Fast and Effective Multiframe-Task Parameter Assignment Via Concave Approximations of Demand

Bo Peng
Department of Computer Science, Wayne State University, Detroit, MI, USA
et7889@wayne.edu

Nathan Fisher
Department of Computer Science, Wayne State University, Detroit, MI, USA
fishern@wayne.edu

Thidapat Chantem
Department of Electrical and Computer Engineering, Virginia Tech, Arlington, VA, USA
tchantem@vt.edu

Abstract

Task parameters in traditional models, e.g., the generalized multiframe (GMF) model, are fixed after task specification time. When tasks whose parameters can be assigned within a range, such as the frame parameters in self-suspending tasks and end-to-end tasks, the optimal offline assignment towards schedulability of such parameters becomes important. The GMF-PA (GMF with parameter adaptation) model proposed in recent work allows frame parameters to be flexibly chosen (offline) in arbitrary-deadline systems. Based on the GMF-PA model, a mixed-integer linear programming (MILP)-based schedulability test was previously given under EDF scheduling for a given assignment of frame parameters in uniprocessor systems. Due to the NP-hardness of the MILP, we present a pseudo-polynomial linear programming (LP)-based heuristic algorithm guided by a concave approximation algorithm to achieve a feasible parameter assignment at a fraction of the time overhead of the MILP-based approach. The concave programming approximation algorithm closely approximates the MILP algorithm, and we prove its speed-up factor is $(1+\delta)^2$ where $\delta > 0$ can be arbitrarily small, with respect to the exact schedulability test of GMF-PA tasks under EDF. Extensive experiments involving self-suspending tasks (an application of the GMF-PA model) reveal that the schedulability ratio is significantly improved compared to other previously proposed polynomial-time approaches in medium and moderately highly loaded systems.

2012 ACM Subject Classification Computer systems organization → Real-time systems

Keywords and phrases generalized multiframe task model (GMF), generalized multiframe task model with parameter adaptation (GMF-PA), self-suspending tasks, uniprocessor scheduling, mixed-integer linear programming, concave approximation, linear programming

Digital Object Identifier 10.4230/LIPIcs.ECRTS.2019.20

Acknowledgements We are grateful to the anonymous reviewers whose comments helped to significantly improve our paper. This research has been supported in part by the US National Science Foundation (Grant Nos. CNS-1618185, IIS-1724227, and CSR-1618979).

1 Introduction

A generalized multiframe (GMF) task, whose model [3] generalizes the multiframe task model [16] (MF) and the sporadic task model, consists of a number of ordered frames where each frame has its own execution time, relative deadline, and frame separation time (the minimum interval between two frames' release times). The GMF model generalizes the sporadic model by using a set of ordered frames to represent an instance of a sporadic task.

Instead of setting an identical implicit frame deadline and minimal separation time for each frame as in the MF model, the GMF model assigns each frame an individual deadline and a minimum frame separation time.

The multiframe models (GMF/MF) have many applications. For example, Andersson [1] presented the schedulability analysis of flows in multi-hop networks comprising software-implemented Ethernet switches according to the GMF model. Ding et al. [11] scheduled a set of tasks with an I/O blocking property under the MF model. The self-suspension tasks [18] can be represented using the GMF model but the problem size can be very large, e.g., in automotive systems. In the keynote [7] of ECRTS 2012, Buttle has shown many scheduling challenges as the number of ECUs in vehicles increases rapidly each year; there are more than 100 ECUs nowadays and each task can easily have 50-300 functions. In such complex systems, there are several self-suspension tasks (each consisting of multiple functions) and their end-to-end latencies need to be maintained in distributed settings.

The GMF model increases flexibility compared to the sporadic and MF task models, and all parameters in the GMF model are typically not mutable after task specification time. However, frame parameters can be adjustable (under the constraints of task parameters) to improve schedulability in applications such as the self-suspension tasks [20] and end-to-end flows [19]. Frame parameters are mainly used to maintain execution order in such applications (e.g., frame priorities in FP scheduling and frame deadlines in EDF scheduling [22]). In order to optimally assign parameters to improve schedulability, Peng and Fisher [18] extended the GMF model and presented the GMF with parameter adaptation model (GMF-PA). In the GMF-PA model, frame deadlines and separations can be selected under a set of constraints. In this flexible model, frame parameters are optimally assigned (towards schedulability) offline for each frame under the MILP algorithms [18].

Although the GMF-PA model is more flexible, it has been shown that both the feasibility and the parameter selection problems are very hard to solve. On the feasibility side, Ekberg and Yi [12] proved that the feasibility of sporadic task systems remains coNP-complete under bounded utilization. On the parameter selection side, the priority assignment of subtasks in end-to-end task systems (originally the classical job-shop scheduling algorithm) has been shown to be NP-hard [13]. The scheduling of self-suspending tasks (even for self-suspending tasks with at most two frames) is NP-hard in the strong sense [21].

In order to address the feasibility test and parameter selection problem, Peng and Fisher [18] gave an exact schedulability test of GMF-PA tasks when frame parameters are integers. The test is based on mixed-integer linear programming (MILP) under EDF scheduling in uniprocessor systems. A sufficient, MILP-based schedulability test was also developed. Although this sufficient approximation algorithm [18] is quite efficient, it is still MILP-based and thus may require exponential-time to solve in general. The goal and contribution of this paper are to give an efficient linear programming-based algorithm that can determine the feasibility and select the frame parameters of GMF-PA tasks.

The MILP-based algorithm contains a set of integer variables which form a set of staircase functions/constraints (detailed in Section 5). To transform the MILP-based algorithm into a LP-based algorithm, our idea is to use a set of linear functions to approximate all staircase functions. As such, the selection of the slope values of the linear functions is directly related to the schedulability of a system; if the slope values are not properly set, the linear functions can grossly over-approximate, resulting in low schedulability ratio (the number of successfully scheduled systems over the total tested).

In order to get a close approximation, we first use a set of concave functions that very closely tracks the demand staircase functions to incur only a very small speed-up factor compared to the MILP algorithm. Since there exist no known efficient methods to solve concave programming problems, we use the concave functions to guide the slope assignment of linear functions in our iterative LP-based algorithm. That is, the LP algorithm runs multiple times during which the algorithm adjusts the slopes of the linear functions based on the concave functions. According to experiments, after a small number of iterations, the LP-based algorithm can approach (or reach) the local optimal[1]. We apply the LP-based algorithms to schedule self-suspending tasks under EDF scheduling in uniprocessor systems as a test case.

Our Contributions:

- We give a concave approximation algorithm based on the MILP algorithm and prove the speed-up factor of the algorithm is $(1+\delta)^2$ with respect to the exact schedulability test of GMF-PA tasks under EDF scheduling on uniprocessors. The positive constant δ is a user-defined constant which can be made arbitrarily close to zero.
- Since there is no known tractable way to solve a concave programming problem, we develop a LP-based heuristic algorithm based on the concave approximation algorithm for GMF-PA tasks. The LP-based algorithm is an efficient schedulability test and can select frame parameters at the same time.
- We apply the LP-based algorithm to schedule multiple-suspending tasks. To exploit the unique property of one-suspending tasks, as opposed to multi-suspending tasks, we present an improved heuristic algorithm for GMF-PA tasks.
- We conduct extensive experiments and show that the LP-based algorithms with fixed numbers of iterations outperform previous work in terms of schedulability and average running time. The fixed numbers of iterations make the LP-based algorithms pseudo-polynomial (the input size depends on the maximum interval length [3]), which is more efficient than the MILP-based approach.

Section 2 surveys the related work. We review our GMF-PA model in Section 3, and we formally state the goal of this paper in Section 4. Section 5 reviews our parameter-adaptation method using mixed-integer linear programming (MILP) to obtain a schedulability test under EDF scheduling. The concave approximation algorithm based on the MILP algorithm is presented in Section 6. Since concave programming algorithm does not scale well, two iterative LP-based algorithms are presented in Section 7. After applying the LP-based algorithms to self-suspending tasks, Section 8 provides extensive experimental results compared to state-of-the-art results. At last, Section 9 concludes this work and proposes future work.

2 Related Work

In this section, we introduce the related work on the GMF-PA model in Section 2.1, and survey one of its applications, the self-suspending tasks, in Section 2.2.

2.1 The Generalized Multiframe Model

The generalized multiframe model (GMF) was presented by Baruah et al.[3] to extend the sporadic task model and multiframe task model (MF) [16]. The recurring real-time task (RRT) model [2] generalizes the GMF model to handle conditional code. The digraph model [23] further generalizes the RRT model to allow arbitrary directed graphs (with loops),

[1] The local optimal of the iterative LP-based algorithm is reached when all variables converge.

and it was shown that the feasibility problem on preemptive uni-processor systems remains tractable (pseudo-polynomial complexity with bounded system utilization). A complete review is surveyed by Stigge and Yi [24].

