CISTER
Research Center in
Real-Time & Embedded
Computing Systems

Technical Report

Optimal Procrastination Interval for Constrained Deadline Sporadic Tasks upon Uniprocessors

Muhammad Ali Awan

Patrick Meumeu Yomsi

Stefan M. Petters

CISTER-TR-131106

Version:

Date: 11/7/2013

Optimal Procrastination Interval for Constrained Deadline Sporadic Tasks upon Uniprocessors

Muhammad Ali Awan, Patrick Meumeu Yomsi, Stefan M. Petters

CISTER Research Unit

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail: muaan@isep.ipp.pt, pamyo@isep.ipp.pt, smp@isep.ipp.pt

http://www.cister.isep.ipp.pt

Abstract

Energy consumption is a major concern in modern real-time embedded systems and leakage current is a main contributor to it. To deal with the leakage current, several procrastination approaches have been proposed in the past in order to reduce the energy consumption. These approaches approximate the procrastination interval for the ease of analysis and sub-optimally utilise the potential to reduce the energy consumption. This paper presents an optimal method to determine the procrastination interval of each task and generalise the task-model to cover the constrained deadline tasks. Analytical and experimental results show the superiority of the proposed technique. In the best case, the proposed technique extends the average sleep interval up to 75% and decreases the energy consumption in idle state up to 55% over the state-of-the-art.

This work was partially supported by National Funds through FCT (Por-tuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within project Ref. FCOMP-01-0124-FEDER-037281 and [REPOMUC] project, ref. FCOMP-01-0124-FEDER-015050, and by ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/70701/2010.

Optimal Procrastination Interval for Constrained Deadline Sporadic Tasks upon Uniprocessors*

Muhammad Ali Awan, Patrick Meumeu Yomsi, Stefan M. Petters
CISTER/INESC-TEC, ISEP
Polytechnic Institute of Porto, Portugal
{muaan, pamyo, smp}@isep.ipp.pt

ABSTRACT

Energy consumption is a major concern in modern real-time embedded systems and leakage current is a main contributor to it. To deal with the leakage current, several procrastination approaches have been proposed in the past in order to reduce the energy consumption. These approaches approximate the procrastination interval for the ease of analysis and sub-optimally utilise the potential to reduce the energy consumption. This paper presents an optimal method to determine the procrastination interval of each task and generalise the task-model to cover the constrained deadline tasks. Analytical and experimental results show the superiority of the proposed technique. In the best case, the proposed technique extends the average sleep interval up to 75% and decreases the energy consumption in idle state up to 55% over the state-of-the-art.

1. INTRODUCTION

Researchers have been studying uniprocessor embedded systems which consist of a finite number of recurring processes (referred to as "tasks" hereafter) for over forty years now. For such systems, each task is commonly characterised by parameters such as its worst-case execution requirement, its activation rate, and its temporal deadline reflecting its timing constraint. Over this period of time, they have come up with a number of very important results, developed some useful algorithmic techniques and built up an entire body of intuitions. Taken together, these results, techniques and intuitions have allowed system designers to come up with a very good understanding of the manner in which uniprocessor embedded systems behave. However, the emerging application requirements in the embedded systems arena have increased dramatically over the past years in terms of computing demands, need of reduced size and weight. Furthermore, besides having specific functional requirements, many embedded systems have stringent timing requirements (the system is then referred to as "*real-time*" (RT) embedded system). The RT embedded system domains include (but are not limited to) air-traffic control, aerospace, automotive, railway control systems, medical, factory automation, mobile phones and military equipment. Among these RT systems, *hard* RT systems are those for which violating any timing requirement can entail severe consequences, e.g., it can damage the system, lead to substantial economic loss, or even harm people or threaten human lives. Throughout this paper, hard RT embedded systems that have *limited power supply* are considered. This additional energy constraint is induced by battery power mobile device, limited or intermittent power supply for example. Even when the application is technically feasible upon the targeted platform in the sense that the platform can provide a sufficient computing capacity for the execution of the application, it has become unreasonable to expect to implement such a system without addressing the issue of minimising its power and energy consumption. To this end, chip manufacturers are putting considerable efforts in this direction and this aim aligns neatly with the desired "wish-list" of most embedded systems.

There are two main sources of energy consumption in embedded systems: the *dynamic power dissipation* which is related to the current that flows when the switching of transistors takes place at run-time and the *leakage power dissipation* which is proportional to the current that flows regardless of gate switching. Since CMOS technology miniaturisation has increased the sub-threshold leakage current exponentially to an extent where leakage power dissipation may dominate the dynamic power dissipation, this factor can no longer be considered as negligible. This fact has been identified as a major concern in the International Technology RoadMap For Semiconductors 2010 Update under special topics [17]. To reduce the impact of leakage current, hardware vendors have provided multiple sleep states with reduced transition overheads (energy/time) when compared to previous processors, which can be exploited by the system designer to shut-down certain parts of the processor.

Procrastination scheduling is commonly used at system level to reduce the leakage power dissipation. In this technique the execution of the processor already in sleep state is delayed as much as possible while ensuring the timing constraints of all tasks are met. As such, the number of sleep transitions are decreased and consequently the energy overhead is minimised. Many power saving algorithms based on procrastination scheduling [18,20,21] approximate the procrastination interval of tasks leading to sub-optimal energy savings. This research fills this gap.

The contribution of this paper is twofold. First, it presents an optimal method to compute the procrastination interval of the tasks for the implicit deadline task model and then it extends the results to a more general case, i.e., the *sporadic constrained deadline task*

*This work was partially supported by National Funds through FCT (Portuguese Foundation for Science and Technology) and by ERDF (European Regional Development Fund) through COMPETE (Operational Programme 'Thematic Factors of Competitiveness'), within [REPOMUC] project, ref. FCOMP-01-0124-FEDER-015050; by National Funds through FCT and by the EU ARTEMIS JU funding within project ref. ARTEMIS/0003/2012, grant nr. 333053 (CONCERTO); and by ESF (European Social Fund) through POPH (Portuguese Human Potential Operational Program), under PhD grant SFRH/BD/70701/2010.

RTNS 2013, October 16 - 18 2013, Sophia Antipolis, France

http://dx.doi.org/10.1145/2516821.2516837.

model where the temporal deadline of each task is allowed to be less than or equal to its activation rate. In sporadic task model, two consecutive instances of a task are separated by at least a minimum inter-arrival time.

The rest of the paper is organized as follows. Section 2 and Section 3 present the state-of-the-art and the system model used in this paper, respectively. Section 4 explains the limitations of the state-of-the-art while determining the procrastination interval and provides a new method to improve it over the existing solutions. The optimality of the proposed method along with its extension to the constrained deadline task model is also discussed in this section. The complexity of the proposed approach is presented in Section 5, which is followed by extensive simulation results presented in Section 6. The discussion is concluded in Section 7.

2. RELATED WORK

Leakage-aware scheduling was first addressed by Lee et al. [21] for periodic hard real-time systems. They proposed two different solutions: the leakage control earliest deadline first algorithm (LC-EDF) and the leakage control dual-priority algorithm (LC-DP) for dynamic and static priority schemes, respectively. LC-EDF initiates the sleep state when the system becomes idle and delays the next busy interval to extend the sleep interval. This algorithm combines short idle intervals in the schedule to generate long sleep intervals and saves transition overheads. LC-DP works on the same mechanism for the static priority schedulers. The proposed algorithm needs external specialised hardware to manage such mechanism online. Baptiste [5] developed a polynomial time algorithm to minimise the static power consumption and transition overhead of the non DVFS system with unit-sized RT aperiodic tasks.

Some efforts were made to combine the leakage-aware scheduling with DVFS to minimise the overall energy consumption. Irani et al. [16] considered shutdown in combination with DVFS and proposed a 3-competitive[1] offline and a constant-competitive ratio online algorithm. They assume a continuous spectrum of available frequencies, an execution model with an inverse relation of frequency with execution time and an external hardware. Niu and Quan [25] addressed the dynamic and leakage power consumption simultaneously on a DVFS enabled processor for hard real-time systems. Their proposed algorithm is based on the latest arrival time of jobs estimated by expanding the schedule for the hyper-period. It cannot be used online due to extensive analysis overhead. The algorithm works with a given set of jobs with constrained deadlines. However, in real-time system it is common to have sequence of infinite job releases and their restrictive approach does not support such a model. Jejurikar et al. [20] improved LC-EDF and integrated it with DVFS to minimise the total power consumption. They also determined the critical speed η_{crit} that provides the lower bound on the processor frequency to minimise the energy consumption per cycle. They proved that the procrastination interval determined by their algorithm is always greater than or equal to the one estimated by the LC-EDF algorithm. Nevertheless, they did not relax on the requirement of the additional hardware.

