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 (Portuguese 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, since the 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.
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 scheduled 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 a 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 \(\eta_{\text{crit}}\) 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.

Jeejurikar 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 claimed 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 sporadic task \(\tau = (r_i, t_i, c_i)\) is composed of \(\ell\) independent tasks. Each task \(r_i\) generates a potentially infinite sequence of jobs and is characterised by a 3-tuple \(\tau_i = (C_i, D_i, T_i)\), 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 \(\tau_i\). These parameters are real-valued and given with the following interpretation. The \(k^{th}\) job \(j_{i,k}\) of \(\tau_i\) is defined as \(j_{i,k} = (r_{i,k}, c_{i,k}, d_{i,k})\), 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 \(d_{i,k} = r_{i,k} + D_i\) is the absolute deadline. The hyper-period \(L^*\) of \(\tau\) is defined as the least common multiple of the tasks periods, i.e. \(L^* \overset{\text{def}}{=} \text{LCM} \{T_i, T_{i+1}, ..., T_j\}\). The notion of LCM is extended to real numbers as follows:

\[
\text{LCM}(a, b) \overset{\text{def}}{=} \inf \{x \in \mathbb{R} : 3p, q \in \mathbb{N}, x = pa = qb\}
\]

(see [7] for further details). The utilisation of task \(\tau_i\) is \(U_i \overset{\text{def}}{=} \frac{C_i}{T_i}\) and the system utilisation is \(U \overset{\text{def}}{=} \sum_{r_i \in \tau} \frac{C_i}{T_i}\). 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 \(P_s\), respectively. A set of \(N\) sleep states with different characteristics is assumed. Each sleep

\[\text{An algorithm is termed as competitive if its competitive ratio, i.e. ratio between the performance of the algorithm and the optimal offline algorithm, is bounded by a constant number.}\]
state $S_n \equiv (P_n, t_{rn}, E_n)$, 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 $t_{rn}$. 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., $D_i = T_i, \forall t_i \in \tau$, 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 $t_1$ is the first which arrives in a sleep mode and has a deadline $D_1$. The procrastination interval $\Delta_t$ of $t_1$ is computed with the condition $\sum_{r_i \in \tau, r_j \notin \tau} C_i T_i + C_b + \Delta_t T_k = 1$. Suppose $t$ is the current time then the timer is initialised with $t + \Delta_t$ 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 $\delta_b \leq \Delta_t$ time units, $t_1$ arrives with a deadline $D_b < D_1$; a new procrastination interval $t + \Delta_b$ is determined as $\sum_{r_i \in \tau, r_j \notin \tau} C_i T_i + C_b + \Delta_b T_k = 1$. The wake-up timer is reset to $t + \Delta_b$. Similarly, for any other task $t_2$ with the highest priority when compared to the tasks in the ready queue, the procrastination interval $\Delta_2$ 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 $t_2$ and with deadlines longer than $t_2$. In this equation, $\delta_i$ is the interval between an arrival of any job of task $t_i$ (having highest priority at that instant) and any next task arrival having priority higher than $t_i$ in the system’s sleep state.

![Figure 1](image1.png)

**Figure 1:** “Accumulated delays under EDF scheduling [21]”

![Figure 2](image2.png)

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

The limitations of LC-EDF are the increased online complexity to maintain a track of $\delta_i$ and considering the utilisation of the low priority tasks.

$$\sum_{r_i \in \tau, r_j \notin \tau} C_i T_i + \sum_{r_i \in \tau, r_j < t} C_i T_i + C_i + \Delta_t T_k = 1 \quad (1)$$

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 initializes the wake-up timer $\zeta$ with its procrastination interval. The timer $\zeta$ counts down with every clock cycle. If another task (say $\tau_3$) 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 $\eta_k$ is the frequency of the processor. The value of $\eta_k$ 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 $\tau$ are ordered in non-decreasing order of their periods, the procrastination algorithm guarantees all task deadlines if the procrastination interval $Z_t$ of each task $t$, satisfies the following two conditions:

$$\forall \tau_i \in \tau, \quad \frac{Z_t}{T_i} + \sum_{\tau_j \in \tau \leq \tau_i} \frac{1}{\eta_k} \leq 1 \tag{2}$$

and $\forall k < i, \quad Z_k \leq Z_i \tag{3}$

While computing the procrastination interval for task $\tau_i$, Jejurikar et al. [20] only considers the utilisation of the tasks having priority greater than or equal to $\tau_i$ (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, $Z_t$ 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 $Z_3 = 0.75$. 

$$Z_i = \left(1 - \sum_{\forall y_k \in R \leq s_i} \frac{C_k}{T_k}\right) T_i$$  \hspace{1cm} (4)

Figure 2 shows the schedule for the aforementioned example. With a careful observation it can be seen that the procrastination interval of $\tau_1$, $\tau_2$ and $\tau_3$ 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 [61]): The demand for any constrained deadline task $\tau_i$ and positive time $t$, denoted by $DBF(\tau_i, t)$, is the maximum cumulative execution requirement of jobs of task $\tau_i$ in any interval of length $t$. Formally, $DBF(\tau_i, t)$ is presented in Equation 5.

$$\forall t \geq 0, \quad DBF(\tau_i, t) \overset{def}{=} \left[\frac{t - D_i}{T_i}\right] + 1 \cdot C_i$$  \hspace{1cm} (5)

From Equation 5, it is easy to see that $DBF(\tau_i, t)$ 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 $DBF(\tau, t) \overset{def}{=} \sum_{i \in \tau} DBF(\tau_i, t)$. 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 $DBF(\tau_i, t)$ of $\tau_1$, $\tau_2$ and $\tau_3$. 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 utilisation. Indeed when $D_i \leq T_i$, 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 $\tau_i$, the PDBF approach subtracts the demand of the task $\tau_i$ 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 $\tau_i$ 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 $\chi_1$, $\chi_2$ and $\chi_3$ denote 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 $\tau_1$ greater than that of $\tau_2$, therefore, the value of $\chi_1$ 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 $DBF(\tau_i, t)$ of task $\tau_i$ presented in Equation 5 can be rewritten as shown in Equation 6.

$$DBF(\tau_i, t) = \left\lfloor\frac{t}{T_i}\right\rfloor C_i \quad \text{as} \quad t \geq 0$$  \hspace{1cm} (6)

**Theorem 2.** Given tasks in $\tau$ 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 $\tau_i$, denoted by $\chi_i$, is computed with Equation 7 while respecting the condition given in Equation 8.

$$\chi_i = \forall \tau_j \in \tau : j \leq i, \forall t \geq 0 \left\{t - \sum_{\forall y_k \in R \leq s_i} DBF(\tau_k, t)\right\}$$  \hspace{1cm} (7)

$$= \forall \tau_i \in \tau : i \leq j, \forall t \in M(i, j) \left\{t - \sum_{\forall y_k \in R \leq s_j} \left\lfloor\frac{t}{T_i}\right\rfloor C_i\right\}$$

where $M(i, j) = \left\{n \cdot T_j : \left\lfloor\frac{n}{T_i}\right\rfloor \leq n \leq \left\lfloor\frac{n}{T_i}\right\rfloor + 1\right\}$

$$\forall k < i, \chi_k \leq \chi_i$$  \hspace{1cm} (8)