The GMF model has great advantages and has been applied to multiple areas, as described earlier. However, current related models typically assume that parameters are fixed during task specification time. In the GMF-PA model [18] which extends the GMF model, frame parameters are flexible and can be chosen by the MILP-based approach in uniprocessor systems. The dGMF-PA model [19] extends the GMF-PA model to represent end-to-end flows in distributed systems. Similar flexible models, such as the parameter-adaptation model [8] and elastic model [6], are also used in many applications.

2.2 The Self-Suspending Task Model

A typical self-suspension task model [15] contains two computational frames separated by a self-suspending frame. After the first computational frame finishes, the job suspends executing the other computational frame until an external operation completes. The order of the frames is required and a task suspends itself to communicate with external devices, I/O operations, computation offloading, etc. We call such tasks *one-suspension* self-suspending tasks.

For one-suspension self-suspending tasks, Ridouard et al. [21] proved that scheduling such periodic self-suspending tasks on a uniprocessor is NP-hard in the strong sense. Due to the hardness of such scheduling problems, Chen and Liu [9] gave a fixed-relative-deadline (FRD) scheduling algorithm to improve the schedulability of sporadic self-suspending tasks on uniprocessor systems. The FRD algorithm assigns frame relative deadlines and schedules the ordering of frames of tasks under EDF scheduling.

The multiple-segment suspending task model [14], which allows multiple suspending frames, explicitly considers the execution sequence of frames in a task. Peng and Fisher [18] utilize MILP to select frame parameters of multiple-segment self-suspending tasks. The MILP algorithm [18] is an optimal FRD algorithm which extends the work by Chen and Liu [9]. A recent review on scheduling self-suspending tasks (mostly on one-suspension tasks) can be found in the work by Chen et al. [10].

3 Model

We review the generalized multiframe (GMF) model [3] and the generalized multiframe model with parameter adaptation (GMF-PA) in this section.

A GMF task τ_i consists of a set of ordered frames and each frame ϕ_i^j has its own execution time E_i^j, relative deadline D_i^j, and frame separation time P_i^j. All frames of a task τ_i can be represented by the 3-vector tuple $(\overrightarrow{E_i}, \overrightarrow{D_i}, \overrightarrow{P_i})$ where $\overrightarrow{E_i}$=$[E_i^0, E_i^1,..., E_i^{N_i-1}]$, $\overrightarrow{D_i}$=$[D_i^0, D_i^1,..., D_i^{N_i-1}]$, and $\overrightarrow{P_i}$=$[P_i^0, P_i^1,..., P_i^{N_i-1}]$. The ℓ'th frame of task τ_i arrives at time a_i^ℓ, has deadline at $a_i^\ell + d_i^\ell$, and worst-case execution time e_i^ℓ. Since frames arrive in sequence, the ℓ'th frame corresponds to frame $\phi_i^{\ell \bmod N_i}$, and we have:

1. $a_i^{\ell+1} \geq a_i^\ell + P_i^{\ell \bmod N_i}$
2. $d_i^\ell = D_i^{\ell \bmod N_i}$
3. $e_i^\ell = E_i^{\ell \bmod N_i}$

Based on the GMF model, the GMF-PA model [18] is derived to allow frame parameters to be assigned instead of fixing them during task specification time. Let $\mathcal{T} = \{\tau_0, \tau_2,..., \tau_{n-1}\}$ be the task system of n GMF-PA tasks executing on one processor. The task τ_i=$[\phi_i^0, \phi_i^1, \phi_i^2,..., \phi_i^{N_i-1}]$ consists of N_i frames where ϕ_i^j=$(E_i^j, \underline{D}_i^j, \overline{D}_i^j, \underline{P}_i^j, \overline{P}_i^j)$. The j'th frame execution time of the i'th task is E_i^j, and the i'th task-wise execution time is $E_i = \sum_{j=0}^{N_i-1} E_i^j$. The lower bound of relative deadline D_i^j (respectively, the minimum inter-arrival time between consecutive frames, P_i^j) is $\underline{D}_i^j$ (respectively, $\underline{P}_i^j$) and the upper bound of D_i^j (respectively, P_i^j) is $\overline{D}_i^j$ (respectively, $\overline{P}_i^j$). The frame parameters D_i^j and P_i^j can be flexibly assigned in the ranges $[\underline{D}_i^j, \overline{D}_i^j]$ and $[\underline{P}_i^j, \overline{P}_i^j]$, respectively. The frame distance $D_i^{j,k} = D_i^k + \sum_{p=0}^{(k-j-1) \bmod N_i} P_i^{(j+p) \bmod N_i}$ represents the relative time between the release of the j'th frame and the deadline D_i^k of the k'th frame. For example, $D_i^{2,4} = P_i^2 + P_i^3 + D_i^4$. The task deadline $\mathcal{D}_i$ is the upper bound of $D_i^{N_i-1} + \sum_{j=0}^{N_i-2} P_i^j$, and the task minimum inter-arrival time $\mathcal{P}_i$ is the upper bound of $\sum_{j=0}^{N_i-1} P_i^j$. The utilization of task τ_i is $U_i = E_i/\mathcal{P}_i$, and the utilization of a task system is $U_{cap} = \sum_{i=0}^{n-1} U_i$.

Frame parameters (D_i^j and P_i^j) must satisfy the localized Monotonic Absolute Deadlines (l-MAD) property [3] to maintain frame execution order. That is, the absolute deadline of the j'th frame must be no later than the one of the $j+1$'th frame ($D_i^j \leq P_i^j + D_i^{(j+1) \bmod N_i}$, $\forall i, j$). Figure 1 shows an example of the GMF model with the l-MAD property. The l-MAD property is widely used in systems which use first-in first-out (FIFO) scheduling for a shared resource. E.g., a network can be seen as a shared resource and packets sent from a computational node to a network node follow FIFO scheduling.

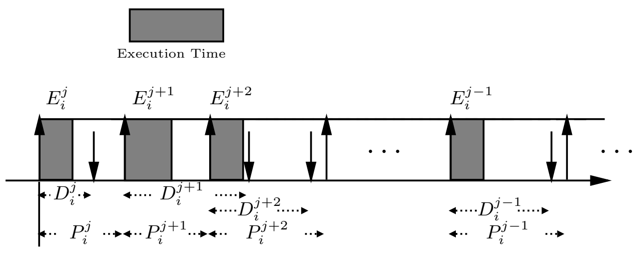


Figure 1 This figure contains all task τ_i's ordered frames from the j'th frame to the $(j-1)$ mod N_i'th frame (we omit "mod N_i" in this figure for simplicity). The starting frame can be any frame ϕ_i^j in an interval length t. Note that each frame deadline can be larger than frame separation time, e.g., $D_i^{j+1} \geq P_i^{j+1}$ in this figure. The details will be shown in Section 5.

4 Problem Statement

Let $\mathsf{dbf}_i(t, \vec{F_i})$ be the *task* demand bound function of a GMF-PA task τ_i within the interval length t. Let $\vec{F_i} = [D_i^0, P_i^0, D_i^1, P_i^1, \ldots, D_i^{N_i-1}, P_i^{N_i-1}]$ represent an assignment of values for all the task parameters (frame deadline and separations) of task τ_i. The task demand bound function $\mathsf{dbf}_i(t, \vec{F_i})$ accounts for task τ_i's accumulated execution time of frames which have both release times and deadlines inside the interval of length t. We use the notation $\mathsf{dbf}_i(t, D_i^{j,k})$ to represent the demand for the k'th frame when the first frame to arrive in the interval length t is the j'th frame. The relationship between the frame demand and task demand will be presented in Section 5. In a uniprocessor system $\mathcal{T}$, the sufficient and necessary condition for schedulability of a task set $\mathcal{T}$ is shown in Equation 1.

$$\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i) \leq t, \quad \forall t. \tag{1}$$

▶ **Problem Definition.** *Given the above model, our goal is to find an optimal and valid assignment $\vec{F}_i$ of frame parameters of all tasks so that the worst-case demand $\sum_{\tau_i \in \mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i)$ over all time intervals of length t is minimized.*

5 The MILP Algorithm

We review the MILP algorithm [18] to solve the problem defined in Section 4 under EDF scheduling in uniprocessor systems since the proposed concave programming and LP-based algorithms are closely related to the MILP algorithm.