Jejurikar et al. [18] showed that LC-DP originally proposed by Lee et al. [21] may cause some of the tasks to miss their deadlines. They improved the original algorithm and combined it with their DVFS algorithm to reduce both dynamic and static power consumption. However, their system model is based on the same assumptions [20, 21]. Later on Chen and Kuo [8] determined some timing anomalies in the work of Jejurikar et al. [18] and showed that their approach still might lead to some tasks missing their deadlines. They proposed another two-phase algorithm that estimates the frequency and the procrastination interval offline, and predicts shutdown instances online. The task not executing for its worst case execution time generates spare capacity in the schedule slack. The slack reclamation algorithm (SRA) of Jejurikar and Gupta [19] reclaims such slack which is used to further procrastinate or slow down the execution of the tasks to minimise the energy. Their proposed slack distribution policy either assigns entirely the dynamically reclaimed slack to slowdown or distributes it between slowdown and procrastination. SRA also needs an external hardware to implement the algorithm.

Chetto and Chetto [10] worked on identification of temporal locality and duration of idle intervals for the periodic task model. Later, these results were extended by Silly to the scheduling of periodic and soft aperiodic tasks with resource constraints [28], and provided a framework for the scheduling of periodic and non preemptable tasks [11]. These works have tackled the similar problem as the one addressed in this paper with the different objective to co-schedule the aperiodic tasks in the presence of periodic task-set. Huang et al. [14, 15] estimated the procrastination interval for a device to activate the shutdown by predicting future events using RT calculus [29] and ensured schedulability with RT interfaces [30].

The scope of this paper is the procrastination scheduling [19–21]. In this framework, the procrastination interval is revisited on the arrival of each task during the sleep interval. The estimated procrastination interval depends on the task-properties. This research aims to provide a new mechanism to reduce the pessimism involved in the state-of-the-art while estimating the procrastination interval and to extend the results to the sporadic constrained deadlines task-model. Last but not least, the sensitivity analysis [3,4,31] provides tools to compute the tolerance over the execution time and the deadline reductions of the tasks. These techniques can also be tweaked and used to compute the procrastination interval of a task.

3. SYSTEM MODEL

The sporadic constrained deadline task model is assumed in this research, where a task-set $\tau \stackrel{\text{def}}{=} \langle \tau_1, \tau_2, \ldots, \tau_\ell \rangle$ is composed of ℓ *independent* tasks. Each task τ_i generates a potentially infinite sequence of jobs and is characterised by a 3-tuple $\tau_i = \langle C_i, D_i, T_i \rangle$, where C_i is the worst-case execution time, D_i is the relative deadline and $T_i \geq D_i$ is the *minimum* inter-arrival time between two consecutive jobs of τ_i. These parameters are real-valued and given with the following interpretation. The k^{th} job $j_{i,k}$ of τ_i is defined as $j_{i,k} \stackrel{\text{def}}{=} \langle r_{i,k}, c_{i,k}, d_{i,k} \rangle$, where $r_{i,k}$ is the absolute release time ($r_{i,k} - r_{i,k-1} \geq T_i$), $c_{i,k} \leq C_i$ is the actual execution time and $d_{i,k} \stackrel{\text{def}}{=} r_{i,k} + D_i$ is the absolute deadline. The hyper-period L^* of τ is defined as the least common multiple of the tasks periods, i.e. $L^* \stackrel{\text{def}}{=} LCM\{T_1, T_2, \cdots, T_\ell\}$. The notion of LCM is extended to real numbers as follows:

$$LCM(a, b) \stackrel{\text{def}}{=} \inf\{x \in \mathbb{R}_+ : \exists p, q \in \mathbb{N}_+, x = pa = qb\}$$

(see [7] for further details). The utilisation of task τ_i is $U_i \stackrel{\text{def}}{=} \dfrac{C_i}{T_i}$ and the system utilisation is $U \stackrel{\text{def}}{=} \sum_{\forall \tau_i \in \tau} \frac{C_i}{T_i}$. Tasks are scheduled using the earliest deadline first algorithm (EDF) [22] and preemption is allowed at no cost or penalty.

This work assumes a single processor which has active and idle states with power consumption of P_A and P_I, respectively. A set of N sleep states with different characteristics is assumed. Each sleep

[1]An algorithm is termed as competitive if its competitive ratio, i.e. ratio between the performance of the algorithm and the optimal offline algorithm, is bounded by a constant number.

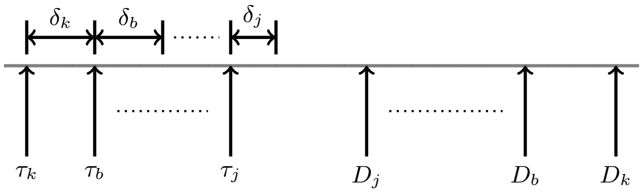

Figure 1: "Accumulated delays under EDF scheduling [21]"

state $S_n \stackrel{\text{def}}{=} \langle P_n, tr_n, E_n \rangle$, where P_n is the power consumption in the sleep state and E_n is the energy overhead associated to a complete sleep transition. A sleep state has the transition overhead time of going into a sleep state and the wake-up transition overhead from a sleep state to an active state. For the sake of simplicity, it is assumed these two transition overheads are equal and denoted as tr_n. Each sleep state has a break-even-time bet_n computed through known approaches [1,9,12]. The definition of bet_n implies that the system will save energy if a sleep state S_n is initiated for more than bet_n. A processor completes its transition once initiated.

4. PROCRASTINATION INTERVAL

Initially, this work assumes implicit deadline task model i.e., $D_i = T_i,\ \forall \tau_i \in \tau$, to compare against the state-of-the-art which assumes this model. Later in Section 4.5, this restriction is relaxed to a more general case, i.e., the constrained deadline task model, where tasks may have deadlines less than their periods ($D_i \leq T_i$). Formally, the procrastination interval is defined as follows.

Definition 1 (Procrastination Interval). *The procrastination interval is the maximum time interval allowed to delay the execution of the ready tasks without violating any timing constraints of the system.*

The longest duration of such an interval is desired to reduce the energy consumption. Before presenting our procrastination technique, existing ones are discussed.

4.1 Limitations of the Existing Procrastination Approaches

In the leakage-aware procrastination scheduling, Lee et al. [21] initially proposed the online mechanism LC-EDF. To understand the basic principle behind this algorithm, let us consider the example given in Figure 1, taken from the work of Lee et al. [21]. Assume that task τ_k is the first which arrives in a sleep mode and has a deadline D_k. The procrastination interval Δ_k of τ_k is computed with the condition $\sum_{\forall \tau_i \in \tau : i \neq k} \frac{C_i}{T_i} + \frac{C_k + \Delta_k}{T_k} = 1$. Suppose t is the current time then the timer is initialised with $t + \Delta_k$ to wake-up the system. After the timer initialisation, a procrastination interval is only recomputed when a newly arrived task has the highest priority when compared to other tasks in the ready queue. For instance, after $\delta_k \leq \Delta_k$ time units, τ_b arrives with a deadline $D_b < D_k$; a new procrastination interval $t + \Delta_b$ is determined as $\sum_{\forall \tau_i \in \tau : i \notin \{k,b\}} \frac{C_i}{T_i} + \frac{C_k + \delta_k}{T_k} + \frac{C_b + \Delta_b}{T_b} = 1$. The wake-up timer is reset to $t + \Delta_b$. Similarly, for any other task τ_j with the highest priority when compared to the tasks in the ready queue, the procrastination interval Δ_j in the sleep state of a processor is determined by using Equation 1, where $lp(j)$ is the set of indices of the tasks arrived before τ_j and with deadlines longer than τ_j. In this equation, δ_i is the interval between an arrival of any job of task τ_i (having highest priority at that instant) and any next task arrival

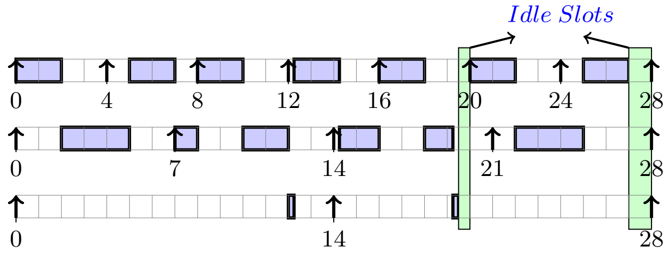

Figure 2: Schedule with $\tau_1 = \langle 2,4,4 \rangle, \tau_2 = \langle 3,7,7 \rangle$ and $\tau_3 = \langle 0.25, 14, 14 \rangle$

having priority higher than τ_i in the system's sleep state. The limitations of LC-EDF are the increased online complexity to maintain a track of δ_i and considering the utilisation of the low priority tasks.

$$\sum_{\forall \tau_i \in \tau : i \notin lp(j), i \neq j} \frac{C_i}{T_i} + \sum_{i \in lp(j)} \frac{C_i + \delta_i}{T_i} + \frac{C_j + \Delta_j}{T_j} = 1 \qquad (1)$$

Jejurikar et al. [20] proposed an offline method to compute the procrastination interval for each task and thus reducing the online complexity. In the online phase, the first task that arrives in sleep mode initialises the wake-up timer ζ with its procrastination interval. The timer ζ counts down with every clock cycle. If another task (say τ_n) arrives before the timer expires, the timer value is adjusted as follows: $\zeta \leftarrow \min(\zeta, t + Z_n)$, where t is the current time and Z_n is the procrastination interval. They proposed Theorem 1 to estimate the procrastination intervals of the tasks offline, where η_k is the frequency of the processor. The value of η_k is set to 1, i.e., maximum frequency, for the ease of presentation. They proved that their derived technique is superior to LC-EDF method to compute the procrastination intervals.

