrease the interference of the paths that do not start with v as much as possible, as long as their (decreased) interference still dominates the (increased) interference of the v-started paths up to ρ_T. In the following we will in detail introduce how to find an upper bound for $\delta(v)$ to meet the above requirement.

Definition 8 (Lifting Point)**.** Given a staircase function f, its *lifting point* p is defined as: $f(p) < f(p^+)$, where $p^+ \triangleq p + \epsilon$ and ϵ indicates an arbitrary small positive value closing to 0.

Definition 9 (Dominating Point)**.** Given two staircase functions f and g, for any *lifting point* p on f, we say the *lifting point* q on g is the *dominating point* of p iif:

$$0 \leq q \leq p \wedge g(q) < f(p^+) \wedge g(q^+) \geq f(p^+). \tag{7}$$

If for each *lifting point* (in the time domain $[0, \rho)$) on f there exists a dominating point on f', we say that f' dominates f up to ρ, which is denoted by $f' \succcurlyeq_\rho f$.

Lemma 3. *Given a concrete path π and a path set $S = \{\pi_1, \cdots, \pi_n\}$, it holds that $rf_\pi \succcurlyeq_\rho rbf_S$ iif $\forall \pi_i \in S$, $\pi \succcurlyeq_\rho \pi_i$.*

Proof: Necessity: Since $\forall \pi_i \in S$, $\pi \succcurlyeq_\rho \pi_i$, we have that $\forall t \in [0, \rho] \mid rf_\pi(t) \geq rbf_S(t)$. So for each *lifting point* p on rbf_S, we have that $rf_\pi(p^+) \geq rbf_S(p^+)$. By the definition, $rf_\pi(0) = 0 < rbf_S(p^+)$, so there must exists some $p' \in [0, p]$ holds (7), i.e., p' dominates p.

Sufficiency: We prove the sufficiency by contradiction. Assume there exists some path $\pi_k \in S$ having $\pi \not\succcurlyeq_\rho \pi_k$. Thus there must exist $t_0 \in (0, \rho)$ such that $rf_{\pi_k}(t_0) > rf_\pi(t_0) \geq 0$. By the definition of *request bound function*, we have that $rbf_S(t_0) \geq rf_{\pi_k}(t_0) > rf_\pi(t_0)$. By the monotonicity of rbf_S, there must exist $p \leq t_0$ having $rbf_S(p) < rbf_S(t_0) \wedge rbf_S(p^+) = rbf_S(t_0)$, i.e., p is a *lifting point* on rbf_S before ρ. Further, $\forall t \leq p \leq t_0$ we have $rf_\pi(t) \leq rf_\pi(t_0) < rbf_S(p^+)$, i.e., p cannot be dominated by any *lifting points* on rf_π, which contradicts the assumption. ■

Increasing the *release delay* of a vertex v is beneficial to reduce the interference along its *inclusive paths* during $[0, \rho + \Delta]$, but may cause increase of interference along paths started with v. We will find the dominating paths to bound the impact of the v-started paths.

In the following, we focus on computing an upper bound of the release delay $\delta(v)$ for each *candidate* vertex $v \in G(T)$ by checking the existence of dominating paths (up to ρ_T) started with each possible prefix (u, v).

```
GenerateGreedyPath(u, ρ)
π ← (u)
v ← u {use v indicating the last vertex of π}
while p(π) < ρ ∧Succ(v) ≠ ∅ do
    from Succ(v) find v′ with the maximal e(v′)
    append v′ to the end of π
    v ← v′
end while
return π
```

Figure 6: Greedy algorithm for generating u-started path

To cover all the possible cases, we should compare each v-started path with the all candidate non v-started dominating paths. However, this is not computationally affordable since the numbers of both v-started and predecessor vertex started paths are exponential with respect to ρ_T.

To solve the above problem, we define the *v-started request bound function* to bound the *request* of any v-started path as below: $\forall t \geq 0$

$$rbf_T^v(t) \triangleq \max\{rf_\pi(t) \mid \pi \in G(T) \wedge \pi \ starts \ at \ v\} \tag{8}$$

On the other hand, for each vertex $u \in G(T) \wedge u \neq v$, we use a greedy approach to generate a concrete path π to check path domination during $[0, \rho]$. At first we set the initial path π to be (u). Then from the successive vertices of the last vertex v of π we select the vertex v' with largest $e(v')$ and append it to the end of π. We repeat this procedure until $p(\pi) \geq \rho$ or is exists no successive vertex of the last vertex of π. The detailed procedure is depicted in the pseudo-code of Figure 6.