Figure 2 shows the MILP algorithm. Notations in bold font are constants and the other notations are variables. Lines 3 and 5 are the requirements that a feasible system must obey. Line 4 shows the l-MAD property. Line 6 shows the calculation of the demand for every possible sequence of frames of task τ_i over any interval of length t. To calculate all possible frame demands, we use the notation[2] $y_i^{j,k}(t)$ to denote the demand of the k'th frame of task τ_i starting from the j'th frame over a t-length interval. To calculate the worst-case demand under EDF scheduling, the starting j'th frame arrives exactly at the start of the interval and subsequent frames arrive as soon as possible (e.g., see [3] for GMF schedulability).

The inequality $\frac{t-t_b}{\mathcal{P}_i} \leq x_i^{j,k}(t) - \frac{realmin}{\mathcal{P}_i}$ is the constraint function that decides the value of $x_i^{j,k}(t)$ where $x_i^{j,k}(t)$ decides the value of $y_i^{j,k}(t)$ in turn. The length t_b is the summation of the previous periods $\lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i$ and the frame distance from the starting j'th frame to k'th frame $D_i^{j,k}$, and the constant $realmin$ is the smallest representable positive number for the MILP solver. For example, the length $t_b = D_i^{1,3} + \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i$ if we consider the interval starting with an arrival of the first frame and ending at the deadline of the third frame. When $t \geq t_b$, the integer variable $x_i^{j,k}(t) \in [0, 1]$ must be one for the inequality in Line 6 to be feasible and the demand E_i^k contributes to $y_i^{j,k}(t)$. When $t < t_b$, $x_i^{j,k}(t)$ can be either zero or one. However, the MILP tends to choose zero for $x_i^{j,k}(t)$ to obtain a smaller demand (shown in Lemma 1). We calculate demand $y_i^{j,k}(t)$ for all possible combinations of i, j, k, and t in Line 6. For simplicity, we use "$\forall$" to represent the ranges of variables. The task index i ranges from zero to $n-1$. The superscripts j and k range from zero to $N_i - 1$. The maximum integer interval length [3] $H = \lceil \frac{U_{cap}}{1-U_{cap}} \cdot \max_{\tau_i \in \mathcal{T}}(\mathcal{P}_i) \rceil$.

▶ **Lemma 1** (from [18]). *The value of $y_i^{j,k}(t)$ in the MILP is the exact worst-case demand of frame ϕ_i^k over a t-length interval when the first frame to arrive in the interval is ϕ_i^j (with respect to the frame parameters assigned to each frame of τ_i by the MILP).*

Line 7 calculates task τ_i's demand $y_i^j(t)$ whose starting frame in the t-length interval is the j'th frame. In Line 8, the demand $y_i(t)$ is the maximum demand for τ_i over all $y_i^j(t)$. At last, the demand of all tasks $\sum_{i=0}^{n-1} y_i(t)$ is set to be no larger than $\mathcal{L} \cdot t$ as shown in Equation 1.

[2] The term $\mathsf{dbf}_i(t, D_i^{j,k})$ represents the frame demand, and the term $y_i^{j,k}(t)$ is a free variable in the mathematical programming formulation that is used to calculate the demand $\mathsf{dbf}_i(t, D_i^{j,k})$.

Parameter Selection and Exact Feasiblity Test.

```
1   minimize: $\mathcal{L}$
2   subject to:
3       $\boldsymbol{E_i^k} \leq \underline{\boldsymbol{D_i^k}} \leq D_i^k \leq \overline{\boldsymbol{D_i^k}}, \quad \forall i, k.$
        $\boldsymbol{E_i^k} \leq \underline{\boldsymbol{P_i^k}} \leq P_i^k \leq \overline{\boldsymbol{P_i^k}}, \quad \forall i, k.$
4       $D_i^k \leq P_i^k + D_i^{(k+1) \mod N_i}, \quad \forall i, k.$
5       $\sum_{k=0}^{N_i-1} P_i^k \leq \boldsymbol{\mathcal{P}_i}, D_i^{N_i-1} + \sum_{j=0}^{N_i-2} P_i^j \leq \boldsymbol{\mathcal{D}_i}, \quad \forall i.$
6       $\boxed{\begin{array}{l} y_i^{j,k}(t) = x_i^{j,k}(t) \cdot \boldsymbol{E_i^k} + \lfloor \frac{\boldsymbol{t}}{\boldsymbol{\mathcal{P}_i}} \rfloor \cdot \boldsymbol{E_i^k}, \quad \forall i, j, k, t. \\ \frac{\boldsymbol{t} - t_b}{\boldsymbol{\mathcal{P}_i}} \leq x_i^{j,k}(t) - \frac{\boldsymbol{realmin}}{\boldsymbol{\mathcal{P}_i}}, \quad \forall i, j, k, t. \\ t_b = D_i^{j,k} + \lfloor \frac{\boldsymbol{t}}{\boldsymbol{\mathcal{P}_i}} \rfloor \cdot \boldsymbol{\mathcal{P}_i} \end{array}}$
7       $y_i^j(t) = \sum_{k=0}^{N_i-1} y_i^{j,k}(t), \quad \forall i, j, t.$
8       $y_i(t) \geq y_i^j(t), \quad \forall i, j.$
9       $\sum_{i=0}^{n-1} y_i(t) \leq \mathcal{L} \cdot \boldsymbol{t} \quad \forall t.$
10  and: $D_i^k, P_i^k, y_i^{j,k}(t), y_i(t), \mathcal{L} \in \mathbb{R}^*, \boxed{x_i^{j,k}(t) \in \{0, 1\}}.$
```

Figure 2 This figure shows the MILP algorithm [18]. In the concave programming and LP-based algorithms (shown in Sections 6 and 7), we only change the frame demand in Line 6 and remove all integer variables $x_i^{j,k}(t)$.

If the system is schedulable, $\mathcal{L} \leq 1$. We minimize $\mathcal{L}$ in the MILP which also minimizes the summation of all task demand over all interval lengths[3] t. The MILP algorithm's necessity and sufficiency for feasibility are proved in Theorem 2.

▶ **Theorem 2** (from [18]). *For arbitrary, real-valued parameters, our MILP is a necessary feasibility test when $\mathcal{L} \leq 1$. When frame parameters are restricted to be integers (i.e., D_i^k, $P_i^k \in \mathbb{N}\ \forall\ i,\ k$), then the MILP is an exact feasibility test when $\mathcal{L} \leq 1$.*

6 The Concave Approximation Algorithm

We reviewed our previously proposed MILP in the last section. In this section, we give a concave approximation algorithm for the MILP algorithm and prove the speed-up factor of the concave approximation algorithm (compared to the optimal FRD/the MILP algorithm) can approach one. Although there is no known efficient way to solve a concave programming problem, our concave approximation algorithm plays a key role in the LP-based algorithms presented in the next section.

6.1 The Concave Functions

We first use the concave function (Equation 2) (illustrated by the blue dashed curve of Figure 3) to approximate the exact frame demand determined by the MILP in Line 6 of Figure 2.

[3] We take integer-valued t since we cannot check all real-valued t. We also use integer constants t in the concave programming and LP-based algorithms later.

$$\boxed{\mathsf{dbf}_i^{\text{concave}}(t, D_i^{j,k}) = \max\left\{0, E_i^k \cdot (1+\delta) - E_i^k \cdot \delta \cdot e^{\mu\cdot(D_i^{j,k} + \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i - t)}\right\} + \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot E_i^k} \quad (2)$$

The concave programming algorithm is constructed by replacing all staircase functions in Line 6 of Figure 2 with $y_i^{j,k}(t) = \mathsf{dbf}_i^{\text{concave}}(t, D_i^{j,k})$ and removing all integer variables. The other lines in Figure 2 remain the same.

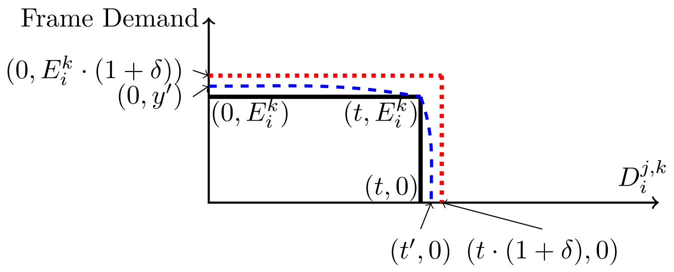


Figure 3 This example shows the frame demand within interval length $t < \mathcal{P}_i$. The blue dashed curve is a concave function and the staircase function in black solid line represents the exact frame demand in the MILP. The red dotted staircase line with error rates δ on both axes represents an upper bound on the concave function.