Theorem 1. *[20] Given tasks in τ are ordered in non-decreasing order of their periods, the procrastination algorithm guarantees all task deadlines if the procrastination interval Z_i of each task τ_i satisfies the following two conditions:*

$$\forall \tau_i \in \tau, \qquad \frac{Z_i}{T_i} + \sum_{\forall \tau_k \in \tau : k \leq i} \frac{1}{\eta_k} \frac{C_k}{T_k} \leq 1 \qquad (2)$$

$$\textit{and} \qquad \forall k < i, \qquad Z_k \leq Z_i \qquad (3)$$

While computing the procrastination interval for task τ_i, Jejurikar et al. [20] only considers the utilisation of the tasks having priority greater than or equal to τ_i (assuming a synchronous release of all tasks also known as *critical instant* in literature). Moreover, if any of the low priority task produce a low procrastination interval when compared to the high priority tasks, the procrastination interval of all the high priority tasks are readjusted by considering Equation 3. This latter equation is driven by the online approach of Jejurikar et al. (see [20] for details). Though the proposed method has its merits as it reduces the set of tasks considered for the procrastination of each task, its limitation is that it *approximates* the procrastination intervals by considering their utilisations. Let us demonstrate this shortcoming with the following example.

Example 1: Assume a task-set consisting of three tasks $\tau_1 = \langle 2,4,4 \rangle$, $\tau_2 = \langle 3,7,7 \rangle$ and $\tau_3 = \langle 0.25, 14, 14 \rangle$. Rearranging Equation 2, Z_i can be computed with Equation 4 as given below.

$$Z_1 = (1 - \frac{2}{4})4 = 2$$

$$Z_2 = (1 - \frac{2}{4} - \frac{3}{7})7 = 0.5$$

$$Z_3 = (1 - \frac{2}{4} - \frac{3}{7} - \frac{0.25}{14})14 = 0.75$$

Final values after applying Equation 3 are $Z_1 = 0.5, Z_2 = 0.5$ and $Z_3 = 0.75$.

$$Z_i = \left(1 - \sum_{\forall \tau_k \in \tau : k \le i} \frac{C_k}{T_k}\right) T_i \tag{4}$$

Figure 2 shows the schedule for the aforementioned example. With a careful observation it can be seen that the procrastination interval of τ_1, τ_2 and τ_3 can be extended to 1, 1 and 1.5 time units respectively without causing any deadline miss in the system, which represents 50% gain over the method used by Jejurikar et al. [20]. This example illustrates that substantial energy gains can be achieved by improving the method to compute the procrastination intervals of the tasks.

4.2 Proposed Approach: Demand Bound Function Based Procrastination (PDBF)

The demand bound function (DBF) [6, 26] is used in this paper to compute the procrastination interval of the tasks in the context of uniprocessor scheduling. The DBF is an abstraction of the computation requirements of tasks which has been observed to correlate very closely with schedulability property of the task-set.

DEFINITION 2. *(DBF [6]): The demand for any constrained deadline task τ_i and positive time t, denoted by $\mathrm{DBF}(\tau_i, t)$, is the maximum cumulative execution requirement of jobs of task τ_i in any interval of length t. Formally, $\mathrm{DBF}(\tau_i, t)$ is presented in Equation 5.*

$$\forall t \ge 0, \qquad \mathrm{DBF}(\tau_i, t) \stackrel{\text{def}}{=} \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right) \cdot C_i \tag{5}$$

From Equation 5, it is easy to see that $\mathrm{DBF}(\tau_i, t)$ is a step-case function in t with first step occurring at time $t = D_i$ and subsequent steps separated by *exactly* T_i time units. The DBF for the whole task-set is $\mathrm{DBF}(\tau, t) \stackrel{\text{def}}{=} \sum_{\tau_i \in \tau} \mathrm{DBF}(\tau_i, t)$. The DBF based procrastination (PDBF) scheme achieves extended sleep intervals for any task-set. For instance, consider the DBF of the aforementioned example in Figure 3. Three stair case functions show the DBF of $\tau_1, \tau_1 + \tau_2$ and $\tau_1 + \tau_2 + \tau_3$. The straight line with a slope of 1 represents the supply bound function (SBF) of the processor.

PDBF uses the same logic as the one given in Theorem 1, i.e., synchronous release of all tasks sorted in a non-decreasing order of their deadlines and computes the procrastination interval of a task with DBF instead of considering tasks utilisations. Indeed when $D_i \le T_i$ the utilisation is no longer a good metric for the computation requirement of the tasks whereas the PDBF approach is easily extensible. To compute the maximum procrastination interval of a task τ_i, the PDBF approach *subtracts the demand of the task τ_i along with the demand of the all higher priority jobs from the* SBF. It has to be noted that this difference is computed between the first deadline of task τ_i and the end of the hyper-period (the reason is explained in Theorem 2). Due to the stair-case property of the DBF, it is sufficient to compute the difference at the deadlines. Let χ_i represent the minimum difference of SBF and the demand, then for the given example, $\chi_1 = 2$, $\chi_2 = 1$ and $\chi_3 = 1.5$. However, Theorem 1 does not allow to have procrastination interval of τ_1 greater than that of τ_2, therefore, the value of χ_1 is scaled down to 1 as well, which implies $\chi_1 = 1$, $\chi_2 = 1$ and $\chi_3 = 1.5$. When $D_i = T_i$, the $\mathrm{DBF}(\tau_i, t)$ of task τ_i presented in Equation 5 can be rewritten as shown in Equation 6.

$$\mathrm{DBF}_I(\tau_i, t) = \left\lfloor \frac{t}{T_i} \right\rfloor C_i \qquad \text{as} \qquad t \ge 0 \tag{6}$$

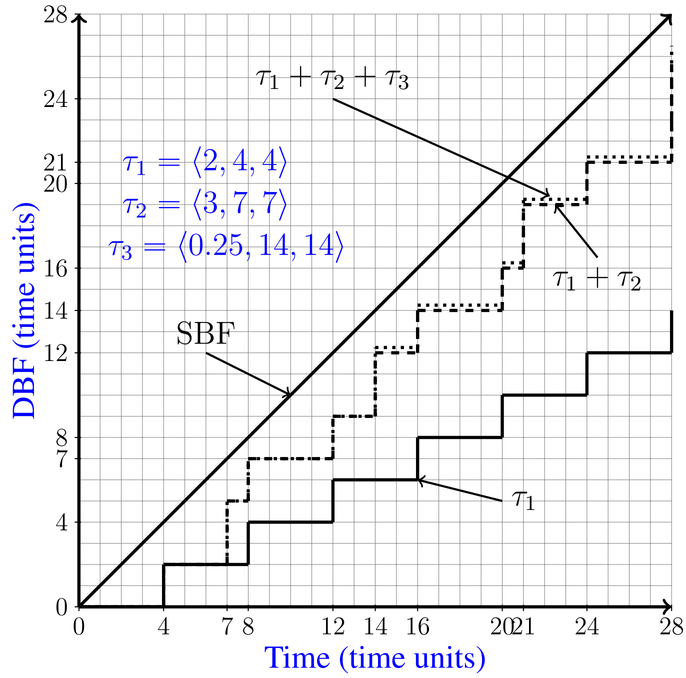

Figure 3: Demand Bound Function of the example

THEOREM 2. *Given tasks in τ are ordered in a non-decreasing order of their relative deadlines, the* PDBF *scheme preserves all task deadlines, if the maximum procrastination interval of task τ_i, denoted by χ_i, is computed with Equation 7 while respecting the condition given in Equation 8.*

$$\begin{aligned}
\chi_i &\stackrel{\text{def}}{=} \min_{\forall \tau_j \in \tau : j \le i, \forall t \ge 0} \left\{ t - \sum_{\forall \tau_k \in \tau : k \le i} \mathrm{DBF}_I(\tau_k, t) \right\} \qquad (7)\\
&= \min_{\forall \tau_j \in \tau : j \le i, \forall t \in M(i,j)} \left\{ t - \sum_{\forall \tau_k \in \tau : k \le i} \left\lfloor \frac{t}{T_k} \right\rfloor C_k \right\}\\
&\textit{where} \quad M(i,j) = \left\{ n_j T_j : \left\lceil \frac{T_i}{T_j} \right\rceil \le n_j \le \left\lfloor \frac{L^*}{T_j} \right\rfloor \right\}
\end{aligned}$$