With the two ideas introduced above, we introduce how to calculate the *release delay time*, as stated in the following lemma.

Lemma 4. *Given a v-started* request bound function rbf_T^v *which is dominated by rf_π of a u-started path π ($u \neq v$) up to $\rho + \delta$, and for each lifting point p on rbf_T^v we use p_d to denote its dominationg point on rf_π. If we increase $\delta(v)$ by δ, such that $\delta \leq dd(v) - e(v) < dd(v)$ and $\delta \leq \rho$ and*

$$\delta \leq \min\left\{ \frac{(p - p_d)}{2} \;\middle|\; p \ is \ lifting \ point \ on \ rbf_T^v \right\}$$
$$where \ p \in (0, \rho + \delta) \wedge rbf_T^v(p^+) > rf_\pi(0^+) = e(u).$$

then all v-started paths are still dominated by π up to ρ after increasing $\delta(v)$ by δ.

Proof: Since $rf_\pi \succcurlyeq_{\rho+\delta} rbf_T^v$, for each v-started path π_i, it satisfies that $\pi \succcurlyeq_{\rho+\delta} \pi_i$. So $\forall t \in [0, \rho + \delta] \mid rf_\pi(t) \geq rf_{\pi_i}(t)$.

As a result of increasing $\delta(v)$ by δ, for each vertex $u \in Pred^-(v)$, $pp(u, v)$ will be increased by δ. And for vertex $v' \in Succ^-(v)$, $pp(v, v')$ will be decreased by δ. Since $\delta \leq dd(v) - e(v) \leq pp(v) - e(v)$, the modified $pp(v, v')$ will not be less than $e(v) > 0$, so the successive vertices will not be overlapped. If the edge (v, v) exists, $pp(v, v)$ keeps its original value.

By the definition of *request functions*, for each v-started path π_i, the modified *request function* rf'_{π_i} after increasing

$\delta(v)$ by δ holds $\forall t \in [0, \rho] \mid rf'_{\pi_i}(t) \leq rf_{\pi_i}(t+\delta)$. And for the non v-started path π, its modified *request function* rf'_π holds $\forall t \in [0, \rho] \mid rf'_\pi(t+\delta) \geq rf_\pi(t)$.

Based on the discussions above, we have that for each $t \in (0, \rho)$, there must exists a lifting point $p \in [0, t+\delta)$ on rbf^v_T such that

$$rbf^v_T(p^+) = rbf^v_T(t+\delta) \geq rf_{\pi_i}(t+\delta) \geq rf'_{\pi_i}(t). \quad (9)$$

If $rbf^v_T(p^+) \leq rf_\pi(0^+)$, it is clear that

$$rf'_{\pi_i}(t) \leq rf_\pi(0^+) \leq rf'_\pi(t).$$

Then consider the case of $rbf^v_T(p^+) > rf_\pi(0^+)$. Because of $\pi \succcurlyeq_{\rho+\delta} \pi_i$, there also exists a lifting point p_d on rf_π which holds

$$\begin{aligned} rbf^v_T(p^+) &\leq rf_\pi(p_d^+) = rf_\pi(p^+ - (p - p_d)) \\ &\leq rf_\pi(t + \delta - (p - p_d)) \\ &\leq rf'_\pi(t + 2 \cdot \delta - (p - p_d)). \end{aligned}$$

Since $2 \cdot \delta \leq p - p_d$, it implies that

$$rbf^v_T(p^+) \leq rf'_\pi(t) \quad (10)$$

Combining the discussion above, for each $t \in (0, \rho)$, we have $rf'_{\pi_i}(t) \leq rf'_\pi(t)$. By the definition of *request bound function*, we can deduce that $rf'_\pi \succcurlyeq_\rho rbf'^v_T$. By Lemma 3, it proves this Lemma. ■

For each vertex $u \in G(T) \setminus \{v\}$, by Lemma 4, we can calculate a safe release delay bound to keep the original predecessor vertex domination. Then we can select the maximal calculated candidate values, since the *request functions* of the non v-started paths will be more smoother with a more greater $\delta(v)$ value.