Equation 2 shows our proposed concave approximation function $\mathsf{dbf}_i^{\text{concave}}(t, D_i^{j,k})$ (e.g., the blue dashed curve in Figure 3) for the k'th frame demand of task τ_i during the t-length interval in which the starting frame is the j'th frame. We define the system-wide maximum error rate[4] δ. The rate δ must be larger than zero to ensure the demand of any approximation function be larger than the staircase function for any given deadline. We set δ as a designer-defined constant in the system, and set the constant $\mu = \frac{1}{\delta} \cdot \ln\left(1 + \frac{1}{\delta}\right)$ as shown in Lemma 3. In Lemma 3, we prove that the maximum error rate of the concave function is smaller than the system maximum error rate δ, and the concave function approaches the staircase function when δ decreases.

▶ **Lemma 3.** *The demand of the concave function in Equation 2 over-approximates the one in the MILP algorithm, and the error rate of the concave function is smaller than the system error constant δ when we set μ in Equation 2 as follows,*

$$\mu = \frac{1}{\delta} \cdot \ln\left(1 + \frac{1}{\delta}\right). \quad (3)$$

Proof. Let δ_y and δ_d be the worst-case error rates on the demand (on y-axis) and deadline (on x-axis) directions of concave functions, respectively. Let $t_b = D_i^{j,k} + \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i - t$. The worst rates happen when, in Figure 3 for example, $E_i^k \cdot (1 + \delta_y) = y'$ and $t \cdot (1 + \delta_d) = t'$. We will prove that $\delta \geq \delta_y$ and $\delta \geq \delta_d$.

When $0 \leq t_b \leq t$, the largest demand of the concave function happens at $t_b = 0$. By substituting $E_i^k \cdot (1 + \delta_y)$ (respectively, 0) for $y_i^{j,k}(t)$ (respectively, t_b), the concave function becomes $E_i^k \cdot (1 + \delta_y) = E_i^k \cdot (1 + \delta) - E_i^k \cdot \delta \cdot e^{\mu\cdot(0-t)}$. After simplification, we get

[4] The error rate (with respect to the exact frame demand function) of an approximation function is its percentage increase in the y-axis direction for $t \leq D_i^{j,k}$ or its percentage increase in the x-axis dimension if $t > D_i^{j,k}$. The maximum error rate is the largest error rate over all $t > 0$. E.g., the error rate on the x-axis of the point $(t \cdot (1 + \delta), 0)$ in Figure 3 is δ. The maximum error rate of any approximation function must be smaller than the system-wide maximum error rate δ.

$\delta_y = \delta - \delta \cdot e^{\mu\cdot(-t)}$. Thus, $\delta \geq \delta_y$ and δ is an upper bound of δ_y. Since the concave function is a decreasing function and it passes the points $(0, E_i^k \cdot (1+\delta_y))$ and (t, E_i^k), the concave function over-approximates the corresponding demand in MILP when $0 \leq t_b \leq t$.

When $t_b > t$, the maximum error on the deadline direction happens at $t_b = t \cdot (1+\delta_d)$. By substituting 0 (respectively, $t \cdot (1+\delta_d)$) for $y_i^{j,k}(t)$ (respectively, t_b), we have $0 = E_i^k \cdot (1+\delta) - E_i^k \cdot \delta \cdot e^{\mu\cdot(t\cdot(1+\delta_d)-t)}$. After simplification, we have $\delta_d = \frac{1}{t\cdot\mu} \cdot \ln(1+\frac{1}{\delta})$. We set $\mu = \frac{1}{\delta} \cdot \ln(1+\frac{1}{\delta})$, and $\delta_d = \frac{\delta}{t}$ after replacing μ in δ_d. Since $t \geq 1$, $\delta \geq \delta_d$. ◀

6.2 Speed-Up Factor Analysis

A speedup factor is a value that quantifies the quality of an approximation algorithm with respect to the optimal scheduling algorithm. A speedup factor $\mathcal{S} > 1$ [4] means that an approximation algorithm can schedule a task system at a speed-$\mathcal{S}$ processor if an optimal algorithm can schedule the system at a speed-one processor.

Let $\mathcal{L}_{MILP}$ be the value of the objective function returned by the MILP algorithm and $\mathcal{L}_{concave}$ be the value returned by the concave programming algorithm. We will prove that $\mathcal{L}_{MILP} < \mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$. $\mathcal{L}_{MILP} < \mathcal{L}_{concave}$ indicates that a task system will be deemed schedulable by the MILP algorithm if the system is schedulable by the concave programming algorithm (which means $\mathcal{L}_{MILP} < \mathcal{L}_{concave} \leq 1$). By the definition of the speed-up factor, $\mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$ indicates that the speed-up factor of our concave programming algorithm is $(1+\delta)^2$ with respect to the MILP algorithm. In other words, $\mathcal{L}_{concave}/(1+\delta)^2 < \mathcal{L}_{MILP}$ indicates a task system can be scheduled by the concave programming algorithm under a $(1+\delta)^2$-speed processor if the system can be scheduled by the MILP algorithm under the corresponding one-speed processor.

We prove $\mathcal{L}_{MILP} < \mathcal{L}_{concave}$ in Lemma 4, and $\mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$ from Lemma 5 to Lemma 8. By Lemmas 4 and 8, we prove that the speed-up factor of our concave programming algorithm is $(1+\delta)^2$ with respect to the MILP algorithm in Theorem 9.

▶ **Lemma 4.** *Let $\mathcal{L}_{MILP}$ and $\mathcal{L}_{concave}$ be the values returned by the MILP and concave programming algorithms (assume they exist), respectively. We have:*

$$\mathcal{L}_{MILP} < \mathcal{L}_{concave}. \tag{4}$$

Proof. Let $\mathcal{L}'_{MILP}$ be the value calculated as follows. Assume there exists such a solver that can solve the concave programming algorithm and return $\mathcal{L}_{concave}$, frame deadlines and separations. We assign the returned frame parameters from the concave programming algorithm to the formulation of the MILP algorithm and get the value of $\mathcal{L}'_{MILP}$.

Under the same values of frame parameters, any frame demand of concave programming algorithm is larger than its corresponding demand of the MILP algorithm, as shown in Lemma 3. The task demands of concave programming algorithm with the preassigned frame parameters are thus also larger than the ones from the MILP approach. When we summarize task demands over any interval length, $\mathcal{L}'_{MILP}$ is thus always less than $\mathcal{L}_{concave}$. Since $\mathcal{L}'_{MILP}$ is calculated by preassigned frame parameters, $\mathcal{L}'_{MILP}$ must not be smaller than $\mathcal{L}_{MILP}$. If the frame parameters returned by the MILP and concave programming algorithms are all identical, $\mathcal{L}'_{MILP} = \mathcal{L}_{MILP}$. In all, $\mathcal{L}_{MILP} \leq \mathcal{L}'_{MILP} < \mathcal{L}_{concave}$ and this lemma is proved. ◀

In order to prove $\mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$, we first define $\mathcal{L}'_{concave}$. Let the MILP algorithm return $\mathcal{L}_{MILP}$, frame deadlines and separations. If we fix the deadline and separation variables of the concave programming formulation to be the values returned by the

MILP, we calculate the value of $\mathcal{L}^{'}_{concave}$. We will prove $\mathcal{L}_{concave} \le \mathcal{L}^{'}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$. $\mathcal{L}_{concave} \le \mathcal{L}^{'}_{concave}$ is proved in Lemma 5. Based on the demand bound functions defined in Equations 8 and 9, we prove $\mathcal{L}^{'}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$ in Lemma 8.

▶ **Lemma 5.** *Let $\mathcal{L}_{concave}$ be the optimal value returned by the concave programming algorithm, and $\mathcal{L}^{'}_{concave}$ be the value calculated by the concave programming algorithm using the frame parameters returned by the MILP. We have:*

$$\mathcal{L}_{concave} \le \mathcal{L}^{'}_{concave}. \tag{5}$$

Proof. Since the concave programming algorithm minimizes $\mathcal{L}_{concave}$, $\mathcal{L}_{concave}$ must be the smallest value over all feasible-assigned/preassigned frame parameters, and $\mathcal{L}_{concave} < \mathcal{L}^{'}_{concave}$. If frame parameters returned by the MILP and concave programming algorithms are same, $\mathcal{L}_{concave} = \mathcal{L}^{'}_{concave}$. In all, $\mathcal{L}_{concave} \le \mathcal{L}^{'}_{concave}$. ◀

For ease of proof, we consider a staircase approximation function $\mathsf{dbf}^a_i(t, D^{j,k}_i)$ illustrated by the red dotted line in Figure 3 for task τ_i over the t-length interval, and the solid line shows an example of the staircase demand $\mathsf{dbf}_i(t, D^{j,k}_i)$.