$$\forall k < i, \; \chi_k \le \chi_i \tag{8}$$

PROOF SKETCH. Suppose a task τ_i arrives in the sleep state. The timer is set to the procrastination interval computed with Equation 7 respecting the condition given in Equation 8. The time interval to wake up the system can only be decreased with an arrival of new task. This procrastination interval can be seen as an additional task τ_{proc} with a priority equal to the highest priority task, execution time equal to the wake-up sleep interval and it executes before the next busy period. Equation 8 ensures that all the tasks with deadlines greater than or equal to τ_i will have procrastination interval greater than or equal to χ_i. Therefore, τ_{proc} will not increase the system demand beyond the SBF in the presence of low priority tasks. Furthermore, the higher priority tasks can only shorten the execution time of τ_{proc} (i.e., procrastination interval) on their arrival to respect their deadlines and the deadlines of the other tasks. The sleep interval is bounded by the procrastination interval of the first task and it only decreases with the new arrivals, therefore, based on the previous logic it will not affect the schedulability of any high priority task. Moreover, it is sufficient to consider the deadlines in the interval $[D_i, L^*]$ as the procrastination interval of a task is only considered when it has the highest priority on its arrival in the ready queue. $\square$

4.3 Procrastination Interval Improvement

The best known maximum procrastination interval is the one derived in Jejurikar et al. [20] method for each task in the state-of-the-art. This is obtained by considering the worst-case scenario i.e., critical instant. This section shows that the procrastination interval computed for any task through PDBF will always be greater than or equal to Z_i (see Lemma 1).

LEMMA 1. *Given tasks in τ are ordered in a non-decreasing order of their relative deadlines, the procrastination interval χ_i for any task τ_i computed with PDBF scheme is always greater than or equal to the procrastination interval Z_i computed through Jejurikar et al. [20] method, i.e.,*

$$\min_{\forall \tau_j \in \tau : j \leq i, \forall t \in M(i,j)} \left\{ t - \sum_{\forall \tau_k \in \tau : k \leq i} \left\lfloor \frac{t}{T_k} \right\rfloor C_k \right\} \geq \left(1 - \sum_{\forall \tau_k \in \tau : k \leq i} \frac{C_k}{T_k} \right) T_i \quad (9)$$

$$\textit{where} \qquad M(i,j) = \left\{ n_j T_j : \left\lceil \frac{T_i}{T_j} \right\rceil \leq n_j \leq \left\lfloor \frac{L^*}{T_j} \right\rfloor \right\}$$

PROOF. Assume, all the tasks are sorted in non-decreasing order of their periods/deadlines. To prove the inequality given in Equation 9, we need to show that for all the deadlines between the first deadline of task τ_i and the hyper-period L^*, the procrastination interval computed with PDBF is greater than or equal to Z_i. Jejurikar et al. [20] computes Z_i on the deadline of the task under consideration. To compare these two approaches, their functions are interpolated for all points in the demand bound function. To achieve this, let us consider the example depicted in Figure 2, a straight line is drawn between two points $A(T_i, T_i \sum_{\forall \tau_k \in \tau : k \leq i} \frac{C_k}{T_k})$ and $B(L^*, L^* \sum_{\forall \tau_k \in \tau : k \leq i} \frac{C_k}{T_k})$ as shown in Figure 4. This figure illustrates the approximation by the straight line while the actual demand with the staircase function. Note: Figure 4 only shows it for χ_2 and Z_2. The slope of this line is equal to $\sum_{\forall \tau_k \in \tau : k \leq i} U_k$. To demonstrate that $\chi_i \geq Z_i$, it is sufficient to prove this inequality in $[T_i, L^*]$ (see Theorem 2). This interval is divided into two cases.

a) At time instances T_i and L^*, i.e., the deadline of τ_i and the hyper-period respectively.

b) An interval between time instant T_i and L^*, i.e., (A, B).

Case a) At the first time instant T_i, Equation 10 compares the two approaches.

$$T_i - \sum_{\forall \tau_k \in \tau : k \leq i} \left\lfloor \frac{T_i}{T_k} \right\rfloor C_k \geq \left(1 - \sum_{\forall \tau_k \in \tau : k \leq i} \frac{C_k}{T_k} \right) T_i \quad (10)$$

$$\Leftarrow \quad - \sum_{\forall \tau_k \in \tau : k \leq i} \left\lfloor \frac{T_i}{T_k} \right\rfloor C_k \geq - \sum_{\forall \tau_k \in \tau : k \leq i} \frac{C_k}{T_k} T_i$$

$$\Leftarrow \quad \cancel{\sum_{\forall \tau_k \in \tau : k \leq i}} \left\lfloor \frac{T_i}{T_k} \right\rfloor \cancel{C_k} \leq \cancel{\sum_{\forall \tau_k \in \tau : k \leq i}} \frac{T_i}{T_k} \cancel{C_k}$$

$$\Leftarrow \quad \left\lfloor \frac{T_i}{T_k} \right\rfloor \leq \left(\frac{T_i}{T_k} \right) \quad (11)$$

Equation 11 shows that at time instant T_i, Equation 9 holds. The same reasoning can be applied at time instant L^* (i.e., by replacing the T_i with L^* in Equation 10).

Case b: As already mentioned in the beginning of this proof, the demand of Jejurikar et al. [20] in an interval (A, B) is computed with a straight line of slope $\sum_{\forall \tau_k \in \tau : k \leq i} U_k$ and is compared against DBF at all deadlines. The equation of the line is $y = mx + c$, where m is a slope and c is a y-*intercept*. The y-*intercept* is zero (i.e., $c = 0$) as the line passes through the origin. Hence, the demand determined through the Jejurikar et al. [20] method is given in Equation 12.

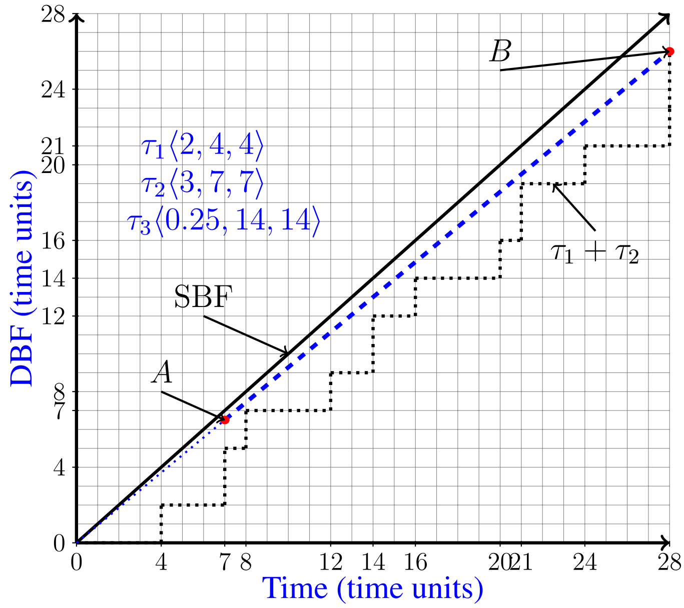

Figure 4: Procrastination Interval for τ_2

$$y = x \sum_{\forall \tau_k \in \tau : k \leq i} \frac{C_k}{T_k} \quad (12)$$

Now consider any deadline that lies in between T_i and L^* and then compare its y-*coordinate* to show that the demand of such deadlines lies below or on the line as the one given in Equation 12. Assume $t \in M(i,j) = \{n_j T_j : \left\lceil \frac{T_i}{T_j} \right\rceil < n_j < \left\lfloor \frac{L^*}{T_j} \right\rfloor \}$. $M(i,j)$ describes the set of all the deadlines between T_i and L^*. As such $n_j T_j$ will be a deadline in an interval (A, B) and its demand is $\sum_{\forall \tau_k \in \tau : k \leq i} \left\lfloor \frac{n_j T_j}{T_k} \right\rfloor C_k$ (Equation 6). Let us put the deadline $n_j T_j$ in the x-*coordinate* of Equation 12 to get the resulting demand of Jejurikar et al. [20] and compare it against $\sum_{\forall \tau_k \in \tau : k \leq i} \left\lfloor \frac{n_j T_j}{T_k} \right\rfloor C_k$ as given in Equation 13.

$$y = n_j T_j \sum_{\forall \tau_k \in \tau : k \leq i} \frac{C_k}{T_k} \geq \sum_{\forall \tau_k \in \tau : k \leq i} \left\lfloor \frac{n_j T_j}{T_k} \right\rfloor C_k \quad (13)$$

$$\Leftarrow \quad \sum_{\forall \tau_k \in \tau : k \leq i} \frac{n_j T_j}{T_k} C_k \geq \sum_{\forall \tau_k \in \tau : k \leq i} \left\lfloor \frac{n_j T_j}{T_k} \right\rfloor C_k$$

$$\Leftarrow \quad \frac{n_j T_j}{T_k} \geq \left\lfloor \frac{n_j T_j}{T_k} \right\rfloor \quad (14)$$

Equation 14 is always true as $x \geq \lfloor x \rfloor$, $\forall x$. Thus, the curve of DBF is always below or on the line for all the deadlines in any interval (A, B).

As the demand of Jejurikar et al. [20] method for all deadlines in the interval $[A; B]$ (case a and b) are greater than or equal to DBF, the lemma follows. $\square$

4.4 Minimum Idle interval Improvements

The minimum bound on the idle period in the schedule is an important metric in procrastination scheduling to select the most efficient sleep state S_n offline. It is the length of the shortest idle interval in the schedule and all the idle intervals will be greater than or equal to this bound. To reduce the online complexity, a processor can choose its sleep state based on this interval that minimises the energy consumption in sleep state while respecting the temporal constraint. By maximising the minimum bound on the idle period, system increase the chance to use the better sleep states (when system has more than one sleep state [1]) which in turn reduces the energy consumption. Therefore, it is also important to maximise the minimum bound on the idle interval.

LEMMA 2. *Given tasks in τ are ordered in a non-decreasing order of their relative deadlines, the minimum idle period guaranteed by* PDBF *scheme is always greater than or equal to the minimum idle period guaranteed by Jejurikar et al. [20].*

PROOF. Assume all the tasks are sorted in non-decreasing order of their periods/deadlines. The minimum procrastination interval Z_{min} determined through Jejurikar et al. [20] algorithm is equal to $Z_{min} = \min_{\forall \tau_i \in \tau} Z_i$. Similarly, the minimum idle period guaranteed by the PDBF scheme $\chi_{min} = \min_{\forall \tau_i \in \tau} \chi_i$. To prove the above mentioned lemma, one needs to prove Equation 15.

$$\min_{\forall \tau_j \in \tau, \forall t \in M(i,j)} \left\{ t - \sum_{\forall \tau_k \in \tau : k \le i} \left\lfloor \frac{t}{T_k} \right\rfloor C_k \right\} \ge \min_{\forall \tau_i \in \tau} \left(1 - \sum_{\forall \tau_k \in \tau : k \le i} \frac{C_k}{T_k} \right) T_i \quad (15)$$

$$\text{where} \qquad M(i,j) = \left\{ n_j T_j : \left\lceil \frac{\min_{\forall \tau_i \in \tau} T_i}{T_j} \right\rceil \le n_j \le \left\lfloor \frac{L^*}{T_j} \right\rfloor \right\}$$

In order to prove this inequality, we have to show that for $t \le L^*$ the demand of the given task-set will remain below or will be equal to the demand computed by Jejurikar et al. [20] method, where $t \in M(i,j) = \left\{ n_j T_j : 1 \le n_j \le \left\lfloor \frac{L^*}{T_j} \right\rfloor \right\}$. In order words, all the deadlines are checked for the difference. To interpolate the demand computed by Jejurikar et al. the demand on the neighbouring deadlines of a task are connected with a straight line. Finally, the demand beyond the last period is extended with a line having a slope equal to the system utilisation. Figure 5 shows the demand of the given example with both DBF and the procrastination algorithm proposed by [20]. For Jejurikar et al. [20] algorithm, the demand of the tasks computed on their first deadline are represented with A, B and C points. Points A and B are connected with a straight line to compare against all the deadlines in the DBF happening in between these two points. Similarly, B and C points are connected, and the demand beyond C for procrastination algorithm is extended with a line having a slope equal to the utilisation of the task-set.

Since the DBF needs to be checked at more instances than A, B and C in the procrastination algorithm, we need to consider constraints. The objective is to find the minimal distance with the unity line and the demand. For all intervals between successive points A, B and C, it is true that the smallest gap between the unity line and the demand within these intervals can be found either of the two delimiting points (for example, for interval $[A, B]$, the smallest gap can either occur at A or B). Since $U \le 1$, it is evident that beyond the largest period, the largest gap can be found at the largest period point. In order to demonstrate that the gap computed with the DBF based value is always greater than or equal to that of procrastination algorithm [20] it is sufficient to show that the DBF test dominates in the following cases.

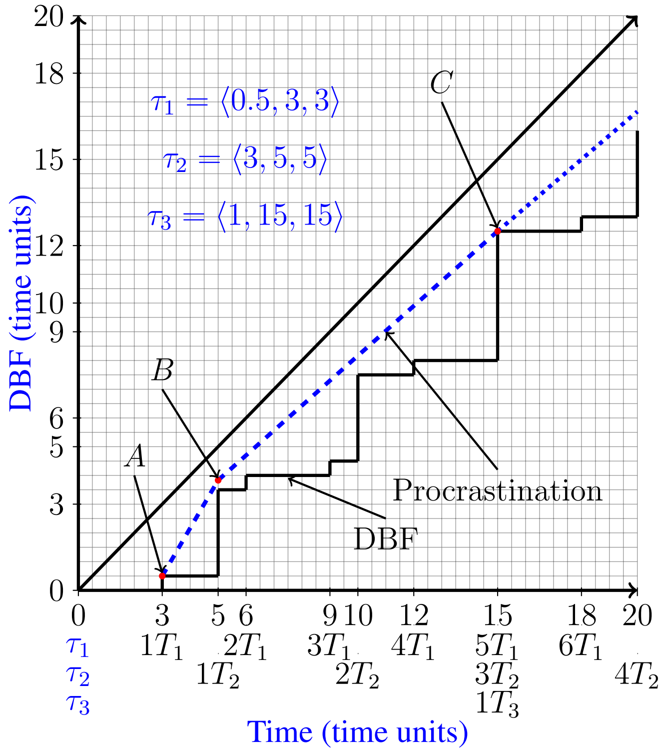

Figure 5: DBF vs SRA

a) First deadline of every task

b) The demand computed by the DBF is always smaller than the connecting lines of the first deadline of all tasks.

Case a) To get the first deadline of every task, we set $t = T_i$ in Equation 15,

$$\min_{\forall \tau_i \in \tau, t = T_i} \left\{ T_i - \sum_{\forall \tau_k \in \tau} \left\lfloor \frac{T_i}{T_k} \right\rfloor C_k \right\} \ge \min_{\forall \tau_i \in \tau} \left\{ T_i - \sum_{\forall \tau_k \in \tau : k \le i} \left(\frac{T_i}{T_k} \right) C_k \right\}$$

$$\Leftarrow \qquad - \sum_{\forall \tau_k \in \tau} \left\lfloor \frac{T_i}{T_k} \right\rfloor C_k \ge - \sum_{\forall \tau_k \in \tau : k \le i} \left(\frac{T_i}{T_k} \right) C_k$$

$$\Leftarrow \sum_{\forall \tau_k \in \tau : k > i} \left\lfloor \frac{T_i}{T_k} \right\rfloor C_k + \sum_{\forall \tau_k \in \tau : k \le i} \left\lfloor \frac{T_i}{T_k} \right\rfloor C_k \le \sum_{\forall \tau_k \in \tau : k \le i} \left(\frac{T_i}{T_k} \right) C_k$$

$$\Leftarrow \qquad 0 + \cancel{\sum_{\forall \tau_k \in \tau : k \le i}} \left\lfloor \frac{T_i}{T_k} \right\rfloor \cancel{C_k} \le \cancel{\sum_{\forall \tau_k \in \tau : k \le i}} \left(\frac{T_i}{T_k} \right) \cancel{C_k}$$

$$\text{as } \left\lfloor \frac{T_i}{T_k} \right\rfloor = 0, \forall\, T_i < T_k$$

$$\Leftarrow \qquad \left\lfloor \frac{T_i}{T_k} \right\rfloor \le \left(\frac{T_i}{T_k} \right) \qquad (16)$$

Equation 16 shows for the first case that Equation 15 holds.

Case b) Suppose that τ_{i-1} is the task preceding τ_i. This case checks Equation 15 for all the deadlines that exist between T_{i-1} and T_i, i.e.,

$$t \in M(i,j) = \left\{ n_j T_j : \left\lceil \frac{T_{i-1}}{T_j} \right\rceil < n_j < \left\lfloor \frac{T_i}{T_j} \right\rfloor, \forall \tau_i \in \tau \right\}.$$

Equation 17 is the general form of the equation of a line between two points (x_1, y_1) and (x_2, y_2). In the representation of the DBF, the $x\text{-}axis$ and $y\text{-}axis$ represent the time and the demand, respectively. Let us assume the coordinates at the deadlines of τ_{i-1} and τ_i are $(x_1, y_1) = \left(T_{i-1}, \sum_{\forall\tau_k\in\tau:k\leq i-1} (\frac{C_k}{T_k})T_{i-1}\right)$ and $(x_2, y_2) = \left(T_i, \sum_{\forall\tau_k\in\tau:k\leq i} (\frac{C_k}{T_k})T_i\right)$, respectively. To find the equation between these two points, substitute their coordinates into Equation 17 correspondingly as shown in Equation 18.

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1 \tag{17}$$

$$= \frac{\left(\sum_{\forall\tau_k\in\tau:k\leq i} \frac{C_k}{T_k}T_i - \sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{C_k}{T_k}T_{i-1}\right)}{T_i - T_{i-1}}(x - T_{i-1}) + \sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{C_k}{T_k}T_{i-1} \tag{18}$$

$$= \frac{\left(\left(\sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{C_k}{T_k} + \frac{C_i}{T_i}\right)T_i - \sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{C_k}{T_k}T_{i-1}\right)}{T_i - T_{i-1}}(x - T_{i-1}) + \sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{C_k}{T_k}T_{i-1}$$

$$= \left(\sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{C_k}{T_k} + \frac{C_i}{T_i - T_{i-1}}\right)(x - T_{i-1}) + \sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{C_k}{T_k}T_{i-1}$$

$$= \left(\sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{C_k}{T_k} + \frac{C_i}{T_i - T_{i-1}}\right)x - \frac{C_iT_{i-1}}{T_i - T_{i-1}} \tag{19}$$

Now consider any deadline that lies in between the deadlines of τ_{i-1} and τ_i (i.e., between (x_1, y_1) and (x_2, y_2)). It is shown that the demand ($y\text{-}coordinate$) of such deadlines will be below or on the line given in Equation 19. To this end, let us say that any deadline between (x_1, y_1) and (x_2, y_2) is specified by $(x_m, y_m) \stackrel{\text{def}}{=} \left(n_jT_j, \sum_{\forall\tau_k\in\tau,t=n_jT_j} \left\lfloor\frac{n_jT_j}{T_k}\right\rfloor C_k\right)$. Substitute the $x\text{-}coordinate$ of this selected point (x_m, y_m) into Equation 19 and compare the resulting value of the $y\text{-}coordinate$ with its y_m. If it is greater than or equal to y_m then DBF is below or on the line. The resulting expression is shown in Equation 20.

$$n_jT_j\left(\sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{C_k}{T_k} + \frac{C_i}{T_i - T_{i-1}}\right) - \frac{C_iT_{i-1}}{T_i - T_{i-1}} \geq \sum_{\forall\tau_k\in\tau,t=n_jT_j} \left\lfloor\frac{n_jT_j}{T_k}\right\rfloor C_k \tag{20}$$

Point (x_m, y_m) is in between T_{i-1} and T_i, therefore the factor $\sum_{\forall\tau_k\in\tau,t=n_jT_j} \lfloor\frac{n_jT_j}{T_k}\rfloor C_k$ can be rewritten as $\sum_{\forall\tau_k\in\tau:k\leq i-1} \lfloor\frac{n_jT_j}{T_k}\rfloor C_k$. Hence, it follows that

$$n_jT_j\left(\sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{C_k}{T_k} + \frac{C_i}{T_i - T_{i-1}}\right) - \frac{C_iT_{i-1}}{T_i - T_{i-1}} \geq \sum_{\forall\tau_k\in\tau:k\leq i-1} \left\lfloor\frac{n_jT_j}{T_k}\right\rfloor C_k$$

$$\Longleftarrow \sum_{\forall\tau_k\in\tau:k\leq i-1} \frac{n_jT_j}{T_k}C_k + \frac{C_i}{T_i - T_{i-1}}(n_jT_j - T_{i-1}) \geq \sum_{\forall\tau_k\in\tau:k\leq i-1} \left\lfloor\frac{n_jT_j}{T_k}\right\rfloor C_k \tag{21}$$

Obviously, $n_jT_j - T_{i-1}$ is greater than 0 as $n_jT_j > T_{i-1}$. Hence, all the deadlines such that

$$\forall\tau_k \in \tau, t \in M(i, k) = \{T_{i-1} \leq n_kT_k \leq T_i\}$$

lie below the line represented by Equation 19.

As the difference computed between the supply and the demand for all deadlines (case a and b) are greater than or equal to their corresponding difference computed through Jejurikar et al. [20] algorithm, lemma follows. □

4.5 Extensions to the constrained deadline task model

The state-of-the-art procrastination algorithms [20,21] cannot be extended for constrained deadline task model ($D_i \leq T_i$) in their current form. One of the advantages of the PDBF approach is it straight forward extension to this model. For the constrained deadline task model, Equation 7 can be rewritten in its general form by replacing $\text{DBF}_I(\tau_k, t)$ with $\text{DBF}(\tau_k, t)$ as given Equation 22, where the set $M(i, j)$ is substituted by

$$M_1(i, j) = \left\{n_jD_j : \left\lfloor\frac{T_i - D_i}{T_j}\right\rfloor + 1 \leq n_j \leq \left\lfloor\frac{L^* - D_i}{T_j}\right\rfloor + 1\right\}.$$

Similarly, the minimum idle interval for constrained deadline task model is given in Equation 23, where

$$M_2(i, j) = \left\{n_jD_j : 1 \leq n_j \leq \left\lfloor\left\lfloor\frac{L^*}{T_j}\right\rfloor\right\rfloor\right\}.$$

$$\chi_i = \min_{\forall\tau_j \in \tau : j \leq i, \forall t \in M_1(i,j)} \left\{t - \sum_{\forall\tau_k\in\tau:k\leq i} \text{DBF}(\tau_k, t)\right\} \tag{22}$$

$$\chi_{min} = \min_{\forall\tau_j \in \tau, \forall t \in M_2(i,j)} \left\{t - \sum_{\forall\tau_k\in\tau:k\leq i} \text{DBF}(\tau_k, t)\right\} \tag{23}$$

Equation 23 aligns with the results provided by of Chetto et al. [10, 11, 28] on the slack time estimation to schedule the aperiodic task in the presence of periodic task-set.

4.6 Optimality of PDBF

In this section, it is shown that the minimum feasible sleep interval of a task-set and the procrastination interval of individual tasks determined through PDBF is optimal, i.e., *maximal without violating any temporal constraint*. The optimal procrastination interval as well as the minimum feasible sleep interval can be determined by using techniques borrowed from the sensitivity analysis framework [13] or Chetto et al. [11]. However, due to space limitations, the DBF-based analysis is used to circumvent this issue. The interested readers are directed to the technical report [2] for a formal proof using the sensitivity analysis.

THEOREM 3. *The minimum idle period determined by the* PDBF *approach for a constrained deadline task-set is optimal.*

PROOF. Since sleep transitions are taken in idle intervals, only the critical instant has to be considered. Lemma 2 demonstrates that $\chi_{min} \geq Z_{min}$ and the chosen sleep interval is safe i.e., no deadline is missed in the resulting schedule. Hence, χ_{min} is not optimistic. At the same time the DBF based analysis demonstrates a concrete scheduling scenario. Thus, χ_{min} is clearly not pessimistic, as the derived value by the PDBF approach can actually occur. Since the derived sleep interval χ_{min} is at the same time neither pessimistic nor optimistic, it is safe and optimal, thus the theorem follows. □

THEOREM 4. *The procrastination interval determined by the* PDBF *approach for individual task in a constrained deadline task model is optimal.*

PROOF. In this case, instead of considering the whole task-set τ, only the set of tasks with a priority greater than or equal the current one are taken into account. Theorem 2 shows that it is sufficient to consider only the set of deadlines after the first deadline of the task under analysis, including the first deadline of the task as well. Afterwards, the proof follows the same principle as that of Theorem 3 where the given procrastination interval has been shown neither optimistic nor pessimistic. □

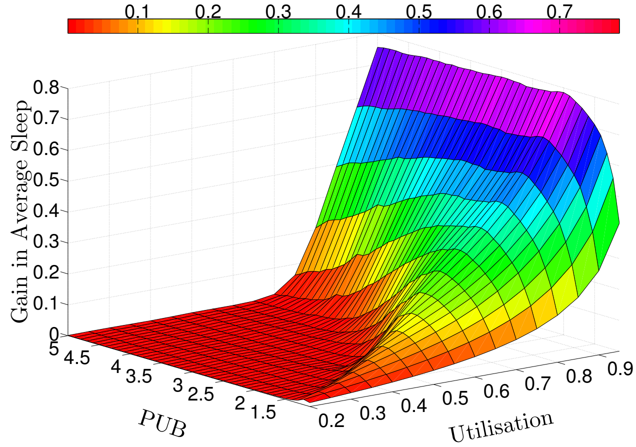

Figure 6: T_{max} variation (Sleep Interval)

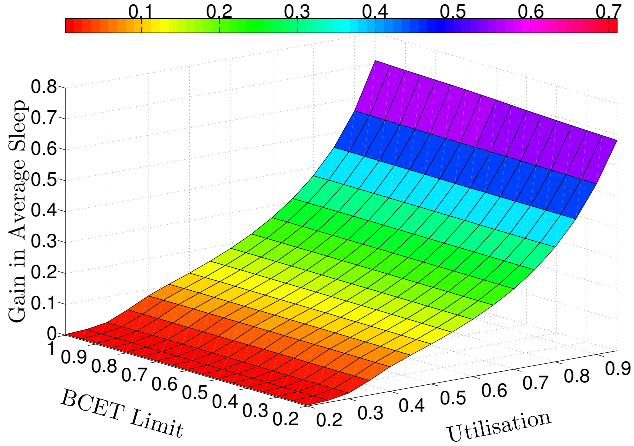

Figure 7: Variation in C^b (Sleep Interval)

5. COMPLEXITY COMPARISON

The complexity of the state-of-the-art approaches as well as that of the proposed approach to compute the procrastination interval can be categorised as offline and online complexity. The offline complexity of the LC-EDF algorithm [21] is zero as all the computations are performed online. Jejurikar et al. [20] method has an offline complexity of $O(\ell^2)$. The PDBF approach has an offline complexity of $O(\ell h)$, where $h = \sum_{\forall \tau_i \in \tau} \frac{L^*}{T_i}$ is the number of jobs in the hyper-period.

The online complexity of the PDBF approach and Jejurikar's method is the same and equals to $O(\ell)$. The LC-EDF algorithm has an online complexity of $O(\ell^2)$. This implies that the external hardware designed for Jejurikar's method can also be used for the PDBF approach as both work on the same principle. On the one hand, the procrastination based energy saving algorithms proposed for Jejurikar's method can be easily integrated with the PDBF approach without any extra effort. On the other hand, the LC-EDF algorithm needs complex external hardware due to the mechanism used to compute procrastination interval online.

6. EVALUATION

The discrete event simulator SPARTS (Simulator for Power Aware and Real-Time Systems) [23, 24] is used to evaluate the effectiveness of the PDBF approach. SPARTS is used with the parameters mentioned in Table 1, where underlined values are the default values if not mentioned otherwise in the description of the experiment. The parameters C^b and Γ are used to generate wide variety of different tasks and their subsequent varying jobs. Suppose, C_i^b and Γ_i are the helper variables to provide the bounds on the best-case execution time (BCET) and sporadic delay of a task respectively. Then C_i^b and Γ_i are randomly selected for the given tasks in interval $C_i \cdot [C^b, 1]$ and $T_i \cdot [\Gamma, 1]$ respectively. Similarly, the actual execution time and sporadic delay of the individual jobs are randomly selected from the following intervals $[C_i^b, C_i]$ and $[T_i, T_i + \Gamma_i]$ respectively. The periods of the task-set are chosen from an interval, $T_{min}[1, \mathrm{PUB}]$, where T_{min} it the lower bound and PUB (Period Upper Bound) is the variable used to define the upper bound on the interval. Each task-set with different parameters mentioned in Table 1 is simulated for 100 times with different seed values to the random number generator and averaged. The simulation time of each task-set is 100sec.

The SRA algorithm [19] is an energy saving approach that takes procrastination intervals of the tasks determined through Jejurikar's method as an input. For a fair comparison, the same algorithm is used by just replacing the input phase with PDBF determined procrastination intervals. For simplicity sake, it is assumed that all the slack in the schedule (spare capacity) is reserved for the shutdown of the processor. Both variations of SRA are implemented in SPARTS and their sleep state is selected offline based on their respective minimum idle interval. It has already been shown in the state-of-the-art that SRA performs better than LC-EDF, hence, this section restrict the comparison to SRA.

Table 1: Overview of Simulator Parameters

Parameters	Values
Task-set sizes $\lvert\tau\rvert \in$	$\{10, 20, \ldots, \underline{50}, \ldots, 100\}$
$T_{min} \in$	$\{\underline{30}, 40, \ldots, 100\}$
PUB $\in$	$\{1.1, 1.2, \ldots, \underline{1.5}, \ldots, 5\}$
BCET Limit $C^b \in$	$\{0.2, 0.25, \ldots, \underline{1}\}$
Sporadic Delay Limit $\Gamma \in$	$\{\underline{0}, 0.05, \ldots, 1\}$

Table 2: Different Sleep States Parameters

No.	Power Mode	$tr_n(\mu s)$	$bet_n(\mu s)$	P_n	E_n
1.	Doze	5	225	3.7	42
2.	Nap	100	450	2.6	950
3.	Sleep	200	800	2.2	1980
4.	Deep Sleep	500	1400	0.6	5750
5.	Typical	0	0	4.7	0
6.	Maximum	-	-	12.1	-

The power model used for simulations is based on the Freescale PowerQUICC III Integrated Communications Processor MPC8536 [27]. The power consumption values are taken from its data sheet for different modes (Maximum, Typical, Doze, Nap, Sleep, Deep Sleep). The core frequency of 1500 MHz and core voltage of 1.1V is used for all the experiments. The transition overheads are not mentioned in their data sheet, therefore, they are assumed for four different sleep states. The transition overhead of the typical mode that corresponds to the idle state in our system model is considered negligible. The power values given in Table 2 sum up core power and platform power consumption. More details are available in the reference manual [27].

Figure 6 presents the gain of PDBF over SRA with respect to average sleep interval for different values of U and PUB. The average sleep interval is computed by accumulating the idle time in the scheduling and dividing it by the number of sleep states. The gain of PDBF increases with an increase in system utilisation. Furthermore, the gain also increases by widening the interval to select T_i of the tasks. At low utilisation PDBF and SRA have

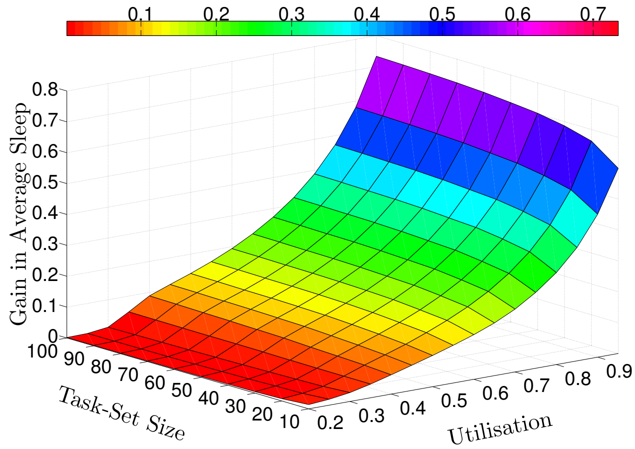

Figure 8: Variation in $|\tau|$ (Sleep Interval)

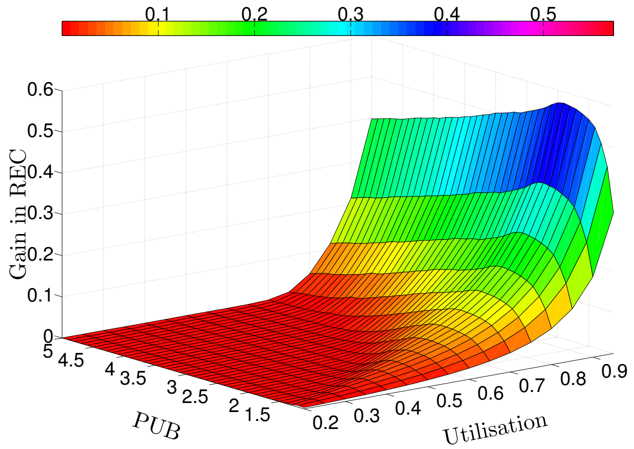

Figure 9: Variation in T_{max} (REC)

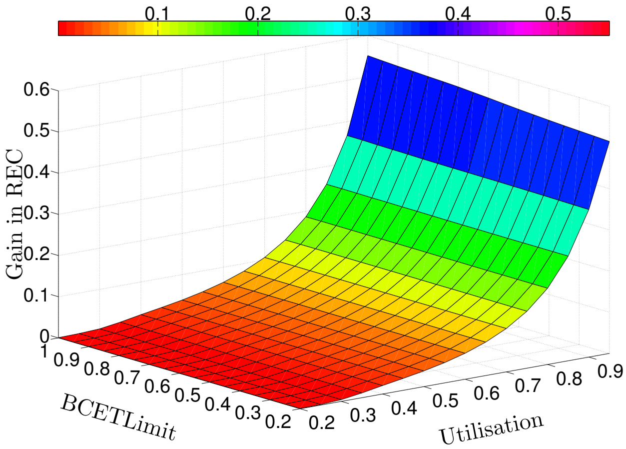

Figure 10: Variation in C^b (REC)

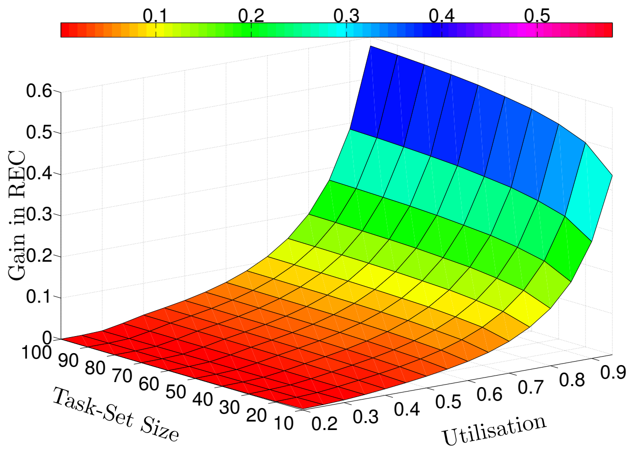

Figure 11: Variation in $|\tau|$ (REC)

enough slack to initiate longer sleep intervals. However, with an increase in system utilisation, the slack in the system decreases, and the procrastination intervals lengths have a high impact on the sleep intervals. Another reason for a high gain at high utilisation is the difference of minimum idle interval between SRA and PDBF. It has been shown in Lemma 2 that $\chi_{min} \geq Z_{min}$. Therefore, SRA starts to lose efficient sleep states at high utilisation, causing its frequent switching. In the best case, the average increase in the sleep interval is approximately 75%.

The gain in average sleep interval is also computed by varying the utilisation against the BCET Limit C^b as shown in Figure 7. Mostly, the gain occurs due to an increase in system utilisation, while the variation in C^b has a minute effect at a very high utilisation of 0.95. As both algorithms use the same mechanism to manage the slack, the difference is negligible. The change in sporadic delay limit Γ is also observed in the experiments against different values of U. The effect of Γ is negligible as well. The variation in task-set size is demonstrated in Figure 8 against different values of U. In the best case (i.e., $|\tau| = 100$), the gain reaches 75%. It is evident that the increase in task-set size increases the gain in average sleep intervals. This can be explained as follows. The procrastination interval of a high priority task is always bounded by its low priority tasks. The difference between the procrastination intervals of different tasks between PDBF and SRA has a cascading effect. For instance, a low priority task τ_i having a procrastination interval Z_i smaller than that of a high priority task will have its Z_i scaled down due to Equation 3. If $Z_i < \chi_i$, then not only the difference exists at level τ_i but also $\forall \tau_k : k < i$. Larger task-set size has higher probability to get this cascading effect.

The active energy consumption of the system is the same in SRA and PDBF as only a single active state is assumed in this work. The difference comes in the energy consumption of the system in idle intervals and termed as reducible energy consumption (REC). The gain of PDBF over SRA with respect to REC is compared for different parameters against system utilisation as demonstrated in Figure 9, Figure 10 and Figure 11. In the best case, the gain in REC is approximately 55%. All the graphs have more or less similar trends as explained in the description of their corresponding results with average sleep intervals.

7. CONCLUSIONS

The PDBF approach is optimal to compute the procrastination intervals of a given task-set. It has been shown theoretically and experimental that PDBF dominates over SRA. The average sleep interval can be increased up to 75%, while the REC can be raised up to 55%. The online complexity of PDBF is the same when compare to that of SRA. The relaxation to the constrained deadline task model is an additional benefit of the proposed approach. In the future, it is intended to extend it to heterogeneous multicore platforms and also to the dependent task model.

8. REFERENCES

[1] M. A. Awan and S. M. Petters. Enhanced race-to-halt: A leakage-aware energy management approach for dynamic priority systems. In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems*, pages 92–101. IEEE Computer Society, 2011.

[2] M. A. Awan, P. M. Yomsi, and S. M. Petters. Optimal procrastination interval upon uniprocessors, CISTER-TR-130608, 2013. https://www.cister.isep.ipp.pt/people/Muhammad%2BAli%2BAwan/publications/.

[3] P. Balbastre, I. Ripoll, and A. Crespo. Analysis of window-constrained execution time systems. *Journal of Real–Time Systems*, 35(2):109–134, 2007.

[4] P. Balbastre, I. Ripoll, and A. Crespo. Minimum deadline calculation for periodic real-time tasks in dynamic priority systems. *IEEE Transactions on Computers*, 57(1):96–109, 2008.

[5] P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 364–367, Miami, Florida, Jan. 2006. ACM.

[6] S. K. Baruah, L. E. Rosier, and R. R. Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Journal of Real–Time Systems*, 1990.

[7] E. Bini. Modeling preemptive edf and fp by integer variables. In *Proceedings of the 4th Multidisciplinary International Scheduling Conference*, Dublin, Ireland, August 2009.

[8] J.-J. Chen and T.-W. Kuo. Procrastination for leakage-aware rate-monotonic scheduling on a dynamic voltage scaling processor. *SIGPLAN Notices*, 41:153–162, June 2006.

[9] H. Cheng and S. Goddard. Integrated device scheduling and processor voltage scaling for system-wide energy conservation. In *Proceedings of the 2005 Workshop on Power Aware Real-time Computing*, pages 24–29, Sept. 2005.

[10] H. Chetto and M. Chetto. Some results of the earliest deadline scheduling algorithm. *IEEE Transactions on Software Engineering*, 15(10):1261–1269, 1989.

[11] M. Chetto. Results on the slack of a periodic task set. Technical report, ri 2008_5, Institut de Recherche en Communications et en Cybernétique de Nantes, june 2008.

[12] V. Devadas and H. Aydin. On the interplay of dynamic voltage scaling and dynamic power management in real-time embedded applications. In *Proceedings of the 8th International Conference on Embedded Software*, pages 99–108, Atlanta, GA, USA, 2008. ACM.

[13] L. George and J.-F. Hermant. Characterization of the space of feasible worst-case execution times for earliest-deadline-first scheduling. In *Journal of Aerospace Computing, Information, and Communication*, volume 6:11, pages 604–623, 2009.

[14] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Adaptive dynamic power management for hard real-time systems. In *Proceedings of the 30th IEEE Real-Time Systems Symposium*, pages 23–32, Washington, DC, USA, 2009. IEEE Computer Society.

[15] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo. Applying real-time interface and calculus for dynamic power management in hard real-time systems. *Journal of Real–Time Systems*, 47(2):163–193, 2011.

[16] S. Irani, S. Shukla, and R. Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3(4):41, 2007.

[17] ITRS. International technology roadmap for semiconductors, 2010 update, overview, 2010.

[18] R. Jejurikar and R. Gupta. Procrastination scheduling in fixed priority real-time systems. In *Proceedings of the Conference on Language, Compiler and Tool Support for Embedded Systems'04*, pages 57–66, Washington DC, 2004.

[19] R. Jejurikar and R. Gupta. Dynamic slack reclamation with procrastination scheduling in real-time embedded systems. In *Proceedings of the 42nd Design Automation Conference*, pages 111–116, Anaheim, 2005.

[20] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *Proceedings of the 41st Design Automation Conference*, San Diego, 2004.

[21] Y.-H. Lee, K. Reddy, and C. Krishna. Scheduling techniques for reducing leakage power in hard real-time systems. In *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Jul. 2003.

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

[23] B. Nikolic, M. A. Awan, and S. M. Petters. SPARTS: Simulator for power aware and real-time systems. In *Proceedings of the 8th IEEE International Conference on Embedded Software and Systems*, pages 999–1004, Changsha, China, Nov. 2011. IEEE.

[24] B. Nikolic, M. A. Awan, and S. M. Petters. SPARTS: Simulator for power aware and real-time systems, 2011. http://www.cister.isep.ipp.pt/projects/sparts/.

[25] L. Niu and G. Quan. Reducing both dynamic and leakage energy consumption for hard real-time systems. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 140–148, Washington DC, USA, 2004. ACM.

[26] R. Pellizzoni and G. Lipari. Feasibility analysis of real-time periodic tasks with offsets. *Journal of Real–Time Systems*, 30:105–128, 2005.

[27] F. Semiconductor. *MPC8536E PowerQUICC III Integrated Processor Hardware Specifications*. Number: MPC8536EEC,Rev. 5, 09/2011.

[28] M. Silly. The edl server for scheduling periodic and soft aperiodic tasks with resource constraints. *Journal of Real–Time Systems*, 17(1):87–111, 1999.

[29] L. Thiele, S. Chakraborty, and M. Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings of the 27th International Symposium on Computer Architecture*, volume 4, pages 101 –104 vol.4, 2000.

[30] L. Thiele, E. Wandeler, and N. Stoimenov. Real-time interfaces for composing real-time systems. In *Proceedings of the 6th International Conference on Embedded Software*, EMSOFT '06, pages 34–43, New York, NY, USA, 2006. ACM.

[31] F. Zhang, A. Burns, and S. Baruah. Sensitivity analysis for edf scheduled arbitrary deadline real-time systems. In *Proceedings of the 16th IEEE Conference on Embedded and Real-Time Computing and Applications*, pages 61–70. IEEE, 2010.