The pseudo-code of the algorithm calculating the release delay upper bound based on the above discussions is shown in Figure 7. In line 3, it first generates the *request bound function* rbf^v_T, reusing the algorithm to compute the request bound functions in [9]. (but set RF_0 to be $\{\langle 0, 0, v\rangle\}$), to bound the accumulated workload through any v-started paths (exponentially many). In line 4 it records the all lifting points on rbf^v_T in the domain $[0, \rho)$. The loop from line 5 to line 13 visits each candidate vertex u $(\neq v)$ to find the maximal *release delay time*. For each vertex u $(\neq v)$, line 6 constructs a candidate dominating path π which starts with u, by the approach introduced in Figure 6. If $rf_\pi \succcurlyeq_{\pi+\Delta(v)} rbf^v_T$ (checked in line 8), line 9 and line 10 compute the *release delay* value according to Lemma 4. Then it records the maximal value to ret in line 11. Finally, line 14 returns the minimal value of the 3 parameters to satisfy the restriction of Lemma 4.

E. Properties of the Algorithm

Complexity: Given a DRT task set τ, by the pseudo-polynomial algorithm calculating *rbf* in [9] and Equation (3), we can calculate $\Delta_{slf}(v)$ in line 4 of Figure 4 in pseudo-polynomial time. As shown in Figure 7, the number of iterations to execute in algorithm $CalDelayBound$ is linear with respect to the number of vertices in τ, and all subroutines in Figure 7 are with pseudo-polynomial complexity.

CalDelayBound(v,ρ_T,Δ_{slf})
1: $\rho \leftarrow \rho_T + \Delta_{slf}$
2: $ret \leftarrow 0$
3: Generate rbf^v_T for the v-started paths up to ρ
4: $RS \leftarrow \{p \mid rbf^v_T(p) < rbf^v_T(p^+)\}$
5: **for each** $u \in G(T) \setminus \{v\}$ **do**
6: $\quad \pi \leftarrow$ GenerateGreedyPath(u, ρ)
7: $\quad$ generate the *request function* rf_π for π
8: $\quad$ **if** $\forall t \in [0, \rho] \mid rbf^v_T(t) \leq rf_\pi(t)$ **then**
9: $\quad\quad DS \leftarrow \{\langle p, p_d\rangle \mid p \in RS, rbf^v_T(p^+) > rf_\pi(0^+)\}$
10: $\quad\quad tem \leftarrow \min\{(p - p_d)/2 \mid \langle p, p_d\rangle \in DS\}$
11: $\quad\quad ret \leftarrow \max(ret, tem)$
12: $\quad$ **end if**
13: **end for**
14: **return** $\min\{ret, \rho_T, \Delta_{slf}\}$

Figure 7: Algorithm for calculating release delay bound.

Thus the overall complexity of calculating $\Delta_{itf}(v)$ is also pseudo-polynomial.

Further, the nested loop in Figure 4 is bounded by the number of vertices in τ, so the overall time complexity of the algorithm in Figure 4 is pseudo-polynomial.

Improvement Monotonicity:

Theorem 2. *Given a DRT task $T \in \tau$ which is SP-schedulable with a priority order $\mathcal{P}$, after any step in the vertex transformation in Figure 4, T is also SP-schedule.*

Proof: Consider the transformation of an arbitrary higher-priority task $T' \in \tau | \mathcal{P}(T') < \mathcal{P}(T)$ by the transforming algorithm in Figure 4.

For each $v' \in G(T')$, Δ_{itf} is bounded by Figure 7. By Lemma 4, we can conclude that with Δ_{itf}, the transformation of v' will not result in increased interference during $[0, \rho_{T'}]$, so for each $v \in CS(\tau) \cap G(T)$ the request function up to $d(v) \leq \rho_{T'}$ (by (6)) will not increase after the transformation of v'. Thus, the critical vertices of $G(T)$ will keep their schedulability, and by Lemma 2, we know task T will still be schedulable after any adjustments of higher-priority vertices.