Equation 6 shows $\mathsf{dbf}_i(t, D^{j,k}_i)$ as the k'th frame-demand function of task τ_i over the t-length interval that starts with the j'th frame. The corresponding task demand $\mathsf{dbf}_i(t, \vec{F}_i)$ is shown in Equation 8, and the reasoning is same as to the relationship between $y_i(t)$ and $y^{j,k}_i(t)$ in the MILP algorithm. I.e., we take the maximum demand over all sequences as the task demand. The approximate frame-demand $\mathsf{dbf}^a_i(t, D^{j,k}_i)$ and task-demand $\mathsf{dbf}^a_i(t, \vec{F}_i)$ (for $\mathsf{dbf}_i(t, D^{j,k}_i)$ and $\mathsf{dbf}_i(t, \vec{F}_i)$, respectively) are defined in Equations 7 and 9, respectively. We prove that the approximation demand over-approximates the concave demand in Lemma 6.

$$\mathsf{dbf}_i(t, D^{j,k}_i) = \begin{cases} 0, & 0 \le t < D^{j,k}_i \\ E^k_i, & D^{j,k}_i \le t \le \mathcal{P}_i \\ E^k_i \cdot \lfloor \frac{t}{\mathcal{P}_i} \rfloor + \mathsf{dbf}_i(t - \mathcal{P}_i \cdot \lfloor \frac{t}{\mathcal{P}_i} \rfloor, D^{j,k}_i), & t > \mathcal{P}_i \end{cases} \tag{6}$$

$$\mathsf{dbf}^a_i(t, D^{j,k}_i) = \begin{cases} 0, & 0 \le t < \frac{D^{j,k}_i}{(1+\delta)} \\ (1+\delta) \cdot E^k_i, & \frac{D^{j,k}_i}{(1+\delta)} \le t \le \mathcal{P}_i \\ E^k_i \cdot \lfloor \frac{t}{\mathcal{P}_i} \rfloor + \mathsf{dbf}^a_i(t - \mathcal{P}_i \cdot \lfloor \frac{t}{\mathcal{P}_i} \rfloor, D^{j,k}_i), & t > \mathcal{P}_i \end{cases} \tag{7}$$

$$\mathsf{dbf}_i(t, \vec{F}_i) = \max_{j=0}^{N_i-1} \{ \sum_{k=0}^{N_i-1} \mathsf{dbf}_i(t, D^{j,k}_i) \} \tag{8}$$

$$\mathsf{dbf}^a_i(t, \vec{F}_i) = \max_{j=0}^{N_i-1} \{ \sum_{k=0}^{N_i-1} \mathsf{dbf}^a_i(t, D^{j,k}_i) \} \tag{9}$$

▶ **Lemma 6.** *The demand of task τ_i over any interval length t in Equation 9 is an upper bound of its corresponding concave approximation demand.*

Proof. In Lemma 3, we proved that $\delta_d \le \delta$. Let $t_\Delta = t - \mathcal{P}_i \cdot \lfloor \frac{t}{\mathcal{P}_i} \rfloor$. From Equation 2 and the definition of δ_d, the concave demand with any value assigned for $D^{j,k}_i \in [0, t_\Delta \cdot (1+\delta_d)]$ is smaller than $E^k_i \cdot (1+\delta)$, and the demand is zero when $D^{j,k}_i > t_\Delta \cdot (1+\delta_d)$. Since $\mathsf{dbf}^a_i(t, D^{j,k}_i) = E^k_i \cdot (1+\delta)$ when $D^{j,k}_i \le t_\Delta \cdot (1+\delta)$ and $\delta_d \le \delta$, the demand function $\mathsf{dbf}^a_i(t, D^{j,k}_i)$ over approximates the concave demand. For task-wise demand $\mathsf{dbf}^a_i(t, \vec{F}_i)$, we take the summation of all frame demand $\mathsf{dbf}^a_i(t, D^{j,k}_i)$ of task τ_i over all sequences (sequences differ from the starting j'th frame in the t-length interval), and take the maximum demand over all sequences as the task demand. The task demand $\mathsf{dbf}^a_i(t, \vec{F}_i)$ also over approximates the corresponding concave demand. In all, we have proved this lemma. ◀

With the demand bound functions shown in Equations 8-9, we prove $\mathcal{L}'_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$ in Lemmas 7-8.

▶ **Lemma 7.** *For the task τ_i's demand $\mathsf{dbf}_i(t, \vec{F}_i)$ and its approximation demand $\mathsf{dbf}^a_i(t, \vec{F}_i)$ in the t-length time interval, we have: $\mathsf{dbf}_i((1+\delta)\cdot t, \vec{F}_i)\cdot(1+\delta) \geq \mathsf{dbf}^a_i(t, \vec{F}_i)$.*

Proof. We first prove $\mathsf{dbf}_i((1+\delta)\cdot t, D_i^{j,k})\cdot(1+\delta) \geq \mathsf{dbf}^a_i(t, D_i^{j,k})$, and $\mathsf{dbf}_i((1+\delta)\cdot t, \vec{F}_i)\cdot(1+\delta) \geq \mathsf{dbf}^a_i(t, \vec{F}_i)$ can be extended by Equations 8 and 9. We classify all interval lengths t in three sets:

$T_1 : 0 \leq t < D_i^{j,k}/(1+\delta)$,
$T_2 : D_i^{j,k}/(1+\delta) \leq t \leq \mathcal{P}_i$,
T_3 : otherwise.

When $t \in T_1$, $\mathsf{dbf}_i(t, D_i^{j,k}) = \mathsf{dbf}^a_i(t, D_i^{j,k}) = 0$. Since demand bound functions are monotonically increasing functions, $\mathsf{dbf}_i((1+\delta)\cdot t, D_i^{j,k})\cdot(1+\delta) \geq \mathsf{dbf}_i(t, D_i^{j,k}) = \mathsf{dbf}^a_i(t, D_i^{j,k})$.

When $t \in T_2$, we know that $\mathsf{dbf}_i(t^*, D_i^{j,k}) = E_i^k$ at $D_i^{j,k} \leq t^* \leq \mathcal{P}_i$ from Equation 6. Let $t^* = t\cdot(1+\delta)$, we have $\mathsf{dbf}_i(t\cdot(1+\delta), D_i^{j,k}) = E_i^k$ at $D_i^{j,k}/(1+\delta) \leq t \leq \mathcal{P}_i$. From Equations 6 and 7, we know that $\mathsf{dbf}_i(t\cdot(1+\delta), D_i^{j,k})\cdot(1+\delta) = \mathsf{dbf}^a_i(t, D_i^{j,k})$ at $D_i^{j,k}/(1+\delta) \leq t \leq \mathcal{P}_i$.

When $t \in T_3$, it is trivial to see the fact that $\mathsf{dbf}_i((1+\delta)\cdot t, D_i^{j,k})\cdot(1+\delta) \geq \mathsf{dbf}^a_i(t, D_i^{j,k})$ since the demand is iteratively calculated from the demand when $t \in T_1 \cup T_2$. ◀

▶ **Lemma 8.** *Let $\mathcal{L}_{MILP}$ be the optimal value returned by the MILP algorithm, and $\mathcal{L}'_{concave}$ be the value calculated by the frame parameters returned by the MILP. We have:*

$$\mathcal{L}'_{concave} < \mathcal{L}_{MILP}\cdot(1+\delta)^2. \tag{10}$$

Proof. Line 9 of Figure 2 shows that $\mathcal{L}$ is the largest value of $\frac{\sum_{i=0}^{n-1} y_i(t)}{t}$ for all values of t in the MILP algorithm (can be derived from Lemma 1). We also require this line in the concave programming algorithm. From Lemma 7, we know that $\mathsf{dbf}_i((1+\delta)\cdot t, \vec{F}_i)\cdot(1+\delta) \geq \mathsf{dbf}^a_i(t, \vec{F}_i)$ for any task τ_i over any t-length interval. Let $t = (1+\delta)\cdot t^*$, we have:

$$\begin{aligned}
\mathcal{L}_{MILP} &= \max_{t>0} \frac{\sum_{\tau_i\in\mathcal{T}} \mathsf{dbf}_i(t, \vec{F}_i)}{t} \quad \text{By Lemma 1} \\
&= \frac{\sum_{\tau_i\in\mathcal{T}} \mathsf{dbf}_i((1+\delta)\cdot t^*, \vec{F}_i)}{(1+\delta)\cdot t^*} \\
&= \frac{\sum_{\tau_i\in\mathcal{T}} \mathsf{dbf}_i((1+\delta)\cdot t^*, \vec{F}_i)\cdot(1+\delta)}{(1+\delta)^2\cdot t^*} \\
&\geq \frac{\sum_{\tau_i\in\mathcal{T}} \mathsf{dbf}^a_i(t^*, \vec{F}_i)}{(1+\delta)^2\cdot t^*} \quad \text{By Lemma 7} \\
&\geq \frac{\mathcal{L}'_{concave}}{(1+\delta)^2} \quad \text{By Lemma 6}
\end{aligned} \tag{11}$$

