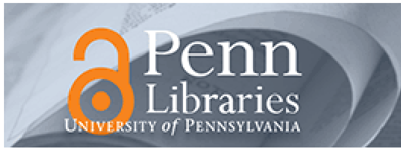

University of Pennsylvania
ScholarlyCommons

Departmental Papers (CIS) Department of Computer & Information Science

June 2008

Robust and Sustainable Schedulability Analysis of Embedded Software

Madhukar Anand
University of Pennsylvania, anandm@cis.upenn.edu

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Follow this and additional works at: http://repository.upenn.edu/cis_papers

Recommended Citation

Madhukar Anand and Insup Lee, "Robust and Sustainable Schedulability Analysis of Embedded Software", . June 2008.

Postprint version. Published in *Proceedings of the ACM SIGPLAN/SIGBED 2008 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2008)*, June 2008.

This paper is posted at ScholarlyCommons. http://repository.upenn.edu/cis_papers/372
For more information, please contact libraryrepository@pobox.upenn.edu.

Robust and Sustainable Schedulability Analysis of Embedded Software

Abstract
For real-time systems, most of the analysis involves efficient or exact schedulability checking. While this is important, analysis is often based on the assumption that the task parameters such as execution requirements and inter-arrival times between jobs are known exactly. In most cases, however, only a worst-case estimate of these quantities is available at the time of analysis. It is therefore imperative that schedulability analysis hold for better parameter values (Sustainable Analysis). On the other hand, if the task or system parameters turn out to be worse off, then the analysis should tolerate some deterioration (Robust Analysis). Robust analysis is especially important, because the implication of task schedulability is often weakened in the presence of optimizations that are performed on its code, or dynamic system parameters.

In this work, we define and address sustainability and robustness questions for analysis of embedded real-time software that is modeled by conditional real-time tasks. Specifically, we show that, while the analysis is sustainable for changes in the task such as lower job execution times and increased relative deadlines, it is not the case for code changes such as job splitting and reordering. We discuss the impact of these results in the context of common compiler optimizations, and then develop robust schedulability techniques for operations where the original analysis is not sustainable.

Keywords
schedulability analysis, sustainable schedulability analysis, robust schedulability analysis

Comments
Postprint version. Published in *Proceedings of the ACM SIGPLAN/SIGBED 2008 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2008)*, June 2008.

This conference paper is available at ScholarlyCommons: http://repository.upenn.edu/cis_papers/372

Robust and Sustainable Schedulability Analysis of Embedded Software *

Madhukar Anand and Insup lee

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 19104
{anandm,lee}@cis.upenn.edu

Abstract

For real-time systems, most of the analysis involves efficient or exact schedulability checking. While this is important, analysis is often based on the assumption that the task parameters such as execution requirements and inter-arrival times between jobs are known exactly. In most cases, however, only a worst-case estimate of these quantities is available at the time of analysis. It is therefore imperative that schedulability analysis hold for better parameter values (*Sustainable Analysis*). On the other hand, if the task or system parameters turn out to be worse off, then the analysis should tolerate some deterioration (*Robust Analysis*). Robust analysis is especially important, because the implication of task schedulability is often weakened in the presence of optimizations that are performed on its code, or dynamic system parameters.

In this work, we define and address sustainability and robustness questions for analysis of embedded real-time software that is modeled by conditional real-time tasks. Specifically, we show that, while the analysis is sustainable for changes in the task such as lower job execution times and increased relative deadlines, it is not the case for code changes such as job splitting and reordering. We discuss the impact of these results in the context of common compiler optimizations, and then develop robust schedulability techniques for operations where the original analysis is not sustainable.

Categories and Subject Descriptors D.4.7 [*Operating Systems*]: Organization and Design – Real-time and embedded systems

General Terms Design, Theory

Keywords Schedulability Analysis, Sustainable schedulability analysis, Robust schedulability analysis

1. Introduction

Work on schedulability analysis of tasks is mostly focused on efficiency of schedulability checking, or exactness of the result. Most of the techniques proposed make use of worst case estimates of task parameters such as execution requirements and inter-arrival times between jobs. While it is important that the task set be schedulable in the worst case, it is also important that the analysis be sustainable if the task parameters turn out to be better than those considered. By sustainability of schedulability analysis, we refer to the property that a system remains schedulable even if the some of the parameters of the task or the system turn out to be better than what the analysis accounted for. For instance, if the task finishes execution before its worst-case execution time, or if a job in the task is replaced by one that requires a smaller amount of resource, then the original analysis should not be affected. Although it may seem obvious that the analysis should hold if things get better, it is not always the case, even for simple tasks. For example, Baruah and Burns (Baruah and Burns 2006; Burns and Baruah 2008) have shown that the Leung and Whitehead test for fixed priority scheduling of periodic tasks (Leung and Whitehead 1982) is not sustainable in many cases, including the case where task periods increase.

Another area of concern with the analysis is tolerating a worsening of parameters. Scaling of processor speeds, presence of variable jitter between jobs, additional delays induced due to dependencies, all affect the analysis. Task parameters could also change from those considered in the analysis. For example, some classical compiler optimizations, such as those targeted towards reducing execution requirements or memory use in real-time programs, could cause problems as far as analysis of meeting deadlines is concerned. In fact, there are cases where even though the optimization reduces overall execution requirements, the system may not be schedulable (as explained in Sec. 3). It is also possible that the task set remains schedulable after the optimization, but the original analysis does not hold. Past research effort has focused on ensuring that compiler optimizations are safe (e.g.,(Marlowe 15-18 Jun 1992; Younis et al. 21-25 Oct 1996)), and techniques for reducing code size without affecting the real-time constraints (e.g., (Lee et al. 2008)) In contrast, we take the view that if the task parameters turn out to be worse off, then the analysis should be robust enough to tolerate some deterioration. Obviously, no analysis can permit an arbitrary deterioration of parameters and consideration of higher deterioration leads to more pessimistic results in the analysis.

In this paper, we develop a framework that takes as input, the space of all anticipated changes to inter-arrival times and deadlines of tasks due to code changes such as job splitting and reordering, and that generates as output, the maximum possible load under those modifications. The idea is that, if this maximum load can be supported by the resource supply, then the analysis will be robust and sustainable under any actual modification of the task.

Some of the specific questions in the context of robustness of analysis include, (1) can some of the jobs in a task change by the way of increased execution times and still remain schedulable? If so, what is the maximum change that can be tolerated without affecting schedulability. (2) given a set of real-time tasks, what is a priority assignment for tolerating the maximum amount of additional interference from the system? (3) if some of the task parameters such as inter-arrival times between jobs change at runtime, how can schedulability still be guaranteed? and, (4) how do com-

* This research was supported in part by FA9550-07-1-0216, NSF CNS-0509143, NSF CNS-0720703, and NSF CNS-0720518

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

LCTES'08, June 12–13, 2008, Tucson, Arizona, USA.
Copyright © 2008 ACM 978-1-60558-104-0/08/06. . . $5.00

piler optimizations affect schedulability of the task? In other words, can some operations be identified that are deemed safe with respect to robust schedulability analysis?

In this paper, we address questions of sustainability and robustness of analysis for embedded software modeled as conditional real-time tasks. Specifically, we consider the task to be modeled as a Recurring branching Task with Control variables (RTC) (see (Anand et al. 2008)). We expect that the task model, being a generalization of periodic, sporadic, and multi-frame tasks can capture requirements of real-world applications more closely. The RTC model also extends the recurring branching task model (Baruah 1998) with guarded transitions based on assignments to control variables.

We first define sustainability and robustness in the context of analysis with conditional real-time tasks, and later address some of the concerns. For sustainability of the analysis, we work with the following definition.

Definition 1 (Sustainability)**.** *A schedulability test for a scheduling policy is* sustainable *if any system modeled by a set of conditional tasks deemed schedulable by the schedulability test remains schedulable when the parameters of one or more individual task(s) are changed in any combination of the following ways: (i) decreased execution requirements of individual jobs, (ii) larger inter arrival times, (iii) larger relative deadlines, and (iv) structural optimizations leading to overall lower resource requirements.*

The definition of sustainability is adapted from a similar definition for simpler task models by Baruah and Burns (Baruah and Burns 2006)[1]

With respect to robustness, we focus on answering questions (1) and (3) from the list of questions identified earlier. For question (1), we try to find the maximum possible constant scaling factor by which all the execution times can be scaled. For question (3), we formulate a constrained optimization problem to find the task parameters that present, in some sense, the worst possible load on the system, and prove that if the requirements of that task are met, then a task with any assignment of the parameters meeting the constraints is schedulable. Question (2) above, relates to the concept of robust priority ordering, which was introduced by Davis and Burns (Davis 3-6 Dec. 2007) in the context of periodic tasks. A similar technique is applicable to robust priority assignment for conditional tasks. We leave the prospect of answering the final question as future work.