Then we consider the transformation of T itself. By Lemma 1, we know that with the Δ_{slf} calculated by (3), each vertex v will not lose its original schedulability after the transformation.

Therefore, this theorem is proved. ■

V. Experimental Evaluation

The target of the experiments in this section is to evaluate the *effectiveness* of the proposed task graph transformation algorithm by comparing the number of schedulable task sets before and after the transformation.

A. Random Task Set Generation

The utilization of a DRT task T is the highest ratio between the accumulated $e(v)$ and the sum of release separation of vertices among all simple cycles in $G(T)$ [23]. The total utilization of a task set is the sum of individual tasks' utilizations. Clearly, a necessary condition for a DRT task set to be schedulable is that the total utilization is bounded by 1.

A task is generated as follows. A random number of vertices is created, connected by edges according to a specified out-degree. Edges are placed so that the graph is strongly connected. After choosing edge labels with a uniform distribution in a specified range, the deadline $d(v)$ of each vertex v is chosen with a uniformly distributed ratio to the minimal release separation of outgoing edges from v. Finally, the execution time $e(v)$ is generated with a uniformly distributed ratio to $d(v)$. The relative deadline of each vertex v is constrained by the minimal $p(v, v')$ of all edges outgoing from v.

The procedure of generating task sets is as follows. First a task set of two randomly generated tasks is constructed and evaluated. Then we randomly generate a new task and add it to the task set in the last step, and repeat this procedure until the total utilization of the task set exceeds 1. Then a new task set of two newly generated tasks is constructed. The whole procedure repeats until a sufficiently large number of task sets are generated and evaluated. The total utilization domain $(0, 1]$ is divided into X ranges with the same step $1/X$, and for each range the evaluation results are counted independently.

In order to evaluate the performance over different types of tasks, we create light tasks, medium tasks and heavy tasks, as shown in the following table. These types differ in the range of out-degree and the ragen of execution times. We conduct experiments for these three settings respectively.

Type	Vertices	Out-degree	p	e
Light	$[7, 15]$	$[1, 3]$	$[50, 300]$	$[1, 4]$
Medium	$[7, 15]$	$[1, 4]$	$[50, 300]$	$[1, 6]$
Heavy	$[7, 15]$	$[1, 5]$	$[50, 300]$	$[1, 8]$

We use the minimal deadline of the all vertices of $G(T)$ to decide task T's priority: the smaller deadline the higher priority. If more than one tasks share the same priority value, we give a random priories order between them.

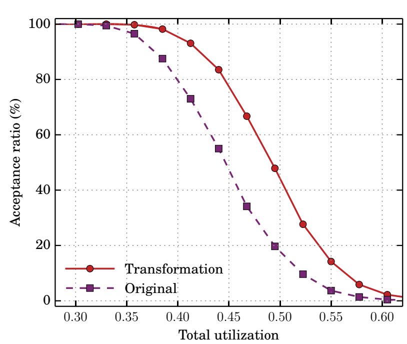


Figure 8: Improvement of acceptance ratio.

The *effectiveness* of our transformation algorithm is evaluated by the metric *acceptance ratio*: the ratio between the number of schedulable task sets and the total number of generated task sets. We compare this ratio between

- *Transformation*: the algorithm proposed in this paper.
- *Original*: the exact SP schedulablility test algorithm without task transformation proposed in [27].

Figure 8 shows the experiment results with the combined task types (randomly select one from the three types for current generated task). Experiment results with different types of tasks are illustrated in Figure 9. For each point in these figures, at least 5000 randomly generated task sets are evaluated. From the results we can find that the acceptance ratio can be improved significantly with our transforming algorithm under different settings. In the range of low total utilization (below 0.3), all the generated task sets are schedulable without transformation, so the improvement is zero. Finally, the improvement decreases again as the total utilization increases, where the task sets are very difficult to become schedulable due to the high total utilization.