◀

▶ **Theorem 9.** *When the concave programming algorithm returns integer frame deadlines and separation times, the speed-up factor of our concave programming algorithm with respect to the MILP algorithm is $(1+\delta)^2$.*

Proof. In Lemmas 4, 5, and 8, we have proved that $\mathcal{L}_{MILP} < \mathcal{L}_{concave} < \mathcal{L}_{MILP}\cdot(1+\delta)^2$. $\mathcal{L}_{MILP} < \mathcal{L}_{concave}$ indicates that a task system is deemed schedulable (with integer frame parameters) by the MILP if the task system is deemed schedulable (with integer parameters)

The LP-based Algorithm for GMF-PA tasks.

```
1  Initialize $D$ as $D_i^k \leftarrow (E_i^k/E_i) \cdot \mathcal{P}_i$, $\mathcal{L}^{\text{last}} \leftarrow \infty$, and $\mathcal{L}^{\text{cur}} \leftarrow \infty$
2  repeat
3           $\mathcal{L}^{\text{last}} \leftarrow \mathcal{L}^{\text{cur}}$
4           $S \leftarrow computeSlope(D)$
5           $[D, \mathcal{L}^{\text{cur}}] \leftarrow Heuristic\text{-}LP(D, S)$
6     until $\mathcal{L}^{\text{last}} - \mathcal{L}^{\text{cur}} < \epsilon$
7  Process frame deadlines D to integers.
8  $[\mathcal{L}^{\text{cur}}] \leftarrow Heuristic\text{-}LP\text{-}fixedDeadline(D, S)$
9  if $\mathcal{L}^{\text{cur}} \leq 1$
10    then return schedulable
11 else return unschedulable
```

Figure 4 The LP-based algorithm for GMF-PA tasks.

by the concave programming algorithm. $\mathcal{L}_{MILP} < \mathcal{L}_{concave}$ shows our concave programming algorithm is an approximation algorithm for the MILP.

We divide $(1+\delta)^2$ on both sides of the inequality $\mathcal{L}_{concave} < \mathcal{L}_{MILP} \cdot (1+\delta)^2$ to get $\mathcal{L}_{concave}/(1+\delta)^2 < \mathcal{L}_{MILP}$. $\mathcal{L}_{concave}/(1+\delta)^2$ represents that we change the speed of the processor from one to $(1+\delta)^2$. Thus, a task system must be scheduled by the concave programming algorithm with a $(1+\delta)^2$-speed processor if the task system is scheduled by the MILP on a single speed processor. From the definition of the speed-up factor, we have proved that the speed-up factor of our concave programming algorithm with respect to the MILP is $(1+\delta)^2$. ◀

7 The Linear Programming-Based Heuristic Algorithm and its Application to One-Suspension Self-Suspending Tasks

Until now, we have constructed the concave programming approximation algorithm for the MILP-based algorithm. Due to the difficulties in solving concave programming (or non-convex optimization) problems in general, we use a heuristic LP-based scheme to efficiently select frame parameters of GMF-PA tasks, and apply it to self-suspending tasks. For ease of presentation, we let $\underline{D}_i^k = E_i^k$, $\overline{D}_i^k = \mathcal{P}_i$, and $P_i^k = D_i^k$. In this case, frames deadlines are constrained by frame execution time and the l-MAD property. We present the LP-based heuristic algorithm in Section 7.1, and further to optimize the LP-based algorithm to schedule one-suspension self-suspending tasks in Section 7.2.

7.1 The Linear Programming-Based Heuristic Algorithm

The general routine of the LP-based scheme for GMF-PA tasks is: 1) We initialize frame parameters of GMF-PA tasks. 2) Given the frame parameters, we recalculate a set of linear functions, which approximate the staircase functions for frame demands in the MILP, guided by the concave programming algorithm. 3) We run the LP algorithm (shown later) based on the assigned linear functions, and receive frame parameters as outputs. If the difference in $\mathcal{L}$ values between the current and the last iterations is no smaller than some threshold, the program goes back to Step 2. 4) We round frame parameters to integers and run the LP algorithm with the fixed integer-valued parameters to get the final assignment.

$$\boxed{\mathsf{dbf}_i^{\text{linear}}(t, D_i^{j,k}) = \max\{0, s_i^{j,k}(t) \cdot (D_i^{j,k} - t') + E_i^k\} + \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot E_i^k, \quad t' = t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i} \tag{12}$$

computeSlope(D).

```
Calculate all $D_i^{j,k'}$ from $D$
$t' \leftarrow t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i$
$y_i^{j,k'}(t') \leftarrow E_i^k \cdot (1+\delta) - E_i^k \cdot \delta \cdot e^{\mu \cdot (D_i^{j,k'} - t')}$
if $D_i^{j,k'} > t'$
    then $s_i^{j,k}(t) \leftarrow (0 - E_i^k)/(\frac{1}{\mu} \cdot \ln(1+\frac{1}{\delta}) + t' - t')$
elseif $D_i^{j,k'} == t'$
    then $s_i^{j,k}(t) \leftarrow \frac{\partial}{\partial D_i^{j,k}} \left[ \mathsf{dbf}_i^{\text{concave}}(t', D_i^{j,k}) \right]$
else $s_i^{j,k}(t) \leftarrow (y_i^{j,k'}(t') - E_i^k)/(D_i^{j,k'} - t')$
return $S$
$\triangleright$ $S$ is the matrix that stores all slopes $s_i^{j,k}(t)$.
```

Figure 5 This algorithm calculates all slopes given all frame deadlines.

In *The LP-based Algorithm for GMF-PA Tasks* (Figure 4), we initialize frame deadlines by proportional deadline assignment (PDA [15]) to $D_i^k = (E_i^k/E_i) \cdot \mathcal{P}_i$. Given the deadline matrix D which stores all D_i^k, we calculate all slopes and store them in matrix S. We replace Line 6 of Figure 2 with Equation 12 to transform the algorithm into a LP algorithm $\textit{Heuristic-LP}(D, S)$ (Line 5 of Figure 4). The slope element $s_i^{j,k}(t)$ of S, which corresponds to $y_i^{j,k}(t)$, is calculated in the algorithm shown in Figure 5 and all lines pass the point (t', E_i^k). The linear functions are illustrated by the red lines in Figures 6-8. If the deadline $D_i^{j,k'}$ (generated from the previous iteration) is smaller than $t' = t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i$, we calculate the demand $y_i^{j,k'}(t')$ of the concave function at $D_i^{j,k'}$. The slope of the line is calculated by two points $(D_i^{j,k'}, y_i^{j,k'}(t'))$ and (t', E_i^k) illustrated in Figure 6. If the deadline $D_i^{j,k'}$ equals t', we calculate the slope by taking the tangent of the concave function at the point (t', E_i^k) shown in Figure 7. If the deadline is larger than t', we use two points (t', E_i^k) and $\left(\frac{1}{\mu} \cdot \ln(1+\frac{1}{\delta}) + t', 0\right)$, which is the cross point of the x-axis and the concave function, to calculate the slope, and the line with the slope is shown in Figure 8. The slope matrix S is adjusted in each iteration of the loop in Figure 4.

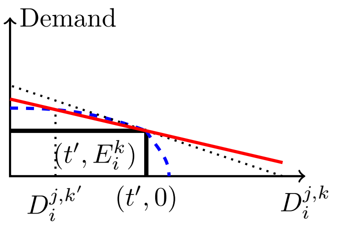


Figure 6 The frame deadline $D_i^{j,k'}$ of the last iteration is smaller than t' in this case.

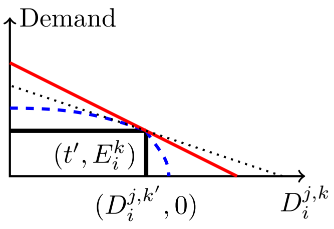


Figure 7 The frame deadline $D_i^{j,k'}$ of the last iteration equals t' in this case.