1.1 Related work

Robustness been introduced with various connotations in the literature (e.g., (Davis 3-6 Dec. 2007; Marlowe 15-18 Jun 1992; Rhan and Liu 21-24 Jun 1994; Buttazzo May 2006; Yerraballi et al. 14-16 Jun 1995; Buttazzo and Stankovic 1993)). For example, Davis and Burns (Davis 3-6 Dec. 2007) have addressed the question of robust priority assignment for periodic tasks, so that the system can tolerate the maximum amount of additional system interference. Buttazzo and Stankovic (Buttazzo and Stankovic 1993) consider the problem of robust scheduling strategy, i.e., designing a scheduling strategy that can react the best under system overload. There is also a large body of work on analysis under scaling of processor speed (e.g., (Buttazzo May 2006; Yerraballi et al. 14-16 Jun 1995)). While many of these questions are relevant in the context of conditional task models, our primary contribution in this work is on developing robust analysis techniques without having to modify the task set, priority assignment of tasks, or the scheduling algorithm.

Sustainability of schedulability analysis was introduced by Baruah and Burns (Baruah and Burns 2006). In their work, they looked at sustainability of standard scheduling tests for systems modeled as periodic tasks and its extensions. Ha and Liu (Rhan and Liu 21-24 Jun 1994) define a property of scheduling algorithms that they call "predictability". Informally, a scheduling algorithm is predictable if any task system that is scheduled by it to meet all deadlines will continue to meet all deadlines if some jobs arrive earlier, or have later deadlines. Other related work on sustainability of scheduling analysis includes the work by Mok and Poon (A.K.Mok and Poon 5-8 Dec. 2005) on non-preemptive scheduling of periodic task systems. There is also a lot of work on scheduling anomalies (e.g.,(Buttazzo May 2006; Andersson 2002; Chen et al. 2005; Racu and Ernst 2006)), especially in the multiprocessor case, under assumptions of variable processor speed, or increased execution times.

Much of the sustainability of analysis questions stem from the fact that estimation of many task parameters such as inter-arrival times or worst case execution times (WCET) are only approximate. In fact, many WCET estimation techniques such as those based on abstract interpretation (Ferdinand et al. 2001) give us an upper bound. An alternative to using these scheduling tests is to use model checking based techniques (e.g.,(Ben-Abdallah et al. 1998; Altisen et al. 2002)) to ascertain the schedulability of task sets. As these techniques produce exact analysis (as opposed to worst-case analysis), they do not have sustainability problems. They are, however, not robust for the same reason. Any change to the task parameters after the analysis is performed invalidates the analysis.

Finally, in other related work, we would like to mention the Hierarchical Timing Language (HTL) (Ghosal et al. 2006) which has been proposed for real-time tasks. The key idea there is that of task refinement which results in sustainable analysis by design.

The rest of the paper is organized as follows. we introduce the task models in Section 2. In Section 3, we analyze the sustainability of schedulability analysis with conditional tasks. In Section 4, we develop robust schedulability analysis techniques. We conclude in Section 5.

2. System model and definitions

Embedded real-time programs are typically implemented as some event-driven code embedded within an infinite loop. In many applications, the action to be taken upon the occurrence of external events depends on factors such as the current state of the system, values of external variables, etc. These systems have been traditionally modeled using well known task frameworks such as periodic/sporadic tasks, task graphs, and timed automata. Periodic/sporadic tasks have been well studied with respect to schedulability, but lack the expressivity of task graphs and timed automata when it comes to modeling embedded real-time programs. On the other hand, task graphs and automata models are more expressive than periodic/sporadic tasks but schedulability analysis for them is hard. In this work, we use a subclass of task graphs, called recurring branching tasks with control variables. These models are expressive enough to model conditional release of jobs within embedded programs, but at the same time allow efficient demand computation for schedulability checking.

2.1 Recurring branching Task with Control variables (RTC)

In this section, we define our task model and its execution semantics. Our system consists of multiple real-time components sharing a global resource (e.g., CPU, shared network, etc.) under a hierarchical scheduling policy. The shared resource demand of each component can be represented by a set of tasks, each comprising of multiple simple tasks as the basis for demand.

The resource supply to the tasks is assumed to be provided according to a resource model (e.g., (Shin and Lee 2003; Lipari and Bini 2003; Feng and Mok 2002)) For example, a periodic resource model (see (Shin and Lee 2003; Lipari and Bini 2003)) $\Gamma = (\Pi, \Theta)$ is a partitioned resource supply such that it guarantees Θ allocations

[1] There are minor changes in terminology. For instance, while the definition in (Baruah and Burns 2006) uses "job release jitter", we refer to that quantity as job inter-arrival time.

of time units every Π time units, where a resource period is a positive integer and a resource allocation time is a real number in $(0,\Pi]$. For the resource model, the minimum resource supply provided by it in an interval t is measured by the supply bound function, sbf. For a periodic model Γ, its supply bound function $\text{sbf}_\Gamma(t)$ is defined to compute the minimum resource supply for every interval length t as follows:

$$\text{sbf}_\Gamma(t) = \begin{cases} t-(k+1)(\Pi-\Theta) & \text{if } t \in [(k+1)\Pi-2\Theta,(k+1)\Pi-\Theta], \\ (k-1)\Theta & \text{otherwise,} \end{cases} \tag{1}$$

where $k = \max\left(\left\lceil \left(t-(\Pi-\Theta)\right)/\Pi \right\rceil, 1\right)$. For the full processor, the supply function is simply $\text{sbf}(t) = t$.

A *simple task* $T = (e,d)$ requires e time units of the resource within d time units of its release. Informally, a RTC model is a structure consisting of nodes and transitions between these nodes, where each node defines a release of a simple task and each transition identifies the minimum jitter between successive task releases.

Definition 2 (RTC Model)**.** *A RTC model Ω is defined by a tuple $\langle V, v^0, V^F, E, \tau, \rho\rangle$ where,*

- *V is a set of nodes,*
- *$v^0 \in V$ is the start node,*
- *$V^F \subseteq V$ is a set of final nodes called leaves,*
- *$E \subseteq V \times V = E_T \cup E_R$ is a set of transitions where E_R is a set of resets,*
- *$\tau : V \to \mathcal{T}$ is a function from nodes to simple tasks,*
- *$\rho : E \to \mathbb{R} \times G \times A$ is a function from a transition to minimum jitter, an enabling condition G, and a variable assignment A*

$E_R = \{\langle v, v^0\rangle | v \in V^F\}$ and $E_T = E \setminus E_R$ such that the underlying graph (V, E_T) is a directed tree.

$a \in A$ consists of assignment for variables in $\mathcal{V}$ and $g \in G$ is any decidable function over the variables $\mathcal{V}$. For this model, we assume that any node releases one simple task. Multiple task releases can be handled by transitions with zero jitter. We make the following assumptions for a RTC model (1) the set of enabling conditions $g_1, \ldots, g_m$ on transitions leaving a node must be exhaustive, i.e., $\bigvee_{j=1}^{m} g_j =$ true (progress). (2) the enabling conditions and assignments have no overhead in terms of space and time. This assumption simplifies presentation of the paper, and the overhead can be easily integrated into our analysis, and (3) the set of leaf nodes V^F is nonempty, and every other node has a run to one of the leaf nodes.

The execution semantics of a RTC model may be described as follows. The execution starts at node v^0 where the task $\tau(v^0)$ is released. After the release, a transition from v^0 that is enabled to one of the descendant nodes of v^0 (say, v) is taken after a minimum delay as specified on $\rho(\langle v^0, v\rangle)$, and this process of task release continues from node v. The enabling conditions/variable assignments on a transition from v_i are assumed to be evaluated/executed immediately after the release of task $\tau(v_i)$, which is instantaneous.

We note that a RTC task model generalizes many known task models such as the periodic, sporadic, multi-frame, and recurring branching task model (Baruah 2003). The 3TS system models (Figure 2) gives an example of RTC model. We now introduce some more definitions related to the RTC model.

Definition 3 (Run)**.** *A run $r \equiv run(v^i, v^{i+j}, t)$ is a sequence of progression of nodes from v^i to v^{i+j} of a RTC model $\Omega = \langle V, v^0, V^F, E, \tau, \rho\rangle$: $v^i \xrightarrow{e^{i+1}} v^{i+1} \xrightarrow{e^{i+2}} \ldots \xrightarrow{e^{i+j}} v^{i+j}$ where $\forall l \in [1, j]$, $e^{i+l} = \langle v^{i+l-1}, v^{i+l}\rangle \in E$ and $t = \sum_{k=1}^{l} \rho(e^{i+k})_1$, where $\rho(e)_1$ is the projection of transition function onto the first component, i.e., its jitter value. We also denote by $\gamma(r)$ the duration t of run r. Also, the* resource demand *of the run r is defined as $\Delta(r) = \sum_{l=0}^{j} \tau(v^{i+l}).e$.*

In this definition $\tau(v^{i+l}).e$ represents the execution requirement of task $\tau(v^{i+l})$.

RTC model example. Consider the example of the Three Tanks System (3TS) (Iercan 2005) shown in Figure 1. The plant consists of interconnected water tanks, where each tank has evacuation taps for simulating perturbation. Two of the tanks (T1 and T2) can pump water into the respective tanks via the pumps P1 and P2. The plant is nonlinear and hence it uses three different controllers for each pump. (1) A controller P (proportional) is used for the case when there is no perturbation (no water leaves the tank). (2) Two controllers PI (proportional integrator) are used when there is some perturbation (water drains out of the tank). When the control error is large, a controller with fast integration speed is used and otherwise, a controller with slow integration speed is used.

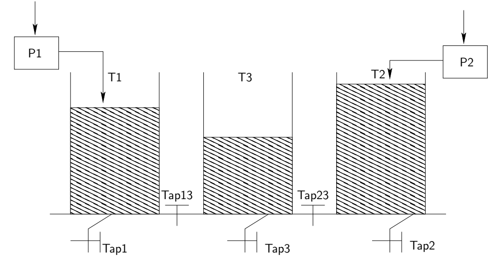

Figure 1. Overview of 3TS

We show the partial code for the system implemented using HTL in Listing 1. Although the original HTL model is hierarchical, we have expanded the internal modes and created a non-hierarchical RTC model. It can be seen that, for the purposes of computing the resource requirement, these are equivalent. The model for the other pump is similar. While the HTL language does not explicitly mention jitter between job triggering, it allows for such jitter and is mainly concerned with ensuring order of logical release of jobs. We have therefore included some jitter between release of jobs in the example.

Figure 2 shows the conditional models for modules in 3TS (see HTL code (Iercan 2005)). The nodes and transitions between nodes for the model are illustrated in the figure. The mapping from transitions to enabling conditions and assignments is listed beside the figure. The reset transitions are indicated by dashed lines. The set of leaves are the nodes which have the dashed transitions out of them. Each node is also annotated with the simple task that is released at that node. A run of the task would begin at the start node *R* and would follow the transition that is enabled. In the case of the controller, depending on the physical conditions, the controller *P* or *PI* would be released, and the appropriate control variables would be set.

```
1   module M_T1 start m_T1_control_P{
      task t_T1_P input(c_double h1) state()
           output(c_double u1) function f_P_1 wcet 100;
      task t_T1_PI input(c_double h1) state()
           output(c_double u1) wcet 150;
6
      mode m_T1_control_P period 500{
        invoke t_T1_P input((h1,1)) output((u1,4));
        switch(isP_2_PI1(e1, e3, s1)) m_T1_control_PI;
      }
11
      mode m_T1_control_PI period 500 program P_T1_PI_ref{
        invoke t_T1_PI input((h1,1)) output((u1,4));
        switch(isPI_2_P1(e1, e3, s1)) m_T1_control_P;
      }
16  }

    program P_T1_PI_ref{
      module M_T1_PI_ref start m_T1_PI_fast{
        task t_T1_PI_fast input(c_double h1) state()
21             output(c_double u1)
```

```
        function f_PI_fast_1 wcet 150;
        task t_T1_PI_slow input(c_double h1) state()
            output(c_double u1)
        function f_PI_slow_1 wcet 100;
26
        mode m_T1_PI_fast period 500{
          invoke t_T1_PI_fast input((h1,1)) output((u1,4))
                parent t_T1_PI;
          switch(isSlow_PI_T1(h1)) m_T1_PI_slow;
31      }

        mode m_T1_PI_slow period 500{
          invoke t_T1_PI_slow input((h1,1)) output((u1,4))
                parent t_T1_PI;
36        switch(isFast_PI_T1(h1)) m_T1_PI_fast;
        }
      }
    }
```

Listing 1. Partial HTL Code for the Three Tanks System

Demand computation. The resource demand bound function ($\mathrm{dbf}_\Omega(t)$) of a RTC model Ω upper bounds the amount of computational resource required to meet the deadlines of all the released jobs in an interval t. This computation is done over tasks that are both released and have their deadlines within the interval. The request bound function (rbf_Ω) of a RTC model Ω, upper bounds the amount of resource demand released in a time interval. The rbf computation takes into account the demand of all the tasks that are released in the interval, including those tasks whose deadlines are outside the interval.

We first introduce the following class of RTC model and explain how efficient demand computation is possible for the class.

Definition 4 (Isochronicity)**.** *A RTC model Ω is* isochronous *if $\forall v^i, v^j \in V^F, \gamma(run(v^0,v^i,t)) + \rho(v^i,v^0) = \gamma(run(v^0,v^j,t)) + \rho(v^j,v^0)$. In this case, the smallest t for which this condition is true, is called the period of Ω. In all other cases, Ω is* anisochronous.

For isochronous RTC tasks, the following technique can be used to compute the dbf value. First, we enumerate and tabulate the dbf_Ω values for every run which has at most one instance of root location in it. In the general case, i.e., for runs with more than one instance of root in them, the run can be broken down into three phases. (1) a $run(v^i,v^0,t_1)$ which ends in a root (v_0) s.t. $t_1 < P_\Omega$, (2) a phase consisting of repeated instances of a run from v_0 to a leaf and back with maximum demand and total duration t_2, and (3) a $run(v^0,v^j,t_3)$ such that the run begins at v_0 s.t. $t_3 < P_\Omega$ and $t_1+t_2+t_3 = t$.

Given t, the duration of the middle phase is at least $\left(\left\lfloor \frac{t}{P_\Omega}\right\rfloor - 1\right) P_\Omega$ and at most $\left\lfloor \frac{t}{P_\Omega}\right\rfloor P_\Omega$ where P_Ω is the period of recurrence of Ω. In the latter case, either $t_1 = 0$ or $t_3 = 0$. We can therefore compute the maximum demand for the overall interval as,

$$\mathrm{dbf}_\Omega(t) = \left(\left\lfloor \frac{t}{P_\Omega}\right\rfloor - 1\right) E_\Omega + \max\left\{E_\Omega + \mathrm{dbf}_\Omega\left(t - \left\lfloor \frac{t}{P_\Omega}\right\rfloor P_\Omega\right), \mathrm{dbf}_\Omega\left(t - \left\lfloor \frac{t}{P_\Omega}\right\rfloor P_\Omega + P_\Omega\right)\right\} \quad (2)$$

where $E_\Omega = \arg\max_{r \equiv run(v^0,v^0,P_\Omega)} \Delta(r)$. A similar result also holds for rbf_Ω. We would also like to mention that the above demand computation technique is similar in flavor to that proposed for for the recurring branching task model (see (Baruah 2003; Anand et al. 2008)) .

For anisochronous RTC tasks, computing $\mathrm{dbf}_\Omega(t)$ can be proved to be NP-hard via a reduction from the *Integer Knapsack* problem in the general case. In this case, an upper bound of the demand can be computed using approximation algorithms. For the rest of the paper, unless otherwise stated, we will assume that the tasks are all isochronous to simplify the presentation.

2.2 Schedulability analysis

Consider a system consisting of n RTC tasks along with their priorities. The problem is to determine if the system can be scheduled using a fixed priority scheduler. An approach similar to that used for recurring branching tasks (Baruah 2003) can be used here. First, given a priority assignment for tasks, schedulability can be decided by considering n problems of determining whether a task is lowest priority viable. The following theorem can then be used for checking whether a task is lowest priority viable.

Theorem 1. *Let $T = \{\Omega_1, \ldots, \Omega_n\}$ be a system of RTC tasks that are preemptively scheduled on a uniprocessor using static priorities with the resource being provided according to a resource model Γ. Task Ω_i is lowest priority viable in T if*

$$\forall t \in TS : \exists t' \leq t : \left[\left(\mathrm{sbf}_\Gamma(t') - \sum_{\Omega \in T\setminus\{\Omega_i\}} \mathrm{rbf}_\Omega(t')\right) \geq \mathrm{dbf}_{\Omega_i}(t)\right] \quad (3)$$

where $t' \geq 0$ and $TS = \left\{t \mid 0 \leq t < \frac{3\cdot\sum_{\Omega\in T} E_\Omega}{1-\sum_{\Omega\in T} E_\Omega/P_\Omega}\right\}$, E_Ω being the maximum resource demand along a loop with largest demand starting from v^0, ending at v^0 and passing through exactly one leaf, i.e., $E_\Omega = \arg\max_r \Delta(r)$ where $r \equiv run(v^0,v^0,P_\Omega)$. P_Ω is the period of recurrence of the task.

Proof. Similar to the proof of Theorem 3, (Baruah 2003), using $\mathrm{sbf}_\Gamma(t)$ for the supply in t. □

The following theorem states the schedulability condition under dynamic priority scheduling.

Theorem 2. *Let $T = \{\Omega_1, \ldots, \Omega_n\}$ be a system of RTC tasks that are preemptively scheduled on a uniprocessor using dynamic priorities with a resource supply model Γ. System T is feasible if and only if,*

$$\forall t \in TS : \sum_{\Omega_i \in T} \mathrm{dbf}_{\Omega_i}(t) \leq \mathrm{sbf}_\Gamma(t) \quad (4)$$

where $t' > 0$ and $TS = \{t \mid 0 < t < \frac{2\cdot\sum_{\Omega\in T} E_\Omega}{1-\sum_{\Omega\in T} E_\Omega/P_\Omega}\}$, E_Ω and P_Ω are as defined in Theorem 1.

Proof. Similar to the proof of Theorem 1, (Baruah 2003), using $\mathrm{sbf}_\Gamma(t)$ for the supply in t. □

3. Sustainability of schedulability analysis

3.1 Results on sustainability of analysis

In this section, we discuss the sustainability of schedulability analysis for a task set comprising of RTC tasks $\Omega_1, \ldots, \Omega_n$. We assume that the task set is executed using a fixed priority algorithm. The sustainability of analysis performed assuming dynamic priorities is similar in flavor, and we omit it due to space constraints.