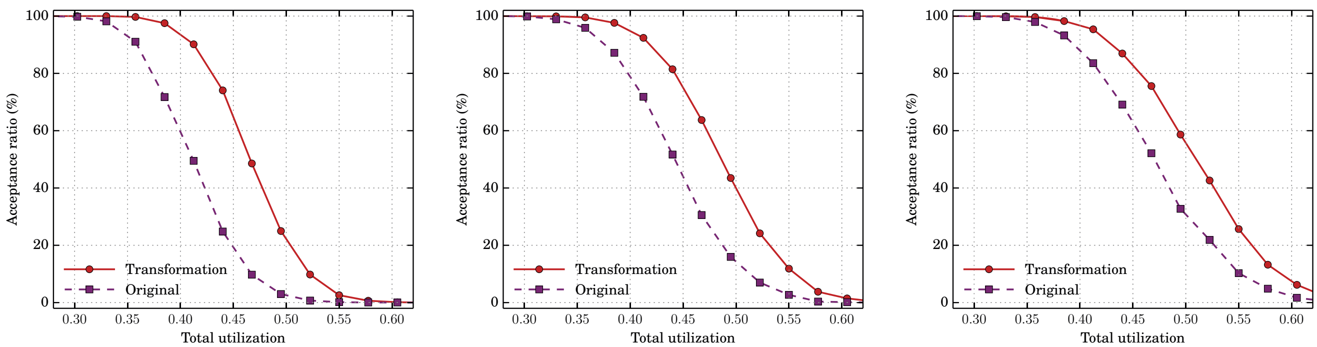
We also evaluate the efficiency of the proposed transformation algorithm. The experiments use an implementation in Python and execute on a desktop computer with an Intel Core i7-2600 CPU (3.40GH). Both our approach and the original approach in [27] execute the refinement-based exact schedualbility analysis once. The difference is that our approach will first do task transformation before that. However, in all the experiments we have conducted the extra timing overhead incurred by task transformation is very low, comparing to the time used for the exact schedulability analysis. (typically $< 5\%$ extra time overhead). So we can conclude that the task transformation is very efficient, and can be used to handle large-scale task systems.

VI. Conclusions and Future Work

We proposed to use task graph transformation to improve the schedulability of DRT task systems. The transformation is performed by inserting certain amount of delay before the release time of each vertex. However, in general the release delay may lead to both positive and negative effects of the system schedulability. The challenge is how to efficiently decide the delay for each vertex of each task to maximize the chance to transform unschedulable task sets to schedulable ones. We developed efficient techniques to solve this problem, which guarantees the interference workload of critical vertices not to be increased in the transformation procedure in the sense that it will never degrade the schedulability of any individual task. This property can efficiently guide the transformation procedure to quickly come to a high-quality solution. Experiments with randomly generated task sets shows our proposed techniques is very efficient and can significantly improve the schedulability of task graph systems. In the future, we will extend this work to deal with more general task graphs with precedence constraints [10], [6] and task graph systems of parallel workload with fork-join semantics [28] on multiprocessor systems, with both global scheduling [18] and partitioning based scheduling [11], [12].

Acknowledgement

This work is partially supported by collaborative Innovation Center of Major Machine Manufacturing in Liaoning, the State Key Laboratory of Synthetical Automation for Process Industries (PAL-N201503), National Natural and Science Foundation of China under grant no. 61300022, 61472072 and 61532007 and National Basic Research Program of China (973 Program) under grant No. 2014CB360509.



(a) Improvement of LIGHT type of tasks. (b) Improvement of MEDIUM type of tasks. (c) Improvement of HEAVY type of tasks.

Figure 9: Comparison of acceptance ratio between different task types.

References

[1] Sanjoy Baruah. Feasibility analysis of recurring branching tasks. In *Real-Time Systems, 1998. Proceedings. 10th Euromicro Workshop on*, pages 138–145. IEEE, 1998.

[2] Sanjoy Baruah. Dynamic-and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003.

[3] Sanjoy Baruah. The non-cyclic recurring real-time task model. In *Proceedings of the IEEE 31st Real-Time Systems Symposium (RTSS)*, pages 173–182, Nov 2010.

[4] Sanjoy Baruah, Deji Chen, Sergey Gorinsky, and Aloysius Mok. Generalized multiframe tasks. *Real-Time Systems*, 17(1):5–22, 1999.

[5] Mariusz Bernacki. Simulink stateflow.

[6] Pontus Ekberg, Nan Guan, Martin Stigge, and Wang Yi. An optimal resource sharing protocol for generalized multiframe tasks. *Journal of Logical and Algebric Method of Programming*, 2015.