**Proof Sketch.** Suppose a task $\tau_i$ 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 $\tau_{proc}$ 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 $\tau_i$ will have procrastination interval greater than or equal to $\chi_i$. Therefore, $\tau_{proc}$ 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 $\tau_{proc}$ (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 $[D_i, L_i]$ as the procrastination interval of a task is only considered when it has the highest priority on its arrival in the ready queue.
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 \( \tau \) are ordered in a non-decreasing order of their relative deadlines, the procrastination interval \( \chi_{i} \) for any task \( \tau_{i} \), 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.,

\[
\min_{\forall \tau_{i} \in \tau: j \leq \tau_{i} \in M(i,j)} \left\{ 1 - \sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in \chi_{i}} \frac{C_{k}}{T_{k}} \right\} \geq \left( 1 - \sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in Z_{i}} \frac{C_{k}}{T_{k}} \right) T_{i} \tag{9}
\]

where \( M(i,j) = \{ n_{j}T_{j} : \frac{T_{i}}{T_{j}} \leq n_{j} \leq \frac{L^{*}}{T_{j}} \} \).

**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 \( \tau_{i} \) 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 interleaved 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 \( A(T_{i}, T_{i} + \sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in C_{k}} C_{k} / T_{k}) \) and \( B(L^{*}, L^{*} + \sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in C_{k}} C_{k} / T_{k}) \) 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 \( \chi_{i} \) and \( Z_{i} \). The slope of this line is equal to \( \sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in U_{k}} \). 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 \( \tau_{i} \) 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.

\[
T_{i} - \sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in C_{k}} \frac{T_{k}}{T_{k}} \geq \left( 1 - \sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in C_{k}} \frac{C_{k}}{T_{k}} \right) T_{i} \tag{10}
\]

\[
\Rightarrow \sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in C_{k}} \frac{T_{k}}{T_{k}} C_{k} \geq \sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in C_{k}} \frac{C_{k}}{T_{k}} T_{i}
\]

\[
\Rightarrow \frac{T_{k}}{T_{k}} \leq \frac{n_{j}T_{j}}{T_{j}} C_{k} \tag{11}
\]

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 \( \sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in U_{k}} \) and is compared against DBF at all deadlines. The equation of the line is

\[
y = mx + c, \text{ where } m \text{ is a slope and } c \text{ is a y-intercept}. \tag{12}
\]

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 \( t \in M(i,j) = \{ n_{j}T_{j} : \frac{T_{i}}{T_{j}} \leq n_{j} \leq \frac{L^{*}}{T_{j}} \} \). \( M(i,j) \) 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

\[
\sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in C_{k}} \frac{n_{j}T_{j}}{T_{k}} C_{k} \tag{Equation 6}
\]

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

\[
\sum_{\forall \tau_{k} \in \tau_{i} \leq \tau_{k} \in C_{k}} \frac{n_{j}T_{j}}{T_{k}} C_{k} \tag{Equation 13}
\]

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 $S_{\text{off}}$. 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 $\tau$ 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 $Z_{\text{min}}$ determined through Jejurikar et al. [20] algorithm is equal to $Z_{\text{min}} = \min_{\forall i \in \tau} Z_i$. Similarly, the minimum idle period guaranteed by the PDBF scheme $\chi_{\text{min}} = \min_{\forall i \in \tau} \chi_i$. To prove the above mentioned lemma, one needs to prove Equation 15.

$$\min_{n_j \in \tau, \forall i \in M(i, j)} \left\{ t - \sum_{\forall k \in \tau} \left( \frac{1}{T_{ik}} \right) C_k \right\} \geq \min_{n_j \in \tau, \forall i \in M(i, j)} \left\{ t - \sum_{\forall k \in \tau} \left( \frac{1}{T_{ik}} \right) C_k \right\} \forall t \leq L^* \tag{15}$$

where

$$M(i, j) = \left\{ n_jT_i : \min_{\forall i \in \tau} \left( \frac{n_jT_i}{T_i} \right) \leq n_j \leq \left( \frac{L^*}{T_i} \right) \right\}$$

In order to prove this inequality, we have to show that for $t \leq L^*$ 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 $t \in M(i, j) = \left\{ n_jT_i : 1 \leq n_j \leq \left( \frac{L^*}{T_i} \right) \right\}$. In order words, all the deadlines are checked for the difference. To interprete 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 $U \leq 1$, 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.

\begin{itemize}
  \item[a)] First deadline of every task
  \item[b)] The demand computed by the DBF is always smaller than the connecting lines of the first deadline of all tasks.
\end{itemize}

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

$$\min_{\forall i \in \tau} \left\{ T_i - \sum_{\forall k \in \tau} \left( \frac{T_i}{T_{ik}} \right) C_k \right\} \geq \min_{\forall i \in \tau} \left\{ T_i - \sum_{\forall k \in \tau} \left( \frac{T_i}{T_{ik}} \right) C_k \right\}$$

\begin{align*}
&\Rightarrow - \sum_{\forall k \in \tau} \left( \frac{T_i}{T_{ik}} \right) C_k \leq - \sum_{\forall k \in \tau} \left( \frac{T_i}{T_{ik}} \right) C_k \\
&\Rightarrow 0 + \sum_{\forall k \in \tau} \left( \frac{T_i}{T_{ik}} \right) \varphi_k \leq \sum_{\forall k \in \tau} \left( \frac{T_i}{T_{ik}} \right) \varphi_k \\
&\Rightarrow \left[ \frac{T_i}{T_k} \right] \geq 0, \forall T_i < T_k \tag{16}
\end{align*}

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

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

$$t \in M(i, j) = \left\{ n_jT_i : \frac{T_{i-1}}{T_i} < n_j < \frac{T_i}{T_k}, \forall T_i \in \tau \right\}$$

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 $x$-axis and $y$-axis represent the time and the demand, respectively. Let us assume the coordinates at the deadlines of $\tau_{i-1}$ and $\tau_i$ are $(x_1, y_1) = \left( T_{i-1}, \sum_{j \in I \cap \tau_{i-1}} \frac{C_k}{T_k} \right)$ and $(x_2, y_2) = \left( T_i, \sum_{j \in I \cap \tau_i} \frac{C_k}{T_k} T_j \right)$, respectively. To find the equation between these two points, substitute their coordinates into Equation 17 correspondingly as shown in Equation 18.

\[ y = \frac{y_2 - y_1}{x_2 - x_1} (x - x_1) + y_1 \quad (17) \]

\[ = \left( \sum_{j \in I \cap \tau_{i-1}} \frac{C_k}{T_k} \right) \frac{T_i - T_{i-1}}{T_i} + \sum_{j \in I \cap \tau_i} \frac{C_k}{T_k} T_j \quad (18) \]

\[ = \left( \sum_{j \in I \cap \tau_{i-1}} \frac{C_k}{T_k} \right) \frac{T_i - T_{i-1}}{T_i} + \sum_{j \in I \cap \tau_i} \frac{C_k}{T_k} T_j \]

\[ = \sum_{j \in I \cap \tau_{i-1}} \frac{C_k}{T_k} + \sum_{j \in I \cap \tau_i} \frac{C_k}{T_k} \frac{T_i - T_{i-1}}{T_i} \quad (19) \]

Now consider any deadline that lies in between the deadlines of $\tau_{i-1}$ and $\tau_i$ (i.e., between $(x_1, y_1)$ and $(x_2, y_2)$). It is shown that the demand (y-coordinate) of such deadlines will be below or on the line given in Equation 19. To this end, let us say that any deadline $(x_1, y_1)$ and $(x_2, y_2)$ is specified by $(x_m, y_m) \text{ def } = \left( n_i T_j, \sum_{j \in I \cap \tau_i} \frac{n_j T_j}{T_k} \right)$. Substitute the $x$-coordinate of this selected point $(x_m, y_m)$ into Equation 19 and compare the resulting value of the y-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.

\[ n_i T_j \left( \sum_{j \in I \cap \tau_i} \frac{C_k}{T_k} \right) - \frac{C_k}{T_k} \frac{T_i - T_{i-1}}{T_i} \geq \sum_{j \in I \cap \tau_i} \frac{n_j T_j}{T_k} C_k \quad (20) \]

Point $(x_m, y_m)$ is in between $T_{i-1}$ and $T_i$, therefore the factor $\sum_{j \in I \cap \tau_i} \frac{n_j T_j}{T_k} C_k$ can be rewritten as $\sum_{j \in I \cap \tau_i} \frac{T_i - T_{i-1}}{T_i} C_k$. Hence, it follows that

\[ n_i T_j \left( \sum_{j \in I \cap \tau_i} \frac{C_k}{T_k} \right) - \frac{C_k}{T_k} \frac{T_i - T_{i-1}}{T_i} \geq \sum_{j \in I \cap \tau_i} \frac{n_j T_j}{T_k} C_k \]

\[ = \sum_{j \in I \cap \tau_i} n_j T_j C_k + \sum_{j \in I \cap \tau_i} \frac{n_j T_j}{T_k} \frac{T_i - T_{i-1}}{T_i} C_k \quad (21) \]

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

\[ \forall \tau_k \in \tau, t \in M(i, k) = \{ T_{i-1} \leq M_k T_k \leq T_i \} \]

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.

\[ \square \]

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 $DBF(\tau_i)$ with $DBF(\tau_i, t)$ as given Equation 22, where the set $M(i, j)$ is substituted by

\[ M_1(i, j) = \left\{ n_j D_j : \frac{T_i - D_j}{T_j} + 1 \leq n_j \leq \left[ \frac{L - D_i}{T_j} \right] + 1 \right\} \]

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

\[ M_2(i, j) = \left\{ n_j D_j : 1 \leq n_j \leq \left[ \frac{L - D_i}{T_j} \right] \right\} \]

\[ \chi_{\min} = \forall \tau_j \in \tau, \tau_i \in M_2(i, j) \left\{ t - \sum_{\forall \tau_k \in \tau \cap \tau_i} DBF(\tau_k, t) \right\} \quad (22) \]

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, $\chi_{\min}$ is not optimistic. At the same time the DBF based analysis demonstrates a concrete scheduling scenario. Thus, $\chi_{\min}$ is clearly not pessimistic, as the derived value by the PDBF approach can actually occur. Since the derived sleep interval $\chi_{\min}$ is at the same time neither pessimistic nor optimistic, it is safe and optimal, thus the theorem follows. \[ \square \]

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 $\tau$, 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. \[ \square \]
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 $O(C^b)$. The PDBF approach has an offline complexity of $O(\ell h)$, where $h = \sum_{t \in \mathbb{R}} L^b_i$ 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 $C^b_i$ and $\Gamma_i$ are used to generate wide variety of different tasks and their subsequent varying jobs. Suppose, $C^b_i$ and $\Gamma_i$ are the helper variables to provide the bounds on the best-case execution time (BCET) and sporadic delay of a task respectively. Then $C^b_i$ and $\Gamma_i$ are randomly selected for the given tasks in interval $C_i, [C^b_i, 1]$ and $T_i, [\Gamma_i, 1]$ respectively. Similarly, the actual execution time and sporadic delay of the individual jobs are randomly selected from the following intervals $[C^b_i, C_i]$ and $[T_i, T_i + \Gamma_i]$ respectively. The periods of the task-set are chosen from an interval, $T_{\text{min}}[1, \text{PUB}]$, where $T_{\text{min}}$ is 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.

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
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 PDBF and SRA. 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 $C^k$ as shown in Figure 7. Mostly, the gain occurs due to an increase in system utilisation, while the variation in $C^k$ 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 $\Gamma$ is also observed in the experiments against different values of $U$. The effect of $\Gamma$ 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 $\tau$, having a procrastination interval $Z$, smaller than that of a high priority task will have its $Z$ scaled down due to Equation 3. If $Z_i < \chi_i$, then not only the difference exists at level $\tau_i$ 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