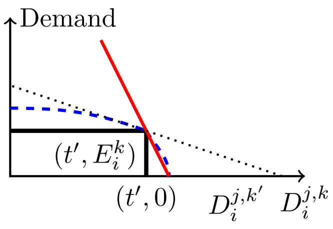


Figure 8 The frame deadline $D_i^{j,k'}$ of the last iteration is larger than t' in this case.

The loop in Figure 4 will not stop recursively calling function $\textit{Heuristic-LP}(D, S)$ until the difference of the $\mathcal{L}$ values in two consecutive iterations is smaller than the positive threshold ϵ. $\mathcal{L}^{\text{last}}$ and $\mathcal{L}^{\text{cur}}$ represent the $\mathcal{L}$ values of the last and current iterations, respectively. The $\textit{Heuristic-LP-fixedDeadline}(D, S)$ algorithm (Line 8 of Figure 4) uses the integer deadlines to maintain sufficiency for schedulability, which is proved in Theorem 11. We first round up frame deadlines to be integers. For each task, we keep reducing the largest frame deadline by one until the summation of them equals to its task deadline/period. We assign the deadline variables to the integer values in Line 7 of Figure 4 and the other parts are the same as in the *Heuristic-LP*(D, S) algorithm. The system is schedulable if $\mathcal{L} \leq 1$.

We prove in Theorem 10 that the `while` loop of the algorithm *The LP-based Algorithm for GMF-PA Tasks* function stops after a finite number of iterations. The sufficiency of the LP-based algorithm for schedulability is proved in Theorem 11.

▶ **Theorem 10.** *The* `while` *loop of the function* The LP-based Algorithm for GMF-PA Tasks *stops in a finite number of iterations.*

Proof. We first prove that $\mathcal{L}$ decreases from one iteration to the next. Before each iteration of the algorithm *Heuristic-LP*(D, S), we use the deadline assignment D' from the last iteration to calculate the slopes S of frame functions in the current iteration. Let $\mathcal{L}^{\text{last}}$ be the value of $\mathcal{L}$ in the last iteration. In the current iteration, let us assume that we use the same set of the deadlines D' to calculated the value $\mathcal{L}^{\text{cur}}$.

In the first and third cases shown in Figures 6 and 8, the frame demand is either smaller (if the last iteration is the first iteration) or equal to the one in the last iteration. In the second case, the frame demand is the same as the one in the last iteration. From all cases, we know that the same set of deadlines causes $\mathcal{L}^{\text{cur}} \leq \mathcal{L}^{\text{last}}$. Since we minimize $\mathcal{L}$ in the algorithm, the returned deadlines by the algorithm *Heuristic-LP*(D, S) must generate a value of $\mathcal{L}$ that is smaller than $\mathcal{L}^{\text{cur}}$. Thus, we have proved that $\mathcal{L}$ decreases from one iteration to the next. We also set a threshold to be the difference of the $\mathcal{L}$ values in two consecutive iterations, and we know that the lower bound of $\mathcal{L}$ equals $\sum_{i=1}^{n} U_i$. In either cases, the loop of the function *The LP-based Algorithm for GMF-PA Tasks* stops in a finite number of iterations. ◀

▶ **Theorem 11.** *The LP-based algorithm is a sufficient schedulability test when $\mathcal{L} \leq 1$.*

Proof. This proof is similar to Theorem 2. The sufficiency of any approximation/heuristic algorithm (w.r.t. the MILP algorithm) for schedulability requires two conditions: 1) the demand of the algorithm over any t-length interval is larger than the one in the MILP. 2) frame parameters must take integer values. The first condition ensures that the demand over approximates on any t, and the second condition ensures that the demand only changes at integer values. We require the second condition since all lengths (represented by t) can only be integers in the MILP algorithm. The LP-based algorithm over approximates system demand among all t, and the algorithm adjusts frame deadlines to be integers in the last iteration. ◀

7.2 The Application of the LP-Based Algorithm to One-Suspension Self-Suspending Tasks

The LP-based scheme can be applied to multiple-segment self-suspending tasks directly. In this section, we further optimize the algorithm for one-suspension self-suspending tasks by reducing the number of free variables and equations. Given that n is the number of tasks and H is the maximum interval length, the algorithm uses $8 \cdot n \cdot H + n$ fewer variables and $15 \cdot n \cdot H + n$ fewer number of constraints than the ones in the standard LP-based scheme.

For each task τ_i, we use variables D_i^1 and $\mathcal{P}_i - S_i - D_i^1$ (instead of D_i^1 and D_i^2) to denote frame deadlines to reduce the number of variables and constraints. S_i is the suspension length of task τ_i. In this case, the demand bound function only relies on D_i^1 and t and $\vec{F}_i = [D_i^1, D_i^1, \mathcal{P}_i - S_i - D_i^1, \mathcal{P}_i - S_i - D_i^1]$ since we let $P_i^k = D_i^k$. A task demand falls in four cases which are shown and proved in Theorem 12.

▶ **Theorem 12.** *The demand bound function of a task τ_i lies in one of the following four cases:*

$$\mathsf{dbf}_i(t,\vec{F}_i) = \begin{cases} \mathsf{dbf}_i^1(t,\vec{F}_i) & = \begin{cases} E_i^1, & 0 < D_i^1 \le t \\ 0, & t < D_i^1 < \mathcal{P}_i - S_i - t \\ E_i^2, & \mathcal{P}_i - S_i - t < D_i^1 \le \mathcal{P}_i - S_i \end{cases} \\ & \text{when } 0 < t < (\mathcal{P}_i - S_i)/2 \\ \mathsf{dbf}_i^2(t,\vec{F}_i) & = \begin{cases} E_i^1, & 0 < D_i^1 < \mathcal{P}_i - S_i - t \\ \max\{E_i^1, E_i^2\}, & \mathcal{P}_i - S_i - t \le D_i^1 \le t \\ E_i^2, & t < D_i^1 < \mathcal{P}_i - S_i \end{cases} \\ & \text{when } (\mathcal{P}_i - S_i)/2 \le t < \mathcal{P}_i - S_i \\ \mathsf{dbf}_i^3(t,\vec{F}_i) & = E_i^1 + E_i^2, \\ & \text{when } \mathcal{P}_i - S_i \le t \le \mathcal{P}_i \\ \mathsf{dbf}_i^4(t,\vec{F}_i) & = \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot (E_i^1 + E_i^2) + \mathsf{dbf}_i(t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i, D_i^1), \\ & \text{when } t > \mathcal{P}_i \end{cases} \tag{13}$$

Proof. Figures 9-10 show an example of the staircase demand of $\mathsf{dbf}_i^1(t,\vec{F}_i)$ and $\mathsf{dbf}_i^2(t,\vec{F}_i)$ with black solid lines, respectively. Roughly, the two staircase/concave demand curves head toward each other when t increases. The first two cases differ when the two staircase functions meet as t increases. The demand $\mathsf{dbf}_i^3(t,\vec{F}_i)$ considers the total task demand and $\mathsf{dbf}_i^4(t,\vec{F}_i)$ iterates over the first three cases.

For the demand $\mathsf{dbf}_i^1(t,\vec{F}_i)$ in the first case, when $0 < t < (\mathcal{P}_i - S_i)/2$, we know that $t < \mathcal{P}_i - S_i - t$ by simple mathematical transformation. In this case, we have two separate staircase functions as shown in Figure 9. When $D_i^1 \le t$, the demand of the first frame is E_i^1, the demand of the second frame is zero because $D_i^1 \le t < \mathcal{P}_i - S_i - t$. $D_i^1 < \mathcal{P}_i - S_i - t$ means $t < \mathcal{P}_i - S_i - D_i^1$ which indicates the deadline of the second frame is larger than t. Thus, $\mathsf{dbf}_i^1(t,\vec{F}_i) = E_i^1$ when $D_i^1 \le t$. When $t < D_i^1 < \mathcal{P}_i - S_i - t$, $\mathsf{dbf}_i^1(t,\vec{F}_i) = 0$ because $t < D_i^1$ and $t < \mathcal{P}_i - S_i - D_i^1$. When $D_i^1 \ge \mathcal{P}_i - S_i - t$, i.e., $t \ge \mathcal{P}_i - S_i - D_i^1$, the demand $\mathsf{dbf}_i^1(t,\vec{F}_i)$ equals E_i^2. Thus, we have proved that the demand of task τ_i is this case when $0 < t < (\mathcal{P}_i - S_i)/2$.