Before we present the analysis, we introduce some notation. For the remainder of the section, we denote the tasks with modified values of parameters by Ω'_i, the task set with these tasks by T', and the limit to which we have to check the schedulability condition as given by Theorem 1, $\frac{3\cdot\sum_{\Omega\in T} E_\Omega}{1-\sum_{\Omega\in T} E_\Omega/P_\Omega}$, by B. The existence of such a limit is important to bound the schedulability checking of a task set. We also denote the schedulability condition in Theorem 1, $\left[\left(\mathrm{sbf}_\Gamma(t') - \sum_{\Omega\in T\setminus\{\Omega_i\}} \mathrm{rbf}_\Omega(t')\right) \geq \mathrm{dbf}_{\Omega_i}(t)\right]$, by $SC(\Gamma,T,\Omega_i,t',t)$.

In the remainder of this section, we seek to prove that $\forall t, \exists t' \leq t : SC(\Gamma,T,\Omega_i,t',t) \Rightarrow \forall t : \exists t' \leq t : SC(\Gamma,T',\Omega'_i,t',t)$ under different modifications to the task set. The viability of task Ω_i then follows from these series of implications. $\forall t \leq B : \exists t' \leq t : SC(\Gamma,T,\Omega_i,t',t) \Rightarrow \forall t : \exists t' \leq t : SC(\Gamma,T,\Omega_i,t',t) \Rightarrow \forall t : \exists t' \leq t : SC(\Gamma,T',\Omega'_i,t',t)$ The last inequality implies that Ω'_i is lowest priority viable.

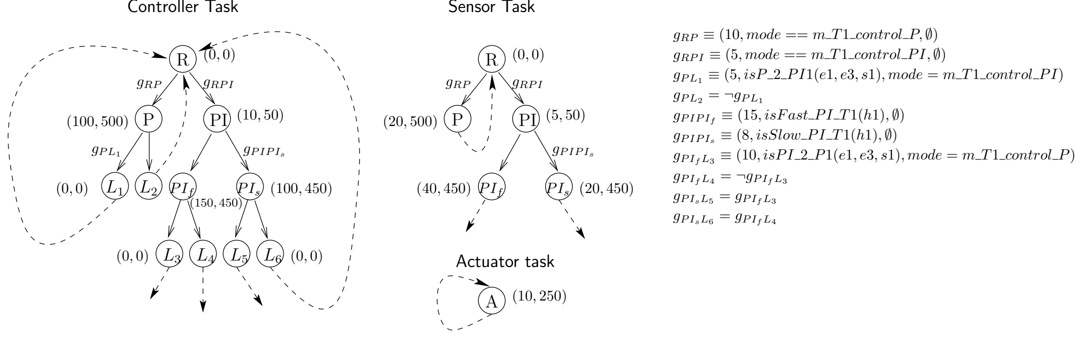

Figure 2. RTC task models the 3TS system

Lemma 1. *Schedulability analysis of conditional real-time tasks with the processor demand criteria under fixed priority scheduling is sustainable with respect to lower execution times.*

Proof. We prove the result considering that the execution times of some tasks in the T decrease, while others remain the same. If the execution times are smaller than what is considered in the analysis, the dbf and rbf values will be lower, i.e., $\forall t : \text{dbf}_{\Omega_i}(t) \leq \text{dbf}_{\Omega_i'}(t)$ and $\forall t : \text{rbf}_{\Omega_j}(t) \leq \text{rbf}_{\Omega_j'}(t)$. If the execution times of Ω_i decrease, then in Equation 3, the RHS will be smaller than what was considered in the analysis, and the inequality would still hold. If the execution times of any other task decreases, the rbf could be smaller and the LHS would increase. In either case, if Equation 3 was true for some t', it will still be true with a task set involving decreased execution times, therefore proving the existence of such a t' for every t in the modified task set. We conclude that the analysis is sustainable with respect to execution times. $\square$

Lemma 2. *Schedulability analysis of conditional real-time tasks with the processor demand criteria under fixed priority scheduling is sustainable with respect to extended relative deadlines.*

Proof. We omit the proof due to space constraints. The idea is similar to the proof of Lemma 1 i.e., to show that dbf cannot increase when relative deadlines are extended. $\square$

Lemma 3. *Schedulability analysis of conditional real-time tasks with the processor demand criteria under fixed priority scheduling is sustainable with respect to greater job inter arrival times.*

Proof. If the inter-arrival times increase, in the general case, the model could become anisochronous as a result of different length paths. We show that the analysis is still sustainable by proving that if a task is modified by increasing the inter-arrival times, the new dbf and rbf values for the modified task is upper bounded by that of the original task and the modified system is lowest priority viable, if the original system is lowest priority viable.

We prove the result considering that the inter-arrival times are extended for some task $\Omega \in T$, such that the overall period of recurrence changes, while other tasks remain the same. The general result of a set of tasks changing their inter-arrival times can be obtained by repeatedly applying the result of just one task changing its inter-arrival times, one at a time. Let us denote the original and modified RTC model by Ω and Ω' respectively. Consider an interval of length $t' > 0$. Let run r' correspond to $\text{dbf}_{\Omega'}(t')$. Since Ω and Ω' differ only in the increased inter-arrival times in the latter, consider a run r in Ω which has all the locations of run r'. Let the length of r be t. Observe that $t \leq t'$ and the demand of the both r and r' is the same. Therefore, we can say that $\text{dbf}_\Omega(t') \geq \text{dbf}_{\Omega'}(t')$ by the monotonicity of dbf function (i.e., $\text{dbf}(t_1) \geq \text{dbf}(t_2)$, whenever $t_1 \geq t_2$). A similar property can also be established in the case of rbf_Ω. This implies that the dbf and rbf values for the modified task are upper bounded by those of the original task.

Now we prove that the modified system is lowest priority viable, if the original system is lowest priority viable. If for every value of t, there exists a $t' < t$ such that the Equation 3 holds for the original system, it holds for the modified system with the same t'. This is so because, as shown above, the dbf and rbf values in the modified system are upper bounded by those of the original system. This could only mean that the LHS of Equation 3 is only greater, and the RHS smaller in the modified system, and therefore if the inequality holds true for some t', in the original task set, it holds true in the modified system as well. We can therefore conclude that if the original task is schedulable, so is the modified task. Therefore, analysis in this case is sustainable. $\square$

Lemma 4. *Schedulability analysis of conditional real-time tasks with the processor demand criteria under fixed priority scheduling is not sustainable with reordering of jobs in the task.*

Proof. We prove non-sustainability of reordering of jobs by giving a counterexample. Consider a system with one task $\Omega :\rightarrow (1,2) \xrightarrow{2} (3,5) \xrightarrow{3}$. For this task, $\text{dbf}_\Omega(2) = 1, \text{dbf}_\Omega(5) = 3$ and $\text{dbf}_\Omega(8) = 4$. Now let us reorder a path in the task to make $\Omega' :\rightarrow (3,5) \xrightarrow{3} (1,2) \xrightarrow{2}$. In the reordered task, we can see that $\text{dbf}_\Omega(5) = 4$. As the dbf increases, we can no longer be sure that the Equation 3 holds, and the analysis is therefore not sustainable. $\square$

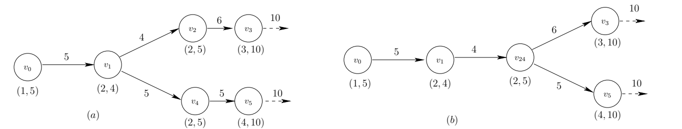

Figure 3. Figure illustrating non-sustainability of job hoisting.

Lemma 5. *Schedulability analysis of conditional real-time tasks with the processor demand criteria under fixed priority scheduling is not sustainable with respect to hoisting (i.e., moving a common job out of conditional branches).*

Proof. Moving a common job out of conditional branches immediately outside does not introduce or eliminate runs. If the inter arrival times change, the dbf values could increase and the analysis would not be sustainable with respect to this operation. As an example, consider that task in Figure 3(a). For the task, $\text{dbf}(9) = 3$. let us say job v_2 and v_4 are identical and are hoisted out as shown in Fig. 3(b). For the modified task, $\text{dbf}(9) = 4$. The analysis is therefore, not sustainable. $\square$

If, however, the inter-arrival times are the same, then the dbf values cannot increase, and the analysis would be sustainable. In Sec 4, when we develop robust techniques for schedulability, we discuss how to handle hoisting, including when the model becomes anisochronous as in Fig.3(b).

Lemma 6. *Schedulability analysis of conditional real-time tasks with the processor demand criteria under fixed priority scheduling is not sustainable with splitting of a job for optimizing it.*

Proof. We prove non-sustainability of job splitting by giving a counterexample. Consider a system with one task $\Omega : \rightarrow (6,10) \xrightarrow{10}$. Observe that $\text{dbf}_\Omega(5) = 0, \text{dbf}_\Omega(10) = 6$. Let us say the job is split to become $\Omega' : \rightarrow (2.5,5) \xrightarrow{5} (2.5,5) \xrightarrow{5}$. Although $\text{dbf}_\Omega(10) = 5$ is lower than before, we now have $\text{dbf}_\Omega(5) = 2.5$ which may break Equation 3. Figure 4 shows the impact of job splitting for the example with a supply that schedules Ω but not Ω'. We conclude

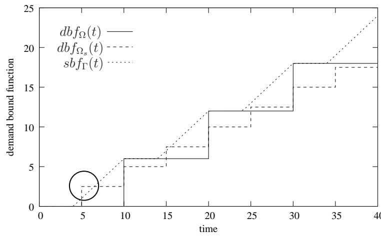

Figure 4. Figure illustrating non-sustainability of job splitting.

that the analysis is not sustainable. □

We summarize the results of all the above lemmas into the following theorem.

Theorem 3. *Processor demand based analysis for conditional real-time task models under fixed priority scheduling is sustainable with respect to lower execution times, extended relative deadlines and greater inter-arrival times, but is not sustainable with respect to structural optimizations such as job hoisting, reordering of jobs or job splitting.*

Proof. Direct, from Lemmas 1-6. □

3.2 Discussion: compiler optimizations and sustainability.

Compiler optimizations on actual code that affect execution times or memory accesses can be mapped to one or more categories considered for sustainability analysis. The sustainability of the operations, however, depend on correlation between the operation and its impact on the task model. We briefly discuss sustainability for some of the common optimizations below.

Common Subexpression Elimination (CSE). In this operation, the compiler searches for instances of identical expressions (i.e., all evaluate to the same value), and analyses whether it is worthwhile replacing them with a single variable holding the computed value. CSE comes in two flavors – local and global. If this operation is local to a job, then the original analysis is sustainable, as this only reduces the execution times and the analysis is sustainable by Lemma 1. Global CSE may render the original analysis not sustainable as it usually involves introducing some instructions initially which can be seen as a form of job splitting (Lemma 6).

Copy/Constant propagation. In this case, the occurrences of targets of direct assignments are replaced by their values. Typically, this results in an reduction of execution times for individual jobs, which implies that the original analysis is sustainable.

Dead code elimination. In this process, the program size is reduced by removing code which does not affect the program. If this is internal to a job, then the analysis is not affected. If one or more jobs get eliminated in the model due to dead code elimination (such as removal of Nop() instructions), then there are a few cases to consider. If the variable assignment is considered in computation of the dbf, i.e., path feasibility given an assignment of variables, then the operation has no effect on the analysis. If feasibility of a path was not considered in the dbf computation, then elimination of jobs could potentially affect the dbf values. If inter-arrival times between jobs are unaffected by this operation, then the original analysis is sustainable by Lemma 3. The analysis will not be sustainable if the inter-arrival times decreases.

Code hoisting. This operation involves moving computations outside of conditional branches to reduce code size. Code hoisting also refers to moving computation outside of a loop so as to save on computing time. In the first case, if the operation does not result in jobs being reordered, and the reordering does not decrease the inter-arrival times between jobs, then the analysis is sustainable (Lemma 5). For the latter definition of code hoisting, it depends on how the jobs are modeled. If the operation is modeled within a job, then it only results in decreased execution times for that job, and therefore the original analysis is sustainable. If code hoisting involves introducing a new job, then the analysis is not sustainable (Lemma 6).

Reduction in strength. Under this optimization, a function of some changing variable is calculated more efficiently by using previous values of the function. If this operation is modeled within a job, then the analysis is sustainable (Lemma 1). If it involves introducing a new job (for initialization statements) then the analysis is not sustainable (Lemma 6).

Loop peeling. In this operation, a loop is either simplified or dependencies eliminated by breaking it into multiple loops which iterate over different contiguous portions of the index range. The operation can be analyzed in the same manner as code hoisting.

4. Robust schedulability analysis

In this section, we focus on robust schedulability analysis techniques. First, we discuss the problem intuitively, then we formulate the robust analysis as an optimization problem with fixed execution times, and then finally, we consider robust analysis with increased execution times.

4.1 Motivation for robust analysis

We have seen in the previous section that task optimizations such as job reordering and job splitting are not sustainable, i.e., if the analysis is performed, and then the task is subject to such optimizations, then the original analysis need not necessarily hold. We remedy the problem here by developing a framework that takes as input, the space of all anticipated changes to the task and outputs the maximum possible task load under those modifications. The idea is that, if this maximum load can be supported by resource supply, then the analysis will be sustainable under any actual modification that happens to the task.

To model changes in a task, we assume that the task parameters can take values from within a constraint set. For instance, we consider values to be specified over a range of real numbers with some additional affine constraints. We expect that this model represents many realistic scenarios, such as a task where there is some variable jitter between the arrival of jobs, or a task subject to optimizations such as job splitting or reordering. Given such a task, the problem is to analyze schedulability so that no matter what the actual values of task parameters are, so long as they are within the constraints specified, the task set will remain schedulable.

For example, consider the task specified in Fig. 5(a). Let us say that the jobs at nodes v_2 and v_3 have a segment that can be split into a job with execution requirement 1 unit, and hoisted beyond node v_1. Further, suppose the deadline and inter-arrival times for the new job are not known at the time of analysis, but it is known that they preserve the overall timing behavior of the task. In this case, we

associate the split job $(1,d_4) \xrightarrow{j_4}$ to a new node v_4, and identify the constraints on inter-arrival times as $j_4+j_3=15, j_4+j_2=15$ and $j_4 \geq 0, j_2 \geq 0, j_3 \geq 0$. As the application preserves the overall timing behavior, we also expect that deadlines d_2, and d_3 are upper bounded by the old deadline values, and the deadline d_4 is also upper bounded by the maximum of deadlines of jobs released at v_2 and v_3. We therefore have, $10 \geq d_4 \geq 3, 10 \geq d_2 \geq 4, 10 \geq d_3 \geq 5$. The scenario is explained in Fig. 5(b).

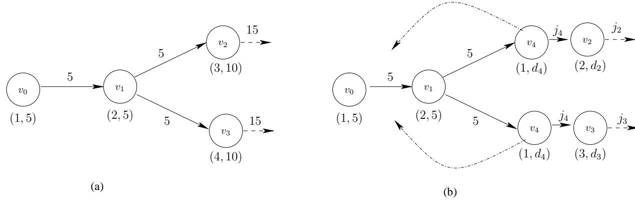

Figure 5. Example task which undergoes optimization

4.2 Robust analysis with fixed execution times

In this section, we discuss the robust analysis with fixed execution times. The case where execution times change is dealt in the next section.

Our approach to address the robust analysis problem in the case when execution times remain the same, is to compute the maximum task load, i.e, $L_\Omega = \max_t \frac{\text{dbf}_\Omega(t)}{t}$ from the space of all anticipated changes to inter-arrival times and deadlines of tasks due to code changes such as job splitting and reordering. After we get this maximum load, we will use it to upper bound dbf_Ω for the task with modified parameter values, i.e., $\text{dbf}_\Omega(t) \leq L_\Omega \cdot t$. Specifically, we obtain the maximum task load by solving an optimization problem on the parameters of Ω. We show that under the formulation, the optimization problem can be solved as a series of convex optimization sub-problems. We now present each of the these steps in detail.

Setting up the optimization problem. For any interval t, the load presented by task Ω is defined to be $\frac{\text{dbf}_\Omega(t)}{t}$. The objective of the optimization problem is to find an assignment to the parameter values from within the constraint set so that $\frac{\text{dbf}_\Omega(t)}{t}$ is maximized for any t. The first problem we have is to compute the interval up to which we have to check for obtaining the maximum task load. Given a RTC task Ω, Lemma 7 gives us the bound on the load.

Lemma 7. $\max_t \frac{\text{dbf}_\Omega(t)}{t} = \max\left\{\frac{E_\Omega}{P_\Omega}, \max_{r \equiv run(v^i,v^j,t')} \frac{\Delta(r)}{t'}\right\}$ *for any RTC task Ω, where r is a run of Ω with at most one instance of root in it, $E_\Omega = \arg\max_{r \equiv run(v_i^0,v_i^0,P_\Omega)} \Delta(r)$, and P_Ω being the period of recurrence of task Ω.*

Proof. First observe that

$$\max_{t \leq P_\Omega} \frac{\text{dbf}_\Omega(t)}{t} = \max\left\{\frac{E_\Omega}{P_\Omega}, \max_{r \equiv run(v^i,v^j,t')} \frac{\Delta(r)}{t'}\right\} \quad (5)$$

is true as r includes all intervals $t \leq P_\Omega$. Now recall the dbf computation procedure for the general case from Sec. 2.1. For $t \geq P_\Omega$,

$$\text{dbf}_\Omega(t) = \left(\left\lfloor \frac{t}{P_\Omega} \right\rfloor - 1\right) E_\Omega + \max\left\{E_\Omega + \text{dbf}_\Omega\left(t - \left\lfloor \frac{t}{P_\Omega} \right\rfloor P_\Omega\right), \text{dbf}_\Omega\left(t - \left\lfloor \frac{t}{P_\Omega} \right\rfloor P_\Omega + P_\Omega\right)\right\} \quad (6)$$

where $E_\Omega = \arg\max_{r \equiv run(v^0,v^0,P_\Omega)} \Delta(r)$. Denoting $t - \left\lfloor \frac{t}{P_\Omega} \right\rfloor P_\Omega$ as t_1, we can observe that,

$$\frac{\max\{E_\Omega + \text{dbf}_\Omega(t_1), \text{dbf}_\Omega(t_1+P_\Omega)\}}{t_1+P_\Omega} \leq \max\left\{\frac{E_\Omega}{P_\Omega}, \max_{r \equiv run(v^i,v^j,t')} \frac{\Delta(r)}{t'}\right\} \quad (7)$$

This is true because the run r on the RHS considers all intervals which have one instance of the root in it, and this includes the interval of duration $t_1 + P_\Omega$ in the dbf computation. The result then follows by using $E_\Omega \leq \max\left\{\frac{E_\Omega}{P_\Omega}, \max_{r \equiv run(v^i,v^j,t')} \frac{\Delta(r)}{t'}\right\}$ and the inequality in Eq. 7 in Eq. 6. ☐

Lemma 7 gives us the bound to compute the maximum task load. Based on the observations, the optimization problem can be formulated as follows.

$$\max\left\{\frac{\Delta(r)}{\max\{d_i, \ldots, \sum_{k=i}^{j-1} j_k + d_j\}}\right\}_{r \equiv r(v_i,v_j,t)} \quad (8)$$

subject to,

$$\begin{aligned} &\forall k: \; j_k \geq 0, \;\; \forall i: \; d_i > E_i, \\ &\sum_{r_1} j_k = J_1, \; (0 \leq J_1 \leq P_\Omega), \ldots, \sum_{r_m} j_k = J_m, \; (0 \leq J_m \leq P_\Omega), \\ &j_0 \in [L_{01}, U_{01}] \ldots j_n \in [L_{n-1\;n}, U_{n-1\;n}], \\ &d_1 \in [L_1, U_1] \ldots d_n \in [L_n, U_n] \end{aligned} \quad (9)$$

where a job at node v^i is taken to be $(E_i,d_i) \xrightarrow{j_i}$, where E_i, and d_i denote the execution requirements, and deadline for the job, and j_i the minimum time before releasing the next job. In the formulation, we have adopted the notation that all variables are denoted by small letters and all constants by capital letters. We also use the index $r(v_i,v_j,t)$ to represent a run of Ω which has at most one instance of the root location in it or a run from the root to a leaf and back. The index r_i goes over runs of the Ω which involve the variable inter-arrival times. At the most, they could go over all runs that either does not involve a reset transition, or if they do, then the run terminates at the root location.

Here are a few observations about the problem formulation. (1) If there are $|V|$ locations in Ω, we would have at most $|V|^3$ terms in the objective function. This is so because runs with at most one root location can be uniquely determined by specifying a start location, leaf, and a terminal location. (2) We consider deadlines greater than execution times. Otherwise, the system can be in full load, and the optimization problem would trivially return by choosing this as the maximum load. (3) We assume that the execution times are known a priori, and cannot be left as variables. This assumption is necessary to keep the objective function (Eq.8) convex. We will elaborate on this when we discuss the technique to solve the optimization problem. (4) Although we consider the deadlines to take values from an interval, we are not dealing with soft real-time systems. It is assumed that once an assignment of deadline is made, it is a hard deadline.

The constraints expressed in Eq. 9 capture for instance, variable inter-arrival times or deadlines. In addition, it can also capture operations such as job splitting and reordering of jobs.

Job splitting. Let us say that a job $(E_i,D_i) \xrightarrow{J_i}$ corresponding to location v_i can be split into m sub-jobs, with execution times $E_{i_1}, \ldots, E_{i_m}$. We assume $E_{i_1}, \ldots, E_{i_m}$ to be known constants. For this case, we introduce m new and consecutive locations $v_{i_1}, \ldots, v_{i_m}$ and remove the existing location v_i from the task. In other words, we introduce the job sequence $(E_{i_1},d_{i_1}) \xrightarrow{j_{i_1}} \ldots \xrightarrow{j_{i_{m-1}}} (E_{i_m},d_{i_m}) \xrightarrow{j_{i_m}}$ into the task instead of $(E_i,D_i) \xrightarrow{J_i}$. Once this is specified, we can set up the optimization problem as described and add the following additional constraints. $\forall k: \; d_{i_k} > E_{i_k}$ and $\forall k: d_{i_k} \leq D_i$ for the deadlines, and $\forall k: \; j_{i_k} \geq 0$ and $\sum_k j_{i_k} = J_i$ for the inter-arrival times.

Reordering of jobs. Let us say that job $(E_i,D_i) \xrightarrow{J_i}$ can be moved to m possible new locations during the reordering of jobs in the task. We create m locations $v_{i_1}, \ldots, v_{i_m}$ in addition to the original one v_i . The job assigned for the new locations are $(e_{i_1},d_{i_1}) \xrightarrow{j_{i_1}}$

$, \ldots, (e_{i_m}, d_{i_m}) \stackrel{j_{i_m}}{\rightarrow}$ respectively, where $\forall k : e_{i_k} = E_i,\ j_{i_k} = J_i$, and $d_{i_k} = D_i$. The constraints can now be set up as described before. The objective function (Eq. 8) in this case is modified in the following manner. The function considers all the runs as before, including the runs with each of the locations $v_{i_1}, \ldots, v_{i_m}$. A run involving two or more of the locations in $\mathcal{V}_i = \{v_{i_1}, \ldots, v_{i_m}\}$, however, considers the impact of only the last one. This means that we consider a demand of only E_i in a run involving one or more locations from the set $\mathcal{V}_i$, and an inter-arrival time of J_i for the last one. The rationale behind this is that, when a job is moved, we want to consider its worst possible impact on the load. We account for the extra demand introduced due a potential move by considering all runs involving locations in $\mathcal{V}_i$. For runs involving more than one potential location, we only need to account this extra demand once, as only one job is being moved. The choice of the last one to be considered is because, in a run involving more than one location from $\mathcal{V}_i$, the inter-arrival time of last one will impact the deadlines of subsequent jobs the least, thereby posing the maximum load.

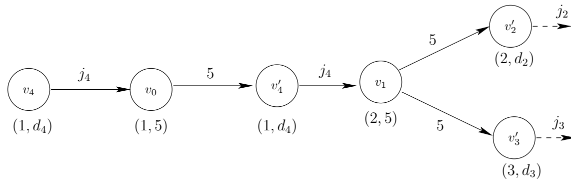

Figure 6. Modeling the job split and reordering

As an example, consider the task described earlier in Fig. 5. The job split and reordering could result in the split job being moved to either before v_0, or after v_0. We model these scenarios in Fig. 6 by adding two locations, i.e., v_4 and v_4' in the figure. We can write the optimization problem for the example as discussed above. Listed below is the objective function with runs involving up to three locations.

$$\max\left\{\frac{1}{d_4}, \frac{1}{5}, \frac{2}{5}, \frac{3}{d_3}, \frac{2}{d_2}, \frac{2}{\max\{d_4, j_4+5\}}, \frac{2}{5+d_4}, \frac{3}{\max\{d_4, j_4+5\}}, \frac{4}{5+d_2}, \frac{3}{\max\{d_2, j_2+d_4\}}, \frac{3}{\max\{d_2, j_2+d_4\}}, \frac{4}{\max\{d_3, j_3+d_4\}}, \ldots\right\} \tag{10}$$

subject to,

$$\begin{aligned} &j_4 + j_3 = 15, j_4 + j_2 = 15 \\ &10 \geq d_4 \geq 3, 10 \geq d_2 \geq 4, 10 \geq d_3 \geq 5 \\ &j_4 \geq 0, j_2 \geq 0, j_3 \geq 0 \end{aligned} \tag{11}$$

Common job hoisting. If a common job can be hoisted out of the conditional branch, and the minimum inter arrival time for that job is different on different conditions, considering this minimum would result in an anisochronous model. For example, hoisting job in location v_2 and v_4 makes the task in Fig. 3 anisochronous. To make the analysis robust, it is sufficient to model the task after hoisting, and reduce the inter-arrival time between the common job moved out and the subsequent job so that the transformation preserves isochronicity. For the example in Fig. 3 (b), the inter-arrival times between jobs v_{24} and v_3 to be 5.

Solving the optimization problem. The second step of the analysis involves solving for the maximum load posed by the task as a convex optimization problem. This is achieved by considering the reciprocal of objective function in Equation 8. Observe that the objective function is maximized whenever the reciprocal of objective function attains the minimum. This is true because all the variables in the system are in the denominator of Equation 8. The new objective function is therefore,

$$\min\left\{\frac{\max\{d_i, \ldots, \sum_{k=i}^{j-1} j_k + d_j\}}{\Delta(r)}\right\}_{r(v_i, v_j, t)} \tag{12}$$

Since the affine function $ax+b$ for $a, b, x \in \mathbb{R}$ is convex, and the sum of two convex function is convex, all the terms inside the maximum of the objective function (e.g., $\sum_{k=i}^{j-1} j_k + d_j$) are convex. Further, the point wise maximum of convex functions is also convex, making the objective function $\frac{\max\{d_i, \ldots, \sum_{k=i}^{j-1} j_k + d_j\}}{\sum_{r(v_i, v_j, t)} e_i}$ convex. The constraints in Eq. 9 are also convex making the overall problem of finding the maximum load, a standard convex optimization problem. The problem can therefore be solved using standard techniques (see Chebyshev approximation, Boyd and Vandenberghe (Boyd and Vandenberghe 2004), pg 293).

We would like to emphasize that the assumption that execution times are fixed is important for solving the problem. If the execution times are left as variables, then the objective function would have been of the linear fractional form, i.e., $f(x) = \frac{a^T x + b}{c^T x + d}$, with $c^T x + d > 0$. The linear fractional function is, however, only quasi-convex. A quasi-convex function optimization problem can have local optima that are not globally optimal, a property that is important for us to be able to find the maximum system load. The assumption that execution times are constant is consistent with the worst case upper bounds of execution times used in practice. Arguments similar to those in Lemma 1 can be used to prove that robust schedulability analysis is sustainable with lower execution times, thereby establishing that using the worst case upper bounds is sound.

With the objective function set up as in Eq. 12, it can be solved by solving a series of minimax sub-problems, and then taking the minimum over all of them. This follows from Lemma 8.

Lemma 8. *Given functions $f_1(x)$ and $f_2(x)$ over $x \in \mathbb{R}^n$, then* $\min_x\{f_1(x), f_2(x)\} = \min\{\min_x f_1(x), \min_x f_2(x)\}$.

Once we have solved for the maximum load in Eq. 8, we can relate it to schedulability of task as follows.

Theorem 4. *Given a RTC task Ω, where the task parameters can take values subject to the affine system of constraints in Eq. 9, we have $\forall t, \frac{\mathrm{dbf}_\Omega(t)}{t} \leq L_\Omega$, where L_Ω is the optimal value of the load as returned by the solution of system Eq. 8.*

Proof. Direct, from Lemma 7. □

We add that the result in Theorem 4 is tight in the sense that there exists an assignment to the task parameters satisfying the constraints of Eq. 9 so that the load of u_Ω is achieved exactly.

For the example task in Fig.5, we have formulated the optimization problem in Eq. 10. Taking the reciprocal of the objective function and reducing, we get the following set of minimax objective functions to be solved subjects to the constraints in Eq. 11: $\min\max\{d_4, j_4+5, j_4+5+d_2\}$, $\min\max\{d_4, j_4+5, j_4+5+d_3\}$, $\min\max\{d_2, j_2+d_4, j_2+j_4+5\}$, and $\min\max\{d_3, j_3+d_4, j_3+j_4+5\}$ Since we have constraints $j_4+j_3 = 15, j_4+j_2 = 15$, the minimum value of last function above is 20. Now solving for all the variables, we find that the optimal assignment is $d_4 = 3, d_2 = 4, d_3 = 5$ and $j_4 = 0, j_2 = 15, j_3 = 15$. The maximum task load is $\frac{3}{5}$. Therefore, the task will be schedulable if the supply can support at least this much load at all times, irrespective of the splitting and reordering of jobs within the task.

We conclude this section by discussing robust schedulability analysis of the task set using the above framework. Let us say that for tasks $\Omega_1, \ldots, \Omega_n$, we compute maximum values of load $L_{\Omega_1}, \ldots, L_{\Omega_n}$, as per their constraints. A similar result can also be computed with rbf's instead of dbf$'s$. Let us denote $\max_t \frac{\mathrm{rbf}_{\Omega_i}(t)}{t}$

thus computed by R_{Ω_i}, for $i = 1,\dots,n$. The robust schedulability of the task set can then be defined using these quantities.

Theorem 5. *Let $T = \{\Omega_1,\dots,\Omega_n\}$ be a system of RTC tasks that are preemptively scheduled on a uniprocessor using static priorities with the resource being provided according to a resource model Γ. Task Ω_i is lowest priority viable in T if*

$$\forall t \in TS : \exists t' \leq t : \left[\left(\mathrm{sbf}_\Gamma(t') - \sum_{\Omega \in T \setminus \{\Omega_i\}} R_\Omega \cdot t'\right) \geq L_{\Omega_i} \cdot t\right] \quad (13)$$

where the E_Ω, P_Ω and TS are as defined before.

The proof of the above theorem is direct from Theorem 1 and the observation that $\mathrm{dbf}_\Omega(t) \leq L_\Omega \cdot t$ and $\mathrm{rbf}_\Omega(t) \leq R_\Omega \cdot t$. The analysis in Theorem 5 is sustainable with any values of task parameters constrained as in Eq. 9. A similar result can also be stated for the dynamic priority case.

4.3 Robust analysis with increased execution times

Thus far, we have discussed a technique to compute the maximum task load in the case where task parameters other than execution times can take values from within a real space with affine constraints. Here we consider the case where the execution time increases by a common factor for the task set, while other parameters are fixed. This view is consistent with many practical scenarios, including applications running on systems which use dynamic voltage scaling of processors, and applications which are meant to be ported on systems with different target processors. In fact, Yerraballi et al (Yerraballi et al. 14-16 Jun 1995) have discussed at length about mapping many different questions on schedulability of systems to the problem of finding the maximum scaling factor for the task set.

The main issue in trying to address the problem for the RTC task set is that the schedulability criteria depends on the average task load. If the execution times increase, the interval up to which we need to check for schedulability also increases, and therefore the original analysis cannot be directly used. This is the case under both fixed priority, and dynamic priority scheduling (Theorems 1 and 2). Rest of the section discusses how to get around this problem for scheduling feasibility with scaled execution times.

Let the common scaling factor be denoted by α. For scheduling feasibility, Eq. 4, Theorem 2 gives the necessary and sufficient condition. If the execution times increase by a constant factor, the dbf for any interval is also scaled up by that factor, i.e., $\mathrm{dbf}_{\Omega'}(t) = \alpha \cdot \mathrm{dbf}_\Omega(t)$, where Ω is the original task, and Ω' is the task with scaled execution times. We now present a series of results that we can use to compute the maximum common scaling factor.

For the purposes of analysis, we define the following bound on a resource supply, and prove a few properties of the task model.

Definition 5. *We define the* linear lower bound function (lsbf) *of a resource supply Γ as a linear function of t which has the following property : $\forall t \geq 0 :\ \mathrm{lsbf}_\Gamma(t) \leq \mathrm{sbf}_\Gamma(t)$, and $\exists\, 0 \leq t_1 < t_2 :\ \mathrm{lsbf}_\Gamma(t_1) = \mathrm{sbf}_\Gamma(t_1)$ and $\mathrm{lsbf}_\Gamma(t_2) = \mathrm{sbf}_\Gamma(t_2)$.*

A linear upper bound of the supply function (usbf) can be defined similarly.

Lemma 9. *Let $T = \{\Omega_1,\dots,\Omega_n\}$ be a system of RTC tasks that are preemptively scheduled on a uniprocessor using dynamic priorities with the resource being provided according to a resource model Γ.*

1. *$\arg\max_t \frac{\sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t)}{t} = \arg\max_t \frac{\sum_{\Omega'_i} \mathrm{dbf}_{\Omega'_i}(t)}{t}$ where Ω'_i is the task with the scaled execution times.*
2. *Let t^* is such that $\mathrm{lsbf}_\Gamma(t^*) = \frac{\sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t_m)}{t_m} \cdot t^*$, where $t_m = \arg\max_t \frac{\sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t)}{t}$. If $t^* \leq d_{min}$, d_{min} being the smallest relative deadline in any task of T, then T is schedulable with Γ.*

Proof. (1) Proposition 9.1 follows directly from the following observation. $\forall t > 0 :\ \frac{\sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t_m)}{t_m} \geq \frac{\sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t)}{t}$
and the fact that $\mathrm{dbf}_{\Omega'_i}(t) = \alpha \cdot \mathrm{dbf}_{\Omega_i}(t)$, α being a constant,

(2) Observe that for $t < d_{min}$, $\sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t) = 0$ and $\mathrm{sbf}_\Gamma(t) \geq \sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t)$ holds trivially. Observe that t^* is the point of intersection of lsbf_Γ and the line $\frac{\sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t_m)}{t_m} \cdot t$. The latter line passes through the origin, and $\mathrm{lsbf}_\Gamma(0) \geq 0$. Therefore, in either case, for $t > t^*$, $\mathrm{sbf}_\Gamma(t) \geq \mathrm{lsbf}_\Gamma(t) \geq \frac{\sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t_m)}{t_m} \cdot t \geq \frac{\sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t)}{t} \cdot t$. For both $t < t^*$ and $t \geq t^*$, we have $\mathrm{sbf}_\Gamma(t) \geq \sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t)$ and T is schedulable by Theorem 2. □

Lemma 9 sets us up for finding the common scaling factor. Proposition 9.1 says that the point in time which represents the highest load is preserved during scaling. Therefore, the scaling factor for the task set is quite simply how much the execution times of the point with maximum load in the task set (t_m) can be scaled without affecting schedulability.

The next question to be answered is how to check for schedulability efficiently in the scaled task set. Observe that the limit to which we have to check schedulability depends on load (Theorem 2), therefore, if the execution times scale, the limit also increases. Proposition 9.2 tries to get around this problem by giving a sufficient criteria for schedulability that can be checked efficiently. The proposition says that if the slope of the lsbf_Γ is greater than the slope of the demand function at the point of highest load of the task set, then the task set is schedulable. More specifically, denoting the point of intersection of the lsbf_Γ with the line $\frac{\sum \mathrm{dbf}_{\Omega_i}(t_m)}{t_m} t$ by t^*, the proposition says that if $t^* < d_{min}$, then the task set is schedulable. If t^* happens to be greater than d_{min}, then we need to check $\forall t < t^*$ that $\mathrm{sbf}_\Gamma(t) \geq \sum \mathrm{dbf}_{\Omega_i}(t)$. Clearly, $\mathrm{sbf}_\Gamma(t) \geq \sum \mathrm{dbf}_{\Omega_i}(t)$ holds for all $t \geq t^*$. These concepts are illustrated in Figure 7.

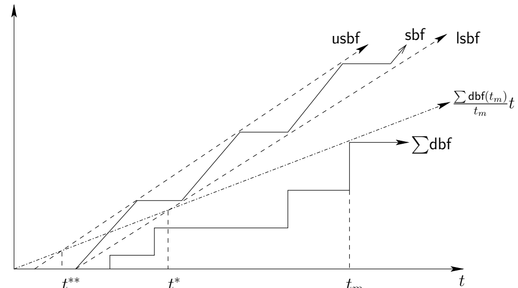

Figure 7. Figure illustrating various concepts of Theorem 9

The final question that remains to be answered is that of computing the point of maximum load (t_m) for a task set. Using Theorem 2 with full supply ($\mathrm{sbf}(t) = t$), we get $t_m \leq \frac{2 \cdot \sum_{\Omega \in T} E_\Omega}{1 - \sum_{\Omega \in T} E_\Omega / P_\Omega}$ if the original task set is schedulable.

For a general resource supply Γ, the following theorem then gives a method to compute the scaling factor.

Theorem 6. *Let $T = \{\Omega_1,\dots,\Omega_n\}$ be a system of RTC tasks that are preemptively scheduled on a uniprocessor using dynamic priorities with a resource supply Γ. The scaling factor of the system T is at least α, where α is defined as,*

$$\alpha = \min\left\{\min_{0<t<t^*} \frac{\mathrm{sbf}_\Gamma(t)}{\sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t)}, \frac{\mathrm{usbf}_\Gamma(d_{min}) \cdot t_m}{d_{min} \sum_{\Omega_i} \mathrm{dbf}_{\Omega_i}(t_m)}\right\} \quad (14)$$

where t_m is the point with the maximum task load.

Proof. For $t > t^*$, $\mathrm{lsbf}_\Gamma(t) \geq \sum_{\Omega_i} \alpha \cdot \mathrm{dbf}_{\Omega_i}(t)$. To ensure schedulability, we need to ascertain that the inequality holds for $t < t^*$. However, t^* depends on the choice of α. To get the result of the

corollary, we set the point of intersection between the usbf_Γ and the line $u = \frac{\sum_{\Omega_i} \alpha \cdot \text{dbf}_{\Omega_i}(t_m)}{t_m} \cdot t$ as d_{min}, thereby fixing t^*. The scaling factor is then simply the minimum of the slope of the line u the value obtained by checking each of the points $t < t^*$. □

For the full processor supply, $t^* = 0$, and the supply function is itself its linear lower bound. Therefore, we can get the exact scaling factor, as $\frac{t_m}{\sum_{\Omega_i} \text{dbf}_{\Omega_i}(t_m)}$, where t_m is is the point where the task set poses the maximum load on the system.

5. Conclusions

In this paper, we have introduced and addressed sustainability and robustness of schedulability analysis for systems modeled as recurring branching tasks. We have noted that the analysis is sustainable with respect to many parameters such as lower job execution times, increased job inter-arrival times, and relaxed deadlines, Structural changes such as job splitting and reordering are not sustainable, even though they can result in lower overall execution times. For such operations, we have developed a robust schedulability analysis framework, that can be used to model and analyze the schedulability, and the results of this analysis are sustainable.

References

A.K.Mok and Wing-Chi Poon. Non-preemptive robustness under reduced system load. *Real-Time Systems Symposium, 2005. RTSS 2005. 26th IEEE International*, pages 10 pp.–, 5-8 Dec. 2005. ISSN 1052-8725. doi: 10.1109/RTSS.2005.31.

K. Altisen, G. Gössler, and J. Sifakis. Scheduler modeling based on the controller synthesis paradigm. *Real-Time Syst.*, 23(1-2):55–84, 2002. ISSN 0922-6443.

Madhukar Anand, Arvind Easwaran, Sebastian Fischmeister, and Insup Lee. Compositional feasibility analysis for conditional task models. In *Proceedings of the Eleventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'08)*, Washington, DC, USA, 2008. IEEE Computer Society.

J. Andersson, B.; Jonsson. Preemptive multiprocessor scheduling anomalies. *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 12–19, 2002. doi: 10.1109/IPDPS.2002.1015483.

Sanjoy Baruah and Alan Burns. Sustainable scheduling analysis. In *RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium*, pages 159–168, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2761-2. doi: http://dx.doi.org/10.1109/RTSS.2006.47.

Sanjoy K. Baruah. Dynamic- and static-priority scheduling of recurring real-time tasks. *Real-Time Systems*, 24(1):93–128, 2003. ISSN 0922-6443. doi: http://dx.doi.org/10.1023/A:1021711220939.

Sanjoy K. Baruah. A general model for recurring real-time tasks. In *RTSS*, pages 114–122, 1998. URL `citeseer.ist.psu.edu/baruah98general.html`.

Hanene Ben-Abdallah, Jin-Young Choi, Duncan Clarke, Young Si Kim, Insup Lee, and Hong-Liang Xie. A process algebraic approach to the schedulability analysis of real-time systems. *Real-Time Syst.*, 15(3): 189–219, 1998. ISSN 0922-6443. doi: http://dx.doi.org/10.1023/A:1008047130023.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004. ISBN 0521833787.

Alan Burns and Sanjoy Baruah. Sustainability in real-time scheduling. *Journal of Computing Science and Engineering*, Vol 2(No. 1):72–94, 2008.

G. Buttazzo. Achieving scalability in real-time systems. *Computer*, 39(5): 54–59, May 2006. ISSN 0018-9162. doi: 10.1109/MC.2006.148.

G. C. Buttazzo and J. Stankovic. Red: A robust earliest deadline scheduling algorithm. In *Proceedings of Third International Workshop on Responsive Computing Systems*, 1993. URL `citeseer.ist.psu.edu/buttazzo93red.html`.

Ya-Shu Chen, Li-Pin Chang, Tei-Wei Kuo, and Aloysius K. Mok. Real-time task scheduling anomaly: observations and prevention. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 897–898, New York, NY, USA, 2005. ACM. ISBN 1-58113-964-0. doi: http://doi.acm.org/10.1145/1066677.1066881.

A. Davis, R. I.; Burns. Robust priority assignment for fixed priority real-time systems. *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 3–14, 3-6 Dec. 2007. ISSN 1052-8725. doi: 10.1109/RTSS.2007.43.

X. Feng and A. Mok. A model of hierarchical real-time virtual resources. In *Proc. of IEEE Real-Time Systems Symposium*, pages 26–35, December 2002.

Christian Ferdinand, Reinhold Heckmann, Marc Langenbach, Florian Martin, Michael Schmidt, Henrik Theiling, Stephan Thesing, and Reinhard Wilhelm. Reliable and precise wcet determination for a real-life processor. In *EMSOFT '01: Proceedings of the First International Workshop on Embedded Software*, pages 469–485, London, UK, 2001. Springer-Verlag. ISBN 3-540-42673-6.

Arkadeb Ghosal, Alberto Sangiovanni-Vincentelli, Christoph M. Kirsch, Thomas A. Henzinger, and Daniel Iercan. A hierarchical coordination language for interacting real-time tasks. In *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*, pages 132–141. ACM Press, 2006. ISBN 1-59593-542-8.

Daniel Iercan. TSL Compiler. Master's thesis, Politehnica University of Timisoara, September 2005.

Sheayun Lee, Insik Shin, Woonseok Kim, Insup Lee, and Sang L. Min. A design framework for real-time embedded systems with code size and energy constraints. *(To appear) ACM Transactions on Embedded Computing Systems (TECS)*, 2008.

J.Y.T. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic real-time tasks. *Performance Evaluation*, 2:37–250, 1982.

G. Lipari and E. Bini. Resource partitioning among real-time applications. In *Proc. of Euromicro Conference on Real-Time Systems*, July 2003.

S.P. Marlowe, T.J.; Masticola. Safe optimization for hard real-time programming. *Systems Integration, 1992. ICSI '92., Proceedings of the Second International Conference on*, pages 436–445, 15-18 Jun 1992. doi: 10.1109/ICSI.1992.217244.

Razvan Racu and Rolf Ernst. Scheduling anomaly detection and optimization for distributed systems with preemptive task-sets. In *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 325–334, 2006. ISBN 0-7695-2516-4.

Ha Rhan and J.W.S. Liu. Validating timing constraints in multiprocessor and distributed real-time systems. *Distributed Computing Systems, 1994., Proceedings of the 14th International Conference on*, pages 162–171, 21-24 Jun 1994. doi: 10.1109/ICDCS.1994.302407.

I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *Proc. of IEEE Real-Time Systems Symposium*, pages 2–13, December 2003.

R. Yerraballi, R. Mukkamala, K. Maly., and H.A. Wahab. Issues in schedulability analysis of real-time systems. *Real-Time Systems, 1995. Proceedings., Seventh Euromicro Workshop on*, pages 87–92, 14-16 Jun 1995. doi: 10.1109/EMWRTS.1995.514297.

M.F. Younis, Marlowe T.J., Tsai G., and Stoyenko A.D. Toward compiler optimization of distributed real-time processes. *Engineering of Complex Computer Systems, 1996. Proceedings., Second IEEE International Conference on*, pages 35–42, 21-25 Oct 1996. doi: 10.1109/ICECCS.1996.558328.