[7] Arthur Gill. Finite-state machines. *IEEE Transactions on Computers*, 19(11), 1970.

[8] S. Gringeri, K. Shuaib, R. Egorov, A. Lewis, B. Khasnabish, and B. Basch. Traffic shaping, bandwidth allocation, and quality assessment for mpeg video distribution over broadband networks. *IEEE Networks*, 12(6):94–107, 1998.

[9] Nan Guan, Chuancai Gu, Martin Stigge, Qingxu Deng, and Wang Yi. Approximate response time analysis of real-time task graphs. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS)*, pages 304–313, 2014.

[10] Nan Guan, Martin Stigge, Pontus Ekberg, and Wang Yi. Resource sharing protocols for real-time task graph systems. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS)*, 2011.

[11] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Fixed-priority multiprocessor scheduling with liu and layland's utilization bound. In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2010.

[12] Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. Parametric utilization bounds for fixed-priority multiprocessor scheduling. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2012.

[13] Mathai Joseph and P Pandya. Finding response times in a real-time system. *The Computer Journal*, 29(5):390–395, 1986.

[14] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*, volume 2050. Springer, 2001.

[15] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1):46–61, January 1973.

[16] Aloysius K Mok and Deji Chen. A multiframe model for real-time tasks. *IEEE Transactions on Software Engineering*, 23(10):635–645, 1997.

[17] Noel Tchidjo Moyo, Eric Nicollet, Frederic Lafaye, and Christophe Moy. On schedulability analysis of non-cyclic generalized multiframe tasks. In *Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 271–278. IEEE, 2010.

[18] Wang Yi Ge Yuu Nan Guan, Martin Stigge. New response time bounds for fixed priority multiprocessor scheduling. In *Proceedings of the 19th IEEE Real-Time Systems Symposium (RTSS)*, 2009.

[19] Marco Di Natale and Haibo Zeng. Task implementation of synchronous finite state machines. In *DATE*, pages 206–211, 2012.

[20] L.T.X. Phan and Insup Lee. Improving schedulability of fixed-priority real-time systems using shapers. In *Proceedings of the IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 217–226, April 2013.

[21] J. Rexford, F. Bonomi, A. Greenberg, and A. Wong. Scalable architectures for integrated traffic shaping and link scheduling in high-speed atm switches.. *IEEE Journal on Selected Areas of Communication*, 15(5):938–950, 1997.

[22] Kai Richter, Marek Jersak, and Rolf Ernst. A formal approach to mpsoc performance verification. *IEEE Computer*, 36(4):60–67, 2003.

[23] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. The digraph real-time task model. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 71–80. IEEE, 2011.

[24] Martin Stigge, Pontus Ekberg, Nan Guan, and Wang Yi. On the tractability of digraph-based task models. In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 162–171. IEEE, 2011.

[25] Martin Stigge, Nan Guan, and Wang Yi. Refinement-based exact response-time analysis. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 143–152. IEEE, 2014.

[26] Martin Stigge and Wang Yi. Hardness results for static priority real-time scheduling. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 189–198. IEEE, 2012.

[27] Martin Stigge and Wang Yi. Combinatorial abstraction refinement for feasibility analysis. In *Procedings of the 34th IEEE Real-Time Systems Symposium (RTSS)*, pages 340–349, 2013.

[28] Jinghao Sun, Nan Guan, Yang Wang, Qingxu Deng, Peng Zeng, and Wang Yi. Feasibility of fork-join real-time task graph models: Hardness and algorithms. *ACM Transactions on Embedded Computing Systems*, 2016.

[29] Ernesto Wandeler, Alexander Maxiaguine, and Lothar Thiele. Performance analysis of greedy shapers in real-time systems. In *Design, Automation and Test in Europe, 2006. DATE'06. Proceedings*, volume 1, pages 6–pp. IEEE, 2006.

[30] Ernesto Wandeler, Alexander Maxiaguine, and Lothar Thiele. On the use of greedy shapers in real-time embedded systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(1):1, 2012.

[31] Haibo Zeng and Marco Di Natale. Schedulability analysis of periodic tasks implementing synchronous finite state machines. In *Proceedings of the 26th Euromicro Conference on Real-Time Systems (ECRTS)*, pages 353–362, 2012.