For the demand $\mathsf{dbf}_i^2(t,\vec{F}_i)$, the proof is similar to the one of the demand $\mathsf{dbf}_i^1(t,\vec{F}_i)$. We know $\mathcal{P}_i - S_i - t \le t$ since $(\mathcal{P}_i - S_i)/2 \le t$. By comparing the deadline and length t, $\mathsf{dbf}_i^2(t,\vec{F}_i) = E_i^1$ when $0 < D_i^1 < \mathcal{P}_i - S_i - t$ and $\mathsf{dbf}_i^2(t,\vec{F}_i) = E_i^2$ when $t < D_i^1 < \mathcal{P}_i - S_i$. When $\mathcal{P}_i - S_i \le t \le \mathcal{P}_i$, we know that either frame can contribute to the demand. However, the two frames cannot contribute together since $t < \mathcal{P}_i - S_i$. In other words, the interval length t cannot fit both frames. Thus, we take the maximum execution of the two frames as the demand when $\mathcal{P}_i - S_i \le t \le \mathcal{P}_i$.

It is trivial to see that $\mathsf{dbf}_i^3(t,\vec{F}_i) = E_i^1 + E_i^2$ when $\mathcal{P}_i - S_i \le t \le \mathcal{P}_i$, and the fourth case iterates over the first three cases. In all, we have proved this theorem. ◀

The LP-based algorithm for one-suspension tasks is based on approximating the exact demand in Theorem 12 and the algorithm *The LP-based Algorithm for GMF-PA Tasks* in Figure 4. We replace Lines 6-8 in the MILP algorithm with the linear functions shown in Equation 16 to get the LP algorithm *Heuristic-LP*(D,S) in Line 5 of Figure 4. The linear functions shown in Equations 14-15 are to approximate the two concave portions of the task demand for $\mathsf{dbf}_i^1(t,\vec{F}_i)$ and $\mathsf{dbf}_i^2(t,\vec{F}_i)$, respectively, illustrated by the red dotted lines in Figures 9-10.

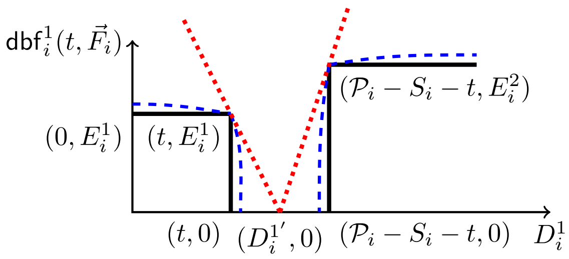


Figure 9 The black solid line shows the demand $\mathsf{dbf}_i^1(t,\vec{F}_i)$, the blue dashed line shows its concave approximation, and the red dotted line shows its linear function when the deadline $D_i^{1'}$ of the last iteration lies between $(t,0)$ and $(\mathcal{P}_i - S_i - t, 0)$.

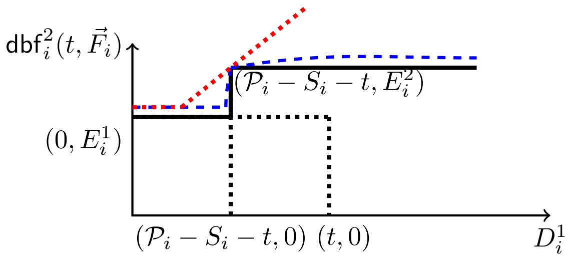


Figure 10 Similar to Figure 9, the dashed and dotted lines show the concave and linear functions of the demand $\mathsf{dbf}_i^2(t,\vec{F}_i)$, shown with the solid line, respectively. The black dotted line shows the frame-wise demand.

$$\mathsf{dbf}_i^{1,\mathrm{linear}}(t,\vec{F}_i) = \begin{cases} \max\begin{cases} \mathsf{dbf}_i^{\mathrm{linear}}(t, D_i^1) \\ \mathsf{dbf}_i^{\mathrm{linear}}(t, \mathcal{P}_i - S_i - D_i^1) \end{cases} \\ \text{when } 0 < D_i^{1'} \le t \\ \max\begin{cases} \frac{0-E_i^1}{D_i^{1'}-t'} \cdot (D_i^1 - t) + E_i^1 \\ \frac{0-E_i^2}{D_i^{1'}-(\mathcal{P}_i-S_i-t')} \cdot (D_i^1 - (\mathcal{P}_i - S_i - t)) + E_i^2 \\ 0 \end{cases} \\ \text{when } t < D_i^{1'} < \mathcal{P}_i - S_i - t \\ \max\begin{cases} \mathsf{dbf}_i^{\mathrm{linear}}(t, D_i^1) \\ \mathsf{dbf}_i^{\mathrm{linear}}(t, \mathcal{P}_i - S_i - D_i^1) \end{cases} \\ \text{when } \mathcal{P}_i - S_i - t \le D_i^{1'} < \mathcal{P}_i - S_i \end{cases} \tag{14}$$

$$\mathsf{dbf}_i^{2,\mathrm{linear}}(t,\vec{F}_i) = \begin{cases} \max\begin{cases} \mathsf{dbf}_i^{\mathrm{linear}}(t, D_i^1) \\ E_i^2 \end{cases} \\ \text{when } E_i^1 \ge E_i^2 \\ \max\begin{cases} \mathsf{dbf}_i^{\mathrm{linear}}(t, \mathcal{P}_i - S_i - D_i^1) \\ E_i^1 \end{cases} \\ \text{when } E_i^1 < E_i^2 \end{cases} \tag{15}$$

$$\mathsf{dbf}_i^{\text{linear}}(t,\vec{F}_i) = \begin{cases} \mathsf{dbf}_i^{1,\text{linear}}(t,\vec{F}_i) & \text{when } 0 < t < (\mathcal{P}_i - S_i)/2 \\ \mathsf{dbf}_i^{2,\text{linear}}(t,\vec{F}_i) & \text{when } (\mathcal{P}_i - S_i)/2 \le t < \mathcal{P}_i - S_i \\ \mathsf{dbf}_i^{3,\text{linear}}(t,\vec{F}_i) & = E_i^1 + E_i^2, \\ & \text{when } \mathcal{P}_i - S_i \le t \le \mathcal{P}_i \\ \mathsf{dbf}_i^{4,\text{linear}}(t,\vec{F}_i) & = \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot (E_i^1 + E_i^2) + \mathsf{dbf}_i(t - \lfloor \frac{t}{\mathcal{P}_i} \rfloor \cdot \mathcal{P}_i, D_i^1), \\ & \text{when } t > \mathcal{P}_i \end{cases} \qquad (16)$$

The approximation demand $\mathsf{dbf}_i^{\text{linear}}(t,\vec{F}_i)$ is calculated based on the t-length interval. Equation 14 shows that the task demand is approximated when $0 < t < (\mathcal{P}_i - S_i)/2$. This case is illustrated by the red dashed lines shown in Figure 9. The functions are also based on the LP-based iterative process and the initial deadline D_i^1 is assigned by PDA $(\mathcal{P}_i - S_i) \cdot \frac{E_i^1}{E_i^1 + E_i^2}$. The slope of the linear function depends on the frame deadline $D_i^{1'}$ from the last iteration. If the deadline $D_i^{1'}$ lies in the region $(t, \mathcal{P}_i - S_i - t)$, we use the two red dotted lines shown in Figure 9 to approximate the staircase demand. The first line passes the points (t, E_i^1) and $(D_i^{1'}, 0)$, and the second line passes the points $(D_i^{1'}, 0)$ and $(\mathcal{P}_i - S_i - t, E_i^2)$. When the frame deadline $D_i^{1'}$ lies in the region $(0, t]$ or $[\mathcal{P}_i - S_i - t, \mathcal{P}_i - S_i)$, we reuse the linear function $\mathsf{dbf}_i^{\text{linear}}(t, D_i^1)$ shown in Equation 12 to calculate the slopes.

Equation 15 shows the task demand when $(\mathcal{P}_i - S_i)/2 \le t < \mathcal{P}_i - S_i$, the demand functions differ by the values of E_i^1 and E_i^2. In the case of the demand $\mathsf{dbf}_i^2(t,\vec{F}_i)$, the first line equals $\min\{E_i^1, E_i^2\}$, and the second line uses the previous method *computeSlope(D)* to adjust the slope of the linear function as shown in Figure 10. Figure 10 shows the approximate lines when $E_i^1 < E_i^2$, and the case is similar when $E_i^1 \ge E_i^2$. When $t \ge \mathcal{P}_i - S_i$, the demand $\mathsf{dbf}_i^{3,\text{linear}}(t,\vec{F}_i)$ and $\mathsf{dbf}_i^{4,\text{linear}}(t,\vec{F}_i)$ are identical to $\mathsf{dbