Real-time Systems Journal manuscript No.
(will be inserted by the editor)

The Limited-preemptive Feasibility of Real-time Tasks on Uniprocessors

Abhilash Thekkilakattil · Radu Dobrin · Sasikumar Punnekkat

Received: date / Accepted: date

Abstract The preemptive scheduling paradigm is known to strictly dominate the non-preemptive scheduling paradigm with respect to feasibility. On the other hand, preemptively scheduling real-time tasks on uniprocessors, unlike non-preemptive scheduling, may lead to unschedulability due to, e.g., preemption related overheads. The limited-preemptive scheduling paradigm, which is a generalization of preemptive and non-preemptive paradigms, has, however, the potential to reduce the preemption related overheads while enabling high processor utilization.

In this paper, we focus on the characterization of the effects of increasing the computational resources on the limited-preemptive feasibility of real-time tasks in order to quantify the sub-optimality of limited-preemptive scheduling. Specifically, we first derive the required processor speed-up bound that guarantees limited-preemptive feasibility of any uniprocessor feasible taskset. Secondly, we demonstrate the applicability of the results in the context of controlling preemption related overheads while minimizing the required processor speed-up. In particular, we identify the preemptive behavior that minimizes preemption-related overheads, as well as derive the optimal processor speed associated with it. Finally, we examine the consequences of having more processors on limited-preemptive feasibility and derive the bound on the number of processors that guarantees a specified limited-preemptive behavior for any uniprocessor feasible real-time taskset.

A. Thekkilakattil
Mälardalen University, Box 883, 72123, Sweden
tel: +4621101689
E-mail: abhilash.thekkilakattil@mdh.se

R. Dobrin
Mälardalen University, Box 883, 72123, Sweden
tel: +4621107356
E-mail: radu.dobrin@mdh.se

S. Punnekkat
Mälardalen University, Box 883, 72123, Sweden
tel: +4621107324
E-mail: sasikumar.punnekkat@mdh.se

This paper essentially bridges the preemptive and non-preemptive real-time scheduling paradigms by providing significant theoretical results building on the limited-preemptive scheduling paradigm, as well as provides analytical inputs to developers in order to perform various trade-offs, e.g., code refactoring, to control the preemptive behavior of real-time tasks.

Keywords Real-time Scheduling · Limited Preemption Feasibility · Resource Augmentation · Sensitivity Analysis

1 Introduction

Modern processors support performance enhancing features such as caches and instruction pipelines that pre-fetch data and instructions to speed-up computations. The adoption of these types of processors in real-time computing requires a careful analysis of the worst case scenarios, that the specialized hardware may introduce, in order to provide hard real-time guarantees. For example, preemptively scheduling real-time tasks can result in high preemption related overheads that may lead to unschedulability. Preemptive scheduling requires capabilities to suspend the currently running task, and perform a context switch, in favor of higher priority tasks. The time required to perform context switches, if significantly high, can lead to deadline misses in the schedule. Similarly, when a preempted task resumes its execution, cache lines may need to be reloaded due to potential loss of cache affinity. The delay incurred to reload cache lines can be considerably high, leading to an increase in utilization by as much as 33% (Bui et al (2008)), eventually causing deadlines to be missed. Besides context switch costs and cache related preemption delays, preemptions may increase bus contention due to frequent off-chip memory accesses, as well as may require clearing and refilling of the instruction pipelines, all of which manifest as temporal overheads affecting the schedulability of the system.

Since the seminal paper by Liu and Layland (1973), real-time scheduling theory has matured to a point where a fairly large set of fundamental questions regarding preemptive uniprocessor scheduling have been sufficiently addressed. The feasibility analysis (e.g., Baruah et al (1990a)) and schedulability analysis (e.g., Audsley et al (1991)) of preemptive real-time tasks typically assume negligible preemption related overheads. Whenever the overheads are not negligible, they are assumed to be accounted for in the worst case execution times of the tasks. Extending some of these works, many methods were proposed to account for the preemption related overheads in the schedulability analysis (e.g., Busquets-Mataix et al (1996); Lee et al (1998); Tan and Mooney (2007); Staschulat et al (2005); Altmeyer et al (2012); Ju et al (2007)). Ward et al (2014) build on the best of these works and present an improved preemption overhead accounting paradigm. However, all the preemptions considered by the above mentioned works may not occur in the actual schedule leading to analysis pessimism.

On the other hand, as pointed out by Short (2010), non-preemptive scheduling is often favored for applications with severe resource constraints due to its low memory requirements, and simple implementation. However, non-preemptive scheduling has received less attention as compared to preemptive scheduling since the works

by George et al (1996), George et al (1995) and Jeffay et al (1991). The facts that non-preemptive scheduling can be infeasible even at arbitrarily low processor utilizations due to blocking on higher priority tasks (Yao et al (2010)), and the strict domination of preemptive scheduling over non-preemptive scheduling (Baruah (2005)) may have contributed towards the limited efforts in addressing the feasibility of non-preemptively scheduling real-time tasks.

One way to address the blocking related infeasibility under non-preemptive scheduling is to adopt the limited-preemptive scheduling paradigm (a detailed survey of which is available in Buttazzo et al (2012)). Under the limited-preemptive scheduling paradigm, the preemptive behavior of real-time tasks is controlled by reducing the number of preemptions and/or by restricting them to pre-determined points in the code. The need to limit preemptions in real-time systems is well recognized by both academia and industry. Buttle (2012), in his keynote, indicated that limited preemptive scheduling is widely favored in the automotive industry where data-intensive real-time tasks consisting of non-preemptable blocks are required to be cooperatively scheduled. In this case the schedulability test must determine whether these non-preemptable blocks can be scheduled without causing deadline misses. If the taskset is unschedulable, it means that at least one of the non-preemptable blocks causes significantly high blocking leading to deadline misses in the schedule.

A majority of the modern processors supports Dynamic Voltage and Frequency Scaling (DVFS) using which the tasks' Worst Case Execution Time (WCET) can be manipulated by changing the CPU frequency, and is typically used to slow down task executions to conserve power (Pillai and Shin (2001)). However, some recent experiments (Saha and Ravindran (2012)) indicate that keeping the processor at idle state for longer duration after completing tasks sooner by running at higher frequencies can save more power. As noted by Thiele (2014), processor frequencies can be increased to improve schedulability in real-time systems under specified power and thermal constraints. Likewise, it is possible to speed-up the processor to ensure that the largest non-preemptive regions of the tasks are 'large' enough to guarantee a specified limited-preemptive behavior in the schedule, e.g., execute large non-preemptable blocks without causing deadline misses. On the other hand, if thermal and power constraints make speeding-up infeasible, more number of processors can be used to guarantee a specified limited preemptive behavior by distributing the tasks among the processors or by parallelizing the code. The widespread availability of multicore processors makes this option particularly attractive.

Resource augmentation, first introduced by Kalyanasundaram and Pruhs (2000), is widely used to quantitatively compare the performance of scheduling algorithms in terms of the extra resources required to achieve optimality— the most commonly considered resource being the processor speed. While Fixed Priority Scheduling (FPS) scheme is not uniprocessor optimal (Liu and Layland (1973)), there exit many scheduling schemes such as preemptive Earliest Deadline First (EDF), that are known to optimally schedule real-time tasks on uniprocessors (Dertouzos (1974)). Consequently, previous works quantified the performance of FPS with respect to optimal scheduling schemes, under both preemptive and non-preemptive paradigms, using resource augmentation. All of these efforts, started by Baruah and Burns (2008) and later continued by Davis et al (2009a), Davis et al (2010) and Davis et al (2009b),

focused specifically on processor speed-up and did not examine the possibility of using more number of processors such as done by Lam and To (1999). Moreover, inspite of the fact that preemptive scheduling strictly dominates limited- and non-preemptive scheduling in feasibly scheduling real-time tasks (Baruah (2005)), none of these efforts were directed towards quantifying their sub-optimality (using neither faster processors nor more number of processors).

In this paper, we fill this gap by quantifying the sub-optimality of limited-preemptive scheduling in terms of a) bound on the processor speed-up, and b) bound on the number of processors, required to guarantee a desired limited-preemptive behavior. The results derived in this paper bound the 'cost' to be paid in order to enable a desired limited-preemptive behavior in terms of the computational resources.

In the following, the main contributions of this paper are summarized, while detailing the differences from Thekkilakattil et al (2013):

1. In this paper, we generalize the speed-up bounds by considering the more realistic execution time model proposed by Marinoni and Buttazzo (2007), instead of the fully linear model assumed in Thekkilakattil et al (2013), as well as, present a tighter bound (a two fold improvement) for the restricted model assumed in Thekkilakattil et al (2013).
   - If only a fraction ϕ of the task execution times scales with processor speed, the speed-up bound S that guarantees limited-preemptive feasibility is given by $S \leq \left(1 + \frac{1}{\phi}\right)$ in many cases.
   - We present an improved result over Thekkilakattil et al (2013), and show that, if the entire execution time of the tasks scales with processor speed, the speed-up bound S that guarantees non-preemptive execution of all tasks for a duration no greater than L_{max} is given by

     $$S \leq 2 \max\left(1, \frac{L_{max}}{D_{min}}\right)$$

     The main result presented in Thekkilakattil et al (2013) was greater than the above result by a factor of 2.
   - We evaluate the speed-up required to guarantee a fully non-preemptive schedule using randomly generated tasksets (which is not done in Thekkilakattil et al (2013)), after presenting the two step sensitivity analysis method for preemption control from Thekkilakattil et al (2013).

2. We derive the bound on the number of processors required (Lam and To (1999)) to guarantee a specified limited-preemptive behavior of *uniprocessor feasible* real-time tasks, which is not considered in Thekkilakattil et al (2013), allowing us to consider a different dimension regarding the feasibility of limiting preemptions.
   - In general, the number of processors is shown to be upper-bounded by the number of tasks in the taskset.

– In the specific case, in which the largest length of the specified limited-preemptive regions in the taskset is no more than half the shortest deadline, the number of processors is shown to be upper-bounded by 3.

Organization: We first present the system model in Section 2, and then review some relevant results in Section 3. We derive the speed-up bounds in Section 4, after which we present a methodology to calculate the minimum speed-up that guarantees a specified limited-preemptive behavior, along with an evaluation, in Section 5. We then derive the upper-bound on the number of processors required to guarantee a specified limited preemptive behavior in Section 6, followed by a discussion on the derived resource augmentation bounds in Section 7, before concluding in Section 8.

2 System model

In this section, we introduce the notations used in the rest of the paper whilst describing the task model, scheduling model, and the execution time model.

2.1 Task model

We consider a set of sporadic real-time tasks $\Gamma = \{\tau_1, \tau_2, ...\tau_n\}$, where each τ_i is characterized by a minimum inter-arrival time T_i, a worst case execution time C_i^S at processor speed S, and a relative deadline D_i. We assume that tasks' worst case execution times are equal to their worst case execution requirements on a speed $S = 1$ processor. Let the length of the longest critical section of a task τ_i, on a processor of speed S, be denoted by CS_i^S. We assume that the tasks are indexed according to the increasing order of their deadlines, which means that $D_{min} = D_1$. We assume that every task τ_i has m_i optimal preemption points (Peng et al (2014)) within its execution, where the m_i^{th} point denotes the end of the task execution. Let $q_{i,j}^S$, $j = 1...m_i$ denote the length of the execution of τ_i from its start up to the j^{th} optimal preemption point on a processor at speed S. In order to focus on the theoretical consequence of resource augmentation on the preemptive behavior of the taskset, and for the sake of clarity of presentation, we assume negligible preemption related overheads at these optimal preemption points. However such an assumption does not affect the generality of our results because the preemption related overheads can be accounted during the placement of the preemption points, e.g., as done by Bertogna et al (2010).

Let β_i^S denote the blocking tolerance of τ_i, which is the largest time for which τ_i can be blocked without causing any deadline miss (Bertogna et al (2010)). Also, let $B_i^S \leq \beta_i^S$ denote the largest time for which τ_i is effectively blocked at run-time. LCM denotes the Least Common Multiple of the time periods of all the tasks in the set. The utilization U_i^S of a task τ_i executing on a processor at speed S is defined as $U_i^S = \frac{C_i^S}{T_i}$ and the utilization of the entire taskset is given by $U^S = \sum_{i=1}^{n} U_i^S$. The demand bound function of a task τ_i, on a processor of speed S, during a time interval $[0,t)$ is given

by Baruah et al (1990b),

$$DBF_i^S(t) = \max\left(0, 1 + \left\lfloor \frac{t - D_i}{T_i} \right\rfloor\right) C_i^S$$

For example, $DBF_i^1(t)$ denotes the cumulative processor time requested by τ_i during a time interval $[0,t)$ on a processor of speed $S = 1$. Additionally, the density of task τ_i is given by $\delta_i = \frac{C_i^1}{D_i}$, and the total density of the taskset by $\delta_{tot} = \sum_{\forall \tau_i} \delta_i$. We define a specified limited-preemption length as follows.

Definition 1 A **specified limited-preemption length** of a task τ_i is defined as the maximum specified length of the non-preemptive regions of τ_i.

For example, a specified limited-preemption length may guarantee a specified upper-bound on the preemption related cost on τ_i that guarantees schedulability. We denote the specified limited-preemption length of a task τ_i at speed S by L_i^S. A specified limited-preemption length can also be denoted by L_i in case it does not change with the processor speed.

Definition 2 A **limited-preemption requirement** on a task τ_i is defined as the requirement that the task τ_i executes non-preemptively for a duration given by the specified limited-preemption length.

A *limited-preemption requirement* on any task τ_i is said to be feasible if τ_i can execute non-preemptively for the specified limited-preemption length, consequently guaranteeing the *feasibility of the specified limited-preemptive behavior* for τ_i.

Definition 3 The **feasibility of a specified limited-preemptive behavior** of a taskset is defined as the existence of a real-time schedule that guarantees the non-preemptive execution of every task for the specified limited-preemption length, while ensuring the absence of deadline misses in the schedule.

The *feasibility of the specified limited-preemptive behavior* of the taskset can be guaranteed by speeding up the processor to control the length of the non-preemptive regions.

2.2 Scheduling model

We assume the Earliest Deadline First (EDF) scheduling paradigm. We assume that, whenever a higher priority job is released during the execution of a lower priority job of τ_i, instead of immediately preempting the job of τ_i, the scheduler blocks the higher priority job for Q_i^S time units on a processor of speed S. Alternately, the tasks can be composed of several non-preemptable chunks of code, whose maximum length is Q_i^S. Here, Q_i^S is the length of the largest non-preemptive region of τ_i derived from the task attributes (Baruah (2005); Bertogna and Baruah (2010)). Consequently, the maximum number of times the task τ_i can be preempted when a processor of speed S is used, is given by $\left\lceil \frac{C_i^S}{Q_i^S} \right\rceil - 1$. If $L_i^S > Q_i^S$, then the system is not schedulable since

the specified limited-preemption length is greater than the bound on the maximum possible limited-preemption length.

We assume a work conserving scheduler, i.e., the scheduler does not idle the processor when there are active tasks awaiting the processor. We leverage on the optimality of EDF (Dertouzos (1974); Jeffay et al (1991)) to study the processor speed-up required to guarantee the feasibility of a required limited-preemptive behavior of real-time tasks.

2.3 Execution time model

In this paper we focus mainly on the theoretical consequences of resource augmentation, specifically processor speed-up, on the preemptive behavior of real-time tasks. We adopt the execution time model proposed by Marinoni and Buttazzo (2007). In this model, the execution time of each task consists of two parts— one that is processor speed dependent and the other that is processor speed independent. Let ϕ_i denote the fraction of execution time of τ_i that scales linearly with the processor speed. Consequently, the fraction $(1-\phi_i)$ of the execution time of τ_i does not scale with the processor frequency.

To ease the readability, and without loss of generality, we assume that the taskset is initially executing on a processor of speed $S=1$. We assume that, if C_i^1 is the execution time at speed $S=1$, the task execution time of τ_i scales as follows:

$$C_i^S = \frac{\phi_i C_i^1}{S} + (1-\phi_i)C_i^1$$

Such a model also allows us to use processor speed-up factors and processor speeds interchangeably. Changing the processor speed from $S=1$ to $S=a$, is equivalent to speeding up the processor by a factor of 'a'. Finally, we define

$$\phi = \min_{\forall \tau_i \in \Gamma} (\phi_i)$$

Therefore, in any time interval t, *at least* $\phi \sum_i^n DBF_i^1(t)$ units of execution scales with the processor speed.

3 Feasibility analysis of real-time systems— a short review

The limited preemptive scheduling model proposed by Baruah (2005) can be seen as generalizations of non-preemptive and preemptive scheduling models as they can simulate a preemptive behavior ranging from non-preemptive to fully preemptive. If Q_i^S is set equal to 0 for all τ_i, the system simulates a fully preemptive model, while if Q_i^S is set equal to C_i^S, the system simulates a fully non-preemptive model (Yao et al (2010)). In our approach we build on Baruah's (Baruah (2005)) model to study the feasibility of preemptive, non-preemptive, and limited-preemptive scheduling of real-time tasks, when the amount of available resources change

Let us now recall some previously published theoretical results presented by Jeffay et al (1991) (in Section 4 of their paper) and Bertogna et al (2010) (in Section

IV of their paper). Due to sustainability of the EDF scheduling scheme (Baruah and Burns (2006)), these theorems can be generalized to a processor of speed S, $(S \geq 1)$. A real-time taskset is feasible if the cumulative processor time requested by the set of tasks during any time interval does not exceed the size of that time interval (Baruah et al (1990b)). The following theorems presented in a revised uniform format, determines the feasibility of uniprocessor real-time scheduling.

Theorem 1 *(from Bertogna et al (2010)) A taskset is feasible on a speed S processor, if and only if,* $\forall i \in [1,n]$,

$$\beta_i^S \geq 0$$

where, β_i^S *is given by*

$$\beta_i^S = \min_{D_i \leq t < D_{i+1}} \left(t - \sum_{j=1}^{n} DBF_j^S(t) \right) \tag{1}$$

$$t = kT_j + D_j, \forall k \in N, j \in [1,n]$$

In the above theorem, D_{n+1} is set as

$$D_{n+1} = \min(LCM, P)$$

Where,

$$P = \max \left\{ D_1, D_2, ..., D_n, \frac{\sum_{i=1}^{n} (T_i - D_i) U_i^S}{1 - U^S} \right\}$$

When the $\beta_i^S = 0, \forall i \in [1,n]$, the taskset is feasible only under a fully preemptive scheduling scheme.

The above theorem can be used to determine the feasibility of limited-preemptive scheduling on a processor at speed S, and is stated by the following theorem.

Theorem 2 *(from Bertogna et al (2010)) A taskset is feasible under limited-preemptive scheduling on a speed S processor if,* $\forall i \in [1,n]$,

$$B_i^S \leq \beta_i^S$$

where the blocking tolerance β_i^S *is given by equation 1 and* B_i^S *is the largest blocking actually experienced by* τ_i *due to the limited preemptions on a processor of speed S.*

The bound Q_k^S on the length of the non-preemptive region of a task τ_k on a processor of speed S is given by the following theorem.

Theorem 3 *(from Bertogna et al (2010)) A taskset is feasible under limited-preemptive scheduling on a speed S processor if,* $\forall k \in [1,n]$,

$$Q_k^S = \min_{1 \leq i < k} \beta_i^S$$

The task can execute entirely non-preemptively if Q_k^S is greater than or equal its execution time C_k^S. Hence, we can use the above theorem to state the non-preemptive feasibility of the taskset, i.e., whether it is possible to find a non-preemptive schedule, as follows:

Theorem 4 *(from Jeffay et al (1991)) A taskset is feasible under non-preemptive scheduling on a speed S processor if,* $\forall k \in [1,n]$,

$$C_k^S \leq Q_k^S$$

4 Speed augmentation for limited-preemptive scheduling

In this section, we examine the consequences of having a faster processor on the limited preemptive scheduling of real-time tasks. In modern processors, due to effects of e.g., the memory wall (McKee (2004)), the entire task execution times may not scale linearly with the processor speed. We relax the assumption, made in (Thekkilakattil et al (2013)), that the entire task execution times scale linearly with processor speed, and examine the consequences of changing the processor speed on the limited-preemptive feasibility. We consider the generalized execution time model proposed by Marinoni and Buttazzo (2007) (that is more realistic) in which only a part of the WCETs scale linearly with the processor speed. First we show, in the following example, that in general it is not practical to use faster processors to achieve limited-preemptive EDF feasibility. We do this by constructing a taskset for which the use processor speed-up to achieve limited-preemptive feasibility is not practical.

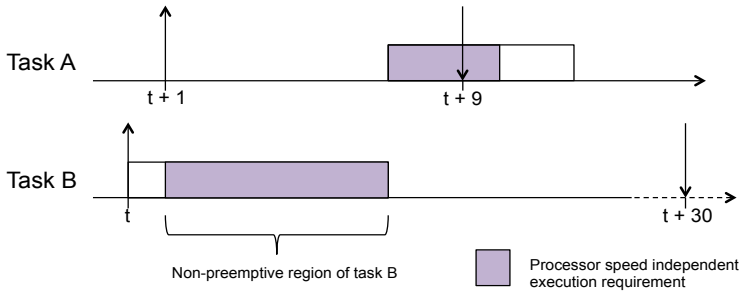


Fig. 1: The schedule for example 1

Example 1 Consider two sporadic tasks A and B, having execution times $C_A^1 = 5$ and $C_B^1 = 10$, $\phi_A = 0.4$ and $\phi_B = 0.4$, and deadlines $T_A = D_A = 8$ and $T_B = D_B = 30$. This means that it is not possible to speed up 3 units of execution of task A and 6 units of execution of task B. Assume that the limited-preemption requirement on task B is 6. Consider the scenario, as shown in Figure 1, when a job of task B is released at time instant t and has immediately started its execution— at time $t+1$ it has finished only 1 unit of execution. If a job of A is released at time $t+1$, clearly A has a higher priority than B. When A tries to preempt B, B immediately starts executing non-preemptively. If task B immediately starts executing the region of code that is independent of the processor speed, clearly the job of task A will miss its deadline. In this case, no amount of speeding up the processor helps because, no matter what the processor speed is, 3 units of execution of task A and 6 units of execution of task B cannot execute faster, and hence the non-preemptive execution of task B for 6 units will lead to a deadline miss on task A at time $t+9$.

The example shows the difficulty of obtaining a speed-up bound for limiting preemptions. However, if the limited-preemption requirement of a task is less than a certain

fraction of the shortest deadline (whose exact value is presented later in this section), we can obtain the resource augmentation bound even if only a part of the execution time scales linearly with the processor speed. We then present an improved result over the one presented in (Thekkilakattil et al (2013)) for the specific case in which it is possible to speed-up the entire task WCET.

Theorem 5 *The processor speed S_i that guarantees the feasibility of a limited-preemption requirement L_i for any task $\tau_i \in \Gamma$ is given by,*

$$S_i = \max_{D_1 \leq t < D_i} \left\{ \frac{\phi \sum_{j=1}^{n} DBF_j^1(t)}{t - L_i - (1-\phi)\sum_{j=1}^{n} DBF_j^1(t)} \right\}$$

Proof The maximum length of the non-preemptive region for τ_i at speed 1 is given by Baruah (2005),

$$Q_i^1 = \min_{D_1 \leq t < D_i} \left\{ t - \sum_{j=1}^{n} DBF_j^1(t) \right\}, \forall t, D_1 \leq t < D_i$$

Our aim is to find the processor speed S_i that guarantees the feasibility of a limited-preemption requirement L_i. Suppose,

$$L_i > Q_i^1 = t - \sum_{j=1}^{n} DBF_j^1(t)$$

We know that, of the total demand bound in any interval t, at least ϕ percentage scales with the processor speed. Thus, we can speed-up this part of the task executions to guarantee the limited preemptive execution of τ_i for L_i units, i.e., $\forall t, D_1 \leq t < D_i$,

$$L_i \leq t - \left\{ \frac{\phi \sum_{j=1}^{n} DBF_j^1(t)}{S_i} + (1-\phi) \sum_{j=1}^{n} DBF_j^1(t) \right\}$$

Hence, $\forall t, D_1 \leq t < D_i$,

$$S_i L_i \leq S_i t - \left\{ \phi \sum_{j=1}^{n} DBF_j^1(t) + S_i(1-\phi) \sum_{j=1}^{n} DBF_j^1(t) \right\}$$

Solving for S_i, we get, $\forall t, D_1 \leq t < D_i$,

$$S_i \geq \left\{ \frac{\phi \sum_{j=1}^{n} DBF_j^1(t)}{t - L_i - (1-\phi)\sum_{j=1}^{n} DBF_j^1(t)} \right\}$$

i.e.,

$$S_i = \max_{D_1 \leq t < D_i} \left\{ \frac{\phi \sum_{j=1}^{n} DBF_j^1(t)}{t - L_i - (1-\phi)\sum_{j=1}^{n} DBF_j^1(t)} \right\}$$

□

Consequently, we find the upper-bound on the required processor speed that guarantees a specified limited-preemption requirement L_i for any task $\tau_i \in \Gamma$.

Lemma 1 *The upper-bound on the minimum processor speed S_i that guarantees the feasibility of a limited-preemption requirement L_i for any task $\tau_i \in \Gamma$, during a time interval t is given by,*

$$S_i \leq \frac{y}{y - \frac{1}{\phi}}$$

where, $y = \frac{t}{L_i}$, $\forall t \in [D_1, D_i)$.

Proof We know from theorem 5 that,

$$S_i = \max_{D_1 \leq t < D_i} \left\{ \frac{\phi \sum_{j=1}^{n} DBF_j^1(t)}{t - L_i - (1-\phi)\sum_{j=1}^{n} DBF_j^1(t)} \right\}$$

Since we have assumed that the taskset is feasible, the upper-bound on the value of $\sum_{j=1}^{n} DBF_j^1(t)$ is t. Hence,

$$S_i \leq \left\{ \frac{\phi t}{t - L_i - (1-\phi)t} \right\} \Rightarrow S_i \leq \frac{t}{\phi t - L_i}$$

Finally, substituting $y = \frac{t}{L_i}$,

$$S_i \leq \frac{\phi y}{\phi y - 1}$$

$$\Rightarrow S_i \leq \frac{y}{y - \frac{1}{\phi}}$$

□

In order to derive the actual value of the upper-bound on the required processor speed that guarantees the feasibility of a limited-preemption requirement L_i, for any $\tau_i \in \Gamma$, during any time interval t we consider the following two cases:

Case 1: $y \geq \left(1 + \frac{1}{\phi}\right)$

Case 2: $0 < y < \left(1 + \frac{1}{\phi}\right)$

In the following lemma, we bound the speed-up required for case 1, i.e., when $y \geq \left(1 + \frac{1}{\phi}\right)$.

Lemma 2 *The upper-bound on the minimum processor speed S_i that guarantees the feasibility of a limited-preemption requirement L_i for any task $\tau_i \in \Gamma$, such that $y \geq \left(1 + \frac{1}{\phi}\right)$, is given by*

$$S_i \leq \left(1 + \frac{1}{\phi}\right)$$

where $y = \frac{t}{L_i}$ and $t \in [D_1, D_i)$

Proof Evaluating the limit of the equation in lemma 1 at $y = \left(1 + \frac{1}{\phi}\right)$, we get,

$$S_i = \left(1 + \frac{1}{\phi}\right)$$

Evaluating the limit using l'Hopital's rule, as y tends to infinity (∞), we get,

$$S_i = 1$$

Therefore, for any value of $y \in [\left(1 + \frac{1}{\phi}\right), \infty]$,

$$S_i \leq \left(1 + \frac{1}{\phi}\right)$$

□

In the case 2 above, in general, it is not practical to use processor speed-up to guarantee a specified limited-preemptive behavior because, in the worst case, the non-preemptive region does not scale with the processor speed (refer Example 1). However, if $\phi = 1$, it is possible to bound the required speed-up. This is derived in the following subsection.

4.1 Speed-up bound under restricted execution time model

In this subsection, for completeness, we consider the special case in which the entire task execution time scales linearly with processor speed considered in Thekkilakattil et al (2013), while achieving a two fold improvement in the speed-up bound over Thekkilakattil et al (2013).

When the entire task execution time scales with the processor speed, i.e., $\phi = 1$, we can restate Lemma 2 as follows:

Lemma 3 *The upper-bound on the minimum processor speed S_i that guarantees the feasibility of a limited-preemption requirement L_i for any task $\tau_i \in \Gamma$, such that $y \geq 2$ and $\phi = 1$, is given by*

$$S_i \leq 2$$

Furthermore, under the assumption that $\phi = 1$, Case 2 presented in the previous section can be split into two sub-cases as follows:

Case 2.a: $1 \leq y < 2$
Case 2.b: $0 < y < 1$

In the following lemma, we derive the speed-up bound for Case 2.a described above:

Lemma 4 *The upper-bound on the minimum speed S_i that guarantees the feasibility of a specified limited-preemption requirement L_i for any task τ_i, such that $1 \leq y < 2$ and $\phi = 1$, is given by*

$$S_i \leq 2$$

where $y = \frac{t}{L_i}$ and $\forall t \in [D_1, D_i)$.

Proof On a unit speed processor t clock ticks are available in any time interval of length t. In the worst case, the processor is fully occupied during t, and hence the limited-preemption requirement L_i cannot be feasibly executed. Let us assume an increase in the processor speed by a factor of 2. This implies that within an interval of time t, there are in effect $t' = 2t$ clock ticks. In this case the limited-preemption requirement L_i can be successfully executed without causing any deadline miss since $2t \geq L_i + t$ (remember that we are considering the case $t \geq L_i$). Therefore, when $1 \leq y < 2$, the tighter upper-bound is given by 2.

Lemma 5 *The upper-bound on the minimum speed S_i that guarantees the feasibility of a limited-preemption requirement L_i for any task τ_i, such that $0 < y < 1$ and $\phi = 1$, is given by*

$$S_i \leq \frac{2L_i}{t}$$

where $y = \frac{t}{L_i}$ and $\forall t \in [D_1, D_i)$.

Proof On using a processor that is $S = \frac{L_i}{t}$ times faster, the number of clock ticks in the time interval t increases from t to $t' = t \times \frac{L_i}{t} = L_i$. Consequently, we can execute the original demand of length no greater than t, and a part $L_i - t$ of the limited-preemption requirement L_i at speed $S = \frac{L_i}{t}$. The remaining limited-preemption requirement that cannot be executed is $L_i' = t$. On a processor of speed $S = \frac{L_i}{t}$, since $t < L_i$, we effectively have $\frac{t'}{L_i'} = \frac{L_i}{t} > 1$. Using Lemma 3 and Lemma 4, the speed-up required denoted by S_i' is $S_i' \leq 2$. Remember that we had already increased the processor speed by $\frac{L_i}{t}$, therefore, the tighter upper-bound S_i is:

$$S_i \leq \frac{2L_i}{t}$$

In the following, we present a theorem that unifies the results in Lemma 3, Lemma 4 and Lemma 5 to obtain an integrated result on how the limited preemptivity changes with respect to processor speed.

Theorem 6 *The upper-bound on the minimum processor speed S_i that guarantees the feasibility of a limited-preemption requirement L_i for any task $\tau_i \in \Gamma$ when $\phi = 1$ is given by, $\forall t > 0$,*

$$S_i \leq 2 \max\left(1, \frac{L_i}{t}\right)$$

where, $y = \frac{t}{L_i}$.

Proof When $\phi = 1$, the entire execution time of every task scales linearly with the processor speed. We know that L_i is bounded by the maximum of the execution times of the tasks in the taskset, at speed $S = 1$ (i.e., its fully non-preemptive execution). Similarly, since t is lower-bounded by the shortest deadline, we obtain $\frac{t}{L_i} > 0$.

When $y \geq 2$, according to Lemma 3, we obtain:

$$S_i \leq 2 \max\left(1, \frac{1}{y}\right) = 2 \times 1 = 2$$

Similarly, when $1 \leq y < 2$, according to Lemma 4, we obtain

$$S_i \leq 2\max\left(1, \frac{1}{y}\right) = 2 \times 1 = 2$$

Finally, when $0 < y < 1$ the speed-up required, according to Lemma 5, is

$$S_i \leq 2\max\left(1, \frac{1}{y}\right) = \frac{2L_i}{t}$$

Therefore, it follows from lemmas 3, 4, and 5 that the speed-up required is bounded as follows:

$$S_i \leq 2\max\left(1, \frac{L_i}{t}\right)$$

□

The above theorem is valid for any time interval. However, in the following, we show that the largest speed-up is obtained at the shortest relative deadline.

Corollary 1 *The upper-bound on the minimum processor speed S_i that guarantees the feasibility of a limited-preemption requirement L_i for any task $\tau_i \in \Gamma$, when $\phi = 1$, is given by*

$$S_i \leq 2\max\left(1, \frac{L_i}{D_{min}}\right)$$

The proof is intuitive as the value of t for which the blocking from L_i is maximum, is the smallest value of t given by the shortest relative deadline $t = D_{min}$ (remember that $D_{min} = D_1$). It is when $t = D_{min}$ that the value $\frac{L_i}{t}$ is maximized. We have thus derived the upper-bound on the processor speed that guarantees the *feasibility of a specified limited-preemptive behavior* for any task $\tau_i \in \Gamma$. The above result can be extended to derive the bound on the speed-up that guarantees non-preemptive execution of the entire task set Γ for a duration of at least L_{max}, where $L_{max} = \max_{\forall \tau_i} L_i$.

Corollary 2 *The upper-bound on the minimum processor speed S that guarantees the feasibility of the specified limited-preemption requirement L_{max} for any taskset Γ, when $\phi = 1$, is given by*

$$S \leq 2\max\left(1, \frac{L_{max}}{D_{min}}\right)$$

where $L_{max} = \max_{\forall \tau_i \in \Gamma}(L_i)$.

Therefore, the speed-up that guarantees the non-preemptive feasibility of the entire taskset is given by the following.

Corollary 3 *The upper-bound on the minimum processor speed S that guarantees the non-preemptive EDF feasibility of any taskset Γ, when $\phi = 1$, is given by*

$$S \leq 2\max\left(1, \frac{C_{max}}{D_{min}}\right)$$

where $C_{max} = \max_{\forall \tau_i \in \Gamma}(C_i^1)$.

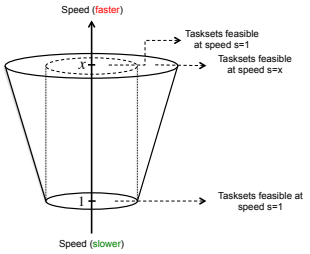


Fig. 2: Speed Vs. feasibility.

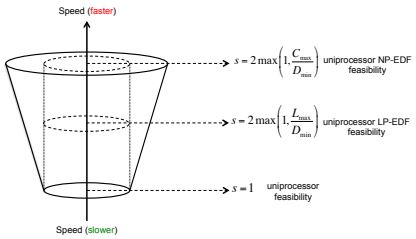


Fig. 3: Speed Vs. LP feasibility.

The executions of non-preemptive chunks of the tasks are independent of each other (Baruah (2005)). Consequently, the processor speed-up bound that guarantees non-preemptive feasibility of the task with largest execution time will also guarantee the non-preemptive execution of the entire taskset.

Sub-optimality: The sub-optimality of limited-preemptive (non-preemptive) scheduling when compared to an optimal uniprocessor preemptive scheduling scheme, with respect to any taskset Γ, can be quantified in terms of processor speed-up bound given in Corollary 2 (Corollary 3).

Resource availability Vs. limited-preemptive feasibility: We illustrate the change in preemptive behavior with respect to processor speed, under the assumption that the entire WCETs scales linearly with the processor speed, in Figure 3. Note that the Figure 3 is an attempt at visually representing the results presented in this paper in an intuitive manner that reflects the dominance of preemptive real-time scheduling over limited- and non- preemptive scheduling.

The base of the bucket in Figure 2 represents the set of *all* uniprocessor feasible real-time tasks on a processor of speed $S = 1$. Obviously, on increasing the processor speed to $S = x$, more tasksets become uniprocessor feasible. Consequently, the original set of tasks that was feasible at speed $S = 1$, becomes a subset of the tasksets feasible at speed $S = x$. At this point, we additionally consider the limited-preemptive EDF feasibility of *all* uniprocessor feasible real-time tasks. Figure 3 illustrates how the limited-preemptive feasibility changes from fully preemptive uniprocessor feasibility at speed $S = 1$ to the fully non-preemptive feasibility at speed $S = 2\max\left(1, \frac{C_{max}}{D_{min}}\right)$.

5 Practical applications of the theoretical results

Many real-time systems consist of data intensive real-time tasks that are cooperatively scheduled, where each task is composed of many non-preemtable chunks of code (Buttle (2012)). If the size of these non-preemptable chunks are significantly large, it may cause large blocking leading to unschedulability. Similarly, many methods (Peng et al (2014)) have been proposed to place preemption points in the task code such that the preemption overheads are minimized. However, if the duration between any

two optimal preemption points is significantly high, it may lead to deadline misses due to blocking. Thiele (2014) in his keynote presented the possibility of speeding up the processor in order to achieve a schedulable system. Equivalently, processor speed-up can be used to control the execution times of the non-preemptable chunks and the processor demand to guarantee that tasks can execute non-preemptively for a specified duration. Thereby, the number of preemptions can be reduced and/or the preemption points can be placed at optimal locations with respect to the preemption costs. In this section, we focus on deriving the minimum processor speed-up that guarantees the specified preemption behavior that minimizes preemption overheads.

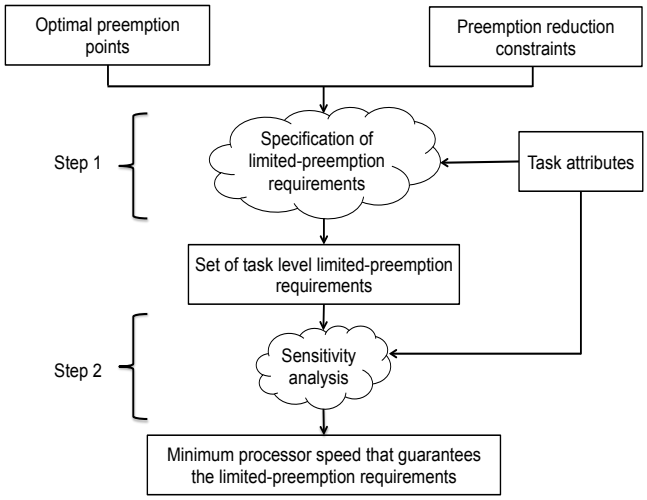


Fig. 4: Methodology overview

While in the previous section we derived the *upper-bound* on the required processor speed-up that guarantees the feasibility of a user specified limited-preemptive behavior, in this section, we apply this bound to enable trade-offs between processor speed and preemption overheads.

Definition 4 The **minimum processor speed** S_{min} that guarantees the feasibility of a specified limited-preemptive behavior is defined as $S_{min} = min(\mathbb{S})$, where $\mathbb{S} \in$ the set of available processor speeds such that, $\forall\ \tau_i$ in Γ,

$$Q_i^{S_{min}} \geq L_i^{S_{min}}$$

Here, $L_i^{S_{min}}$ is the specified limited-preemption length for τ_i that guarantees the *feasibility of a specified limited-preemptive behavior* per τ_i. We can then calculate the speed-up required to guarantee the feasibility of the limited-preemption requirement L_i^S, which will in turn guarantee specified bounds on the preemption related costs.

Methodology Overview: Our method is composed of 2 steps as shown in Figure 4:

Step 1: Specifying task level limited-preemption requirements to (a) reduce the number of preemptions. (b) enable preemptions at optimal preemption points. (c) enable critical sections execution within non-preemptive regions.

Step 2: Perform sensitivity analysis using the task parameters and the specified limited-preemption requirements to derive the minimum processor speed that guarantees the desired limited-preemptive behavior.

In the following sub-sections we describe each of the steps in detail, followed by some evaluation results.

5.1 Specifying task-level limited-preemption requirements

We can derive task level limited-preemption requirements to (a) reduce the number of preemptions. (b) enable preemptions at optimal preemption points. (c) enable critical sections execution within non-preemptive regions as follows.

i. Reducing the number of preemptions: If the schedulability of a taskset is guaranteed considering the upper-bound on the preemption related overheads, it is indeed schedulable considering the exact overheads. The preemption related overheads can be upper-bounded by the product of the upper-bound on the number of preemptions and the upper-bound on the penalty associated with a single preemption. The upper-bound on the number of times a task τ_i, characterized by a non-preemptive region of maximum length Q_i^S, can be preempted while executing on a speed S processor is given by Baruah (2005) and Yao et al (2010) as $\left\lceil \frac{C_i^S}{Q_i^S} \right\rceil - 1$. Therefore, the limited-preemption length L_i^S, $\forall \tau_i \in \Gamma$, on a speed S processor, that guarantees at most p_i preemptions on τ_i, can be specified as:

$$L_i^S \geq \frac{C_i^S}{p_i+1} \Rightarrow L_i^S = \left\lceil \frac{C_i^S}{p_i+1} \right\rceil \tag{2}$$

It is evident that, on a speed 1 processor, if $Q_i^1 < L_i^1$, where L_i^1 is calculated according to equation 2, τ_i can be guaranteed to incur no more than p_i preemptions. Hence, we have to find a processor speed S which ensures that:

$$Q_i^S \geq L_i^S = \left\lceil \frac{C_i^S}{p_i+1} \right\rceil$$

ii. Enabling preemptions at optimal preemption points: The possibility of enforcing preemptions only at optimal preemption points (Peng et al (2014)) depends on the maximum length of the non-preemptive region on a processor of a given speed S. Remember that $q_{i,j}^S$ denotes the length of execution of τ_i up to its j^{th} optimal preemption point on a speed S processor. Hence, the limited-preemption requirement for a task τ_i can be specified as the largest interval between any two consecutive optimal preemption points of τ_i when it executes on a speed S processor:

$$L_i^S = \max_{1 \leq j < m} (q_{i,j+1}^S - q_{i,j}^S, q_{i,1}^S) \tag{3}$$

Consequently, our goal is to find the processor speed S that satisfies:

$$Q_i^S \geq L_i^S = \max_{1 \leq j < m} (q_{i,j+1}^S - q_{i,j}^S, q_{i,1}^S)$$

iii. Executing critical sections within non-preemptive regions: If the maximum length of the non-preemptive region Q_i^1 of a task τ_i is shorter than its largest critical section CS_i^1, on a speed 1 processor, resource sharing protocols are required. This issue, on the other hand, can be solved by using a faster processor. The processor speed that guarantees the non-preemptive execution of critical sections, under a limited-preemptive scheduling paradigm, is given by the speed S that satisfies the relation:

$$Q_i^S \geq L_i^S = CS_i^S \tag{4}$$

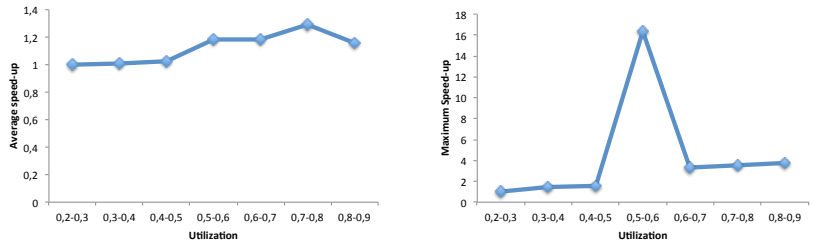
By specifying the limited-preemption length as the maximum of the lengths calculated using equations 2, 3 and 4, we can both guarantee the requirements of reducing the number of preemptions as well as retain the possibility of preemption placement at optimal preemption points, while guaranteeing the execution of critical sections entirely within non-preemptive regions.

5.2 Sensitivity analysis for preemption control

If the length of the largest non-preemptive region is less than the specified limited-preemption length on a speed 1 processor, i.e., $Q_i^1 < L_i^1$, it means that τ_i cannot execute non-preemptively for the specified duration. Therefore, we need to use a faster processor of speed S such that $Q_i^S \geq L_i^S$. In most situations, changing the processor speed may also change the specified limited-preemption lengths to satisfy the desired preemption related cost control requirements, as well as the maximum possible lengths of the limited-preemptive regions of the tasks. The lowest processor speed that guarantees the specified limited-preemption requirements lies in the interval $[S_{low} = 1, S_{high}]$, where S_{high} corresponds to the bounds derived in the previous section. We can perform a sensitivity analysis on the speeds between 1 and S_{high} in order to calculate the minimum processor speed S_{min} which guarantees that every task τ_i can exhibit the specified limited-preemptive behavior, i.e., $Q_i^{S_{min}} \geq L_i^{S_{min}}$.

The length of the maximum non-preemptive regions increase with decrease in the demand bound as shown by Baruah (2005). Therefore, it can be easily shown that the maximum length of the non-preemptive regions increases monotonically with the processor speed (even if only a part of the WCET scales linearly with processor speed). Hence the correctness and optimality of our method is given by the correctness of the binary search.

Evaluation: We validate the theoretical results by evaluating the sensitivity analysis for a set of 1000 tasksets generated using UUniFast algorithm (Bini and Buttazzo (2005)). We calculated the required minimum speed-up that guarantees a fully non-preemptive schedule under the assumption that the entire WCETs scale linearly with speed . We are interested in the speed-up that guarantees a fully non-preemptive schedule because it corresponds to the maximum speed-up required to guarantee any



(a) Average required speed-up.

(b) Maximum required speed-up.

Fig. 5: Required speed-up that guarantees non-preemptive feasibility.

limited-preemptive behavior, i.e., if the task can execute fully non-preemptively, it can execute non-preemptively for a lesser duration.

We plotted the average and maximum speed-ups required to guarantee a fully non-preemptive schedule for different utilization ranges (presented in Figure 5). We observed that, in general, the required average and maximum speed-up factors increases with utilization. Additionally, the average required speed-up was found to be well below 2 as seen from Figure 5a. However, from Figure 5b, we observed that the maximum speed-up required was the highest for utilizations between 0.5 and 0.6, and is greater than 16. On closer examination, we found that the corresponding taskset "suffered" from the *long task problem*, referred to by Short (2010), in which at least one task has an execution time greater than the shortest deadline. It is possible to construct similar tasksets (that have very high required speed-up factors) in all utilization ranges, i.e., it is possible to construct tasksets having any utilization that require arbitrarily large speed-ups to guarantee non-preemptive feasibility. Abdelzaher et al (2002) have identified a large class of real-time tasks, called liquid tasks, where the shortest deadline is much greater than the largest computation time in the taskset. Our evaluations indicate that using faster processors to guarantee specified limited-preemptive behaviors can be particularly feasible for liquid tasks since the speed-up required may not be significantly large.

6 Processor augmentation for limited-preemptive scheduling

This paper quantitatively compares limited-preemptive scheduling with fully preemptive scheduling using the notion of resource augmentation. In the previous sections, we considered the use of processor speed-up to quantify the sub-optimality of limited-preemptive scheduling. Instead, in this section, we examine the consequences of having more number of processors on the limited-preemptive feasibility (such as done by Lam and To (1999) for global preemptive EDF). Specifically, we derive the *processor augmentation bound* that is defined as the upper-bound on the minimum number of processors on which any uniprocessor feasible taskset is guaranteed the specified limited-preemptive behavior. The use of more number of processors is particularly interesting because of the widespread availability of multi-core systems that can be

leveraged upon to limited-preemptively schedule hard real-time tasks, while scheduling the soft and non real-time tasks in the background.

We first show that, in the worst case, the number of processors required to guarantee a specified limited preemptive behavior is equal to the number of tasks in the taskset. We then derive the processor augmentation bounds for the specific case in which the specified limited-preemptive length is no larger than half the shortest deadline, and show that the upper-bound on the number of extra processor required is 3. In the following, let us consider the general case in which no restriction is placed on the specified length of the limited-preemptive region.

Lemma 6 *The minimum number of processors on which a uniprocessor feasible taskset Γ is guaranteed any specified limited preemptive behavior is n, which is the number of tasks in Γ.*

Proof We provide proof by constructing a taskset for which n processors are required to guarantee limited preemptive feasibility— remember that no restrictions exists on the values of the deadlines and specified limited-preemption lengths.

Consider a taskset Γ with each $\tau_i, i = 2,3,..,n$, having a specified limited-preemption length L_i such that $L_i > D_{i-1}$. In this case it is easily seen that if any two tasks are scheduled on the same processor, it is impossible to guarantee the limited-preemptive execution of one of the tasks for a duration equal to the corresponding specified limited-preemption length. This is because if a task τ_j with the shorter relative deadline is released when another task τ_k is executing, and if τ_k blocks τ_j for a duration L_k then τ_j will miss its deadline since $L_k > D_j$. □

The above result shows that it is not possible to guarantee specified limited-preemptive behaviors on fewer processors than the number of tasks. Therefore, in general, the increased processing capacity provided by multicore platforms cannot be leveraged to control preemptive behavior of real-time tasks using limited-preemptive scheduling (Baruah (2005)), unless each core is assigned a single task. However, the above result is derived for the worst case in which no assumptions are made about the taskset. If the largest specified limited-preemption length is no more than half the shortest deadline, a tighter bound can be obtained. In the following subsection, we derive a density based test for limited-preemptive scheduling, and then use this test to derive a tighter processor augmentation bound for limited-preemptive feasibility.

6.1 Density based test for limited-preemptive scheduling

There exists utilization/density based tests for schedulability and feasibility of preemptive real-time tasks under various assumptions (Liu and Layland (1973)). However, to our knowledge, no utilization or density based test exists for limited- and non-preemptive scheduling even under restrictive assumptions. Speeding-up the processor by a constant factor is equivalent to ensuring a bound on the processor utilization (density, in case deadlines can be less than periods).

Yao et al (2010)'s observation that there exists no least upper-bound on the processor utilization below which the non-preemptive feasibility can be guaranteed, relies on the fact that it is possible to construct tasksets with arbitrarily low utilization

that are infeasible under non-preemptive scheduling. We observe that the unschedulability primarily arises because of the fact that at least one task has an execution time greater than the shortest deadline. Consequently, such a condition can be seen as a *necessary unschedulability* test for non-preemptive scheduling, in particular non-preemptive EDF scheduling, of sporadic real-time tasks.

Observation 1 *A sporadic real-time taskset Γ is infeasible under non-preemptive EDF if,*

$$\exists \tau_i \in \Gamma : C_i \geq D_{min}$$

In the following, we first derive a density based test for limited-preemptive EDF feasibility of sporadic real-time tasks. The test when instantiated in the context of non-preemptive EDF provides us with a *sufficient* density based test for non-preemptive EDF feasibility of sporadic real-time tasks.

Lemma 7 *A sporadic real-time taskset Γ is feasible under limited-preemptive EDF, such that every task can execute non-preemptively for at most L_{max} units, if,*

$$\delta_{tot} \leq 1 - \frac{L_{max}}{D_{min}}$$

where $L_{max} = \max_{\forall \tau_i \in \Gamma}(L_i^1)$.

Proof A taskset Γ is limited-preemptive EDF feasible if during any time interval of length t, the sum of total demand bound and the largest limited-preemptive region of the tasks in the taskset is less than or equal to t. It is known that,

$$\sum_{\forall \tau_i in \Gamma} DBF_i(t) \leq \delta_{tot} \times t$$

Therefore, a sufficient condition to guarantee limited-preemptive EDF feasibility is given by, $\forall t \geq D_{min}$

$$\delta_{tot} \times t + L_{max} \leq t \Rightarrow \delta_{tot} \leq 1 - \frac{L_{max}}{t}$$

The value of t that maximizes $\frac{L_{max}}{t}$ is $t = D_{min}$, and hence, the taskset is limited-preemptive EDF feasible if:

$$\delta_{tot} \leq 1 - \frac{L_{max}}{D_{min}}$$

□

Instantiating the above test in the context of a fully non-preemptive EDF scheduler, we get the following test for non-preemptive EDF feasibility.

Corollary 4 *A sporadic real-time taskset Γ is feasible under non-preemptive EDF if,*

$$\delta_{tot} \leq 1 - \frac{C_{max}}{D_{min}}$$

where $C_{max} = \max_{\forall \tau_i \in \Gamma}(C_i^1)$.

The tests presented above generalizes to a utilization based test when the deadlines of the tasks are equal to their time periods. This density based test is interesting since it runs in a time polynomial in the number of tasks, when compared to the exact demand bound based tests by Jeffay et al (1991) and Baruah (2005) that runs in pseudo-polynomial time. The polynomial complexity of the density based test comes at the cost of necessity i.e., the test presented above is only a sufficient condition for schedulability. The density based test is especially applicable to the liquid task model presented by Abdelzaher et al (2002) in which the shortest deadline is orders of magnitude greater than the largest execution time.

6.2 Processor augmentation bound derivation

We now show that it is enough to use 3 processors to guarantee limited preemptivity of a uniprocessor feasible taskset Γ for which $\frac{D_{min}}{L_{max}} \geq 2$.

Lemma 8 *The number of processors on which a uniprocessor feasible taskset Γ is guaranteed limited preemptive feasibility, such that $\frac{D_{min}}{L_{max}} \geq 2$, is upper-bounded by* 3.

Proof Substituting $\frac{D_{min}}{L_{max}} \geq 2$ in Lemma 7, we get that if the total density of the taskset is no greater than 50%, the task set is LP-EDF feasible on a uniprocessor.

Therefore, if we partition Γ into subsets such that the utilization of each subset does not exceed 50%, then we can guarantee the limited preemptive feasibility of each subset on m unit speed processors, where m is equal to the number of such subsets.

In the worst case, in order to partition Γ with total utilization $U_{tot} \leq 1$ into subset of tasks, each with total utilization $\leq \frac{1}{2}$, we need at most 3 processors. □

The use of more number of processors to achieve predictability can be potentially interesting in systems where the slack, after scheduling the hard real-time tasks, are used to schedule soft real-time or non real-time tasks, e.g., using servers. The hard real-time tasks can be partitioned upon the multiple processors/cores to achieve predictability, while the soft- and non- real-time tasks can execute upon servers while maximizing the service to them using known schemes (Leontyev and Anderson (2008)).

7 Discussion

In this section, we discuss the resource augmentation bounds derived in the paper in different (but related) contexts, as well as clarify some details:

– **Enable limited-preemptive feasibility:** Feasibility guarantees cannot be provided under limited preemptive scheduling if the length of the largest non-preemptive region is greater than the shortest deadline in the taskset. However, as pointed out by Short (2010), solutions are available to overcome this problem, e.g., by using code-refactoring or by changing design parameters. Code re-factoring can be performed to reduce the execution time of real-time tasks, e.g., by the use of

more efficient code, that effectively amounts to using a faster processor. Similarly, scaling all deadlines/time periods by the same factor is similar to speeding up the processor [1]. Additionally, parallel algorithms can be used to *speed-up* task executions, and hence, a specified limited-preemptive behavior can be guaranteed by using one or more of the above techniques.

Our results: The speed-up bound presented in this paper quantifies the extent to which code-refactoring must be performed to reduce task execution times, or the extent to which task parameters must be *adjusted*, in order to address the unschedulability arising out of tasks with very large execution times. Similarly, the speed-up bound also quantifies the requirement on the amount of parallelizable code (Amdahl (1967)) when parallel algorithms are used to guarantee limited-preemptivity.

– **Accuracy of timing analysis tools:** Most timing analysis tools overestimate the Worst Case Execution Time (WCET) in order to provide safe bounds, consequently enabling hard real-time guarantees. However, overestimating WCETs cause significant loss of system utilization. One of the reasons behind this overestimation is the fact that it is very difficult to accurately account for preemption related overheads, especially on fully preemptive systems. On the other hand, the worst case preemption behavior that maximizes the associated overheads occurs very rarely in practice— nevertheless the system should be built to handle the worst case to provide hard guarantees. Preemption related overheads depend on the number of preemptions, as well as the points at which these preemptions occur. Hence, limiting preemptions to specified points in code improve WCET predictions since preemption overheads can be more accurately accounted during timing analysis.

  Our results: The accuracy of timing analysis tools can be quantified in terms of the feasibility of limiting preemptions to specified preemption points in the code. We plan to further investigate this in a future work by considering the preemption costs in the analysis.

8 Conclusions

This paper essentially bridges the preemptive and non-preemptive real-time scheduling paradigms by providing significant theoretical results building on the limited-preemptive scheduling paradigm. We investigated the sub-optimality of limited-preemptive scheduling with respect to a uniprocessor optimal scheduling algorithm, like the preemptive EDF, using the widely accepted notion of resource augmentation. For this purpose, we investigated how extra resources affect the preemptive behavior of real-time tasks, and derived bounds on the 1) required processor speed-up and 2) required

[1] Note that design parameters such as deadlines and time periods in many systems are negotiable not only in many soft real-time applications, but also in many hard real-time applications (please refer to Buttazzo and Abeni (2002) for more details).

number of processors, that guarantees a specified limited-preemptive behavior. The derived bounds when instantiated in the context of fully non-preemptive EDF allows us to quantify the sub-optimality of non-preemptive scheduling. Finally, we use the derived speed-up bounds to calculate the minimum processor speed-up required that guarantees a specified limited-preemptive behavior, which in turn minimizes preemption related overheads in the schedule.

Future work includes extending limited preemptive scheduling to the case of multiprocessors and deriving corresponding speed-up factors (a recent work that extended the results presented in this paper to the case of global EDF is presented in Thekkilakattil et al (2014)), as well as applying resource augmentation to quantify the accuracy of timing analysis tools.

Acknowledgements We would like to thank Sanjoy Baruah for suggesting the proof of Lemma 6, Jim Anderson for pointing out the relationship between the resource augmentation bound and accuracy of timing analysis tools, as well as the reviewers for their valuable feedback.

References

Abdelzaher T, Andersson B, Jonsson J (2002) The aperiodic multiprocessor utilization bound for liquid tasks. In: The Real-Time and Embedded Technology and Applications Symposium

Altmeyer S, Davis RI, Maiza C (2012) Improved cache related pre-emption delay aware response time analysis for fixed priority pre-emptive systems. Real-Time Systems

Amdahl GM (1967) Validity of the single processor approach to achieving large scale computing capabilities. In: American Federation of Information Processing Societies Spring Joint Computer Conference

Audsley N, Burns A, Richardson MF, Wellings AJ (1991) Hard real-time scheduling: The deadline-monotonic approach. In: The IEEE Workshop on Real-Time Operating Systems and Software

Baruah S (2005) The limited-preemption uniprocessor scheduling of sporadic task systems. In: The Euromicro Conference on Real-Time Systems

Baruah S, Burns A (2006) Sustainable scheduling analysis. In: The Real-Time Systems Symposium

Baruah S, Burns A (2008) Quantifying the sub-optimality of uniprocessor fixed-priority scheduling. In: The International Conference on Real-Time and Network Systems

Baruah S, Mok A, Rosier L (1990a) Preemptively scheduling hard-real-time sporadic tasks on one processor. In: The Real-Time Systems Symposium

Baruah SK, Rosier LE, Howell RR (1990b) Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Real-Time Systems

Bertogna M, Baruah S (2010) Limited preemption EDF scheduling of sporadic task systems. The IEEE Transactions on Industrial Informatics

Bertogna M, Buttazzo G, Marinoni M, Yao G, Esposito F, Caccamo M (2010) Preemption points placement for sporadic task sets. In: The Euromicro Conference on Real-Time Systems

Bini E, Buttazzo GC (2005) Measuring the performance of schedulability tests. Real-Time Systems

Bui BD, Caccamo M, Sha L, Martinez J (2008) Impact of cache partitioning on multitasking real time embedded systems. In: The International Conference on Embedded and Real-Time Computing Systems and Applications

Busquets-Mataix JV, Serrano JJ, Ors R, Gil P, Wellings A (1996) Adding instruction cache effect to schedulability analysis of preemptive real-time systems. In: The IEEE Real-Time Technology and Applications Symposium

Buttazzo G, Abeni L (2002) Adaptive workload management through elastic scheduling. Real-Time Systems

Buttazzo G, Bertogna M, Yao G (2012) Limited preemptive scheduling for real-time systems: A survey. The IEEE Transactions on Industrial Informatics

Buttle D (2012) Real-time in the prime-time. Keynote speech given at the Euromicro Conference on Real-Time Systems URL `http://ecrts.eit.uni-kl.de/fileadmin/user_media/ecrts12/ECRTS12-Keynote-Buttle.pdf`

Davis R, Rothvoss T, Baruah S, Burns A (2009a) Exact quantification of the sub-optimality of uniprocessor fixed priority pre-emptive scheduling. Real-Time Systems

Davis R, George L, Courbin P (2010) Quantifying the Sub-optimality of Uniprocessor Fixed Priority Non-Pre-emptive Scheduling. In: The International Conference on Real-Time and Network Systems

Davis RI, Rothvoss T, Baruah SK, Burns A (2009b) Quantifying the sub-optimality of uniprocessor fixed priority pre-emptive scheduling for sporadic tasksets with arbitrary deadlines. In: The International Conference on Real-Time and Network Systems

Dertouzos ML (1974) Control robotics: The procedural control of physical processes. In: IFIP Congress

George L, Muhlethaler P, Rivierre N (1995) Optimality and non-preemptive real-time scheduling revisited. Research report, INRIA

George L, Rivierre N, Spuri M (1996) Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling. Research report, INRIA

Jeffay K, Stanat DF, Martel CU (1991) On non-preemptive scheduling of periodic and sporadic tasks. In: The Real-time Systems Symposium

Ju L, Chakraborty S, Roychoudhury A (2007) Accounting for cache-related preemption delay in dynamic priority schedulability analysis. In: IEEE Design Automation and Test in Europe

Kalyanasundaram B, Pruhs K (2000) Speed is as powerful as clairvoyance. Journal of ACM

Lam TW, To KK (1999) Trade-offs between speed and processor in hard-deadline scheduling. In: The ACM-SIAM symposium on Discrete algorithms

Lee CG, Hahn J, Seo YM, Min SL, Ha R, Hong S, Park CY, Lee M, Kim CS (1998) Analysis of cache-related preemption delay in fixed-priority preemptive scheduling. IEEE Transactions on Computers

Leontyev H, Anderson J (2008) A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees. In: The Euromicro Conference on Real-Time Systems

Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. The Journal of ACM

Marinoni M, Buttazzo G (2007) Elastic dvs management in processors with discrete voltage/frequency modes. IEEE Transactions on Industrial Informatics

McKee SA (2004) Reflections on the memory wall. In: Proceedings of the conference on Computing frontiers

Peng B, Fisher N, Bertogna M (2014) Explicit preemption placement for real-timeconditional code. In: The Euromicro Conference on Real-Time Systems

Pillai P, Shin KG (2001) Real-time dynamic voltage scaling for low-power embedded operating systems. In: The ACM symposium on Operating systems principles

Saha S, Ravindran B (2012) An experimental evaluation of real-time dvfs scheduling algorithms. In: Proceedings of the Annual International Systems and Storage Conference

Short M (2010) The case for non-preemptive, deadline-driven scheduling in real-time embedded systems. In: Lecture Notes in Engineering and Computer Science: Proceedings of the World Congress on Engineering

Staschulat J, Schliecker S, Ernst R (2005) Scheduling analysis of real-time systems with precise modeling of cache related preemption delay. In: The Euromicro Conference on Real-Time Systems

Tan Y, Mooney V (2007) Timing analysis for preemptive multitasking real-time systems with caches. ACM Transactions on Embedded Computing Systems

Thekkilakattil A, Dobrin R, Punnekkat S (2013) Quantifying the sub-optimality of non-preemptive real-time scheduling. In: The Euromicro Conference on Real-Time Systems

Thekkilakattil A, Baruah S, Dobrin R, Punnekkat S (2014) The global limited preemptive earliest deadline first feasibility of sporadic real-time tasks. In: The Euromicro Conference on Real-Time Systems

Thiele L (2014) Model-based design of real-time systems. Keynote speech given at the Euromicro Conference on Real-Time Systems URL `http://ecrts.eit.uni-kl.de/fileadmin/files_ecrts14/documents/ECRTS14_Keynote_LotharThiele.pdf`

Ward B, Thekkilakattil A, Anderson J (2014) Optimizing preemption-overhead accounting in multiprocessor real-time systems. In: The International Conference on Real-Time Networks and Systems

Yao G, Buttazzo G, Bertogna M (2010) Comparitive evaluation of limited preemptive methods. In: The International Conference on Emerging Technologies and Factory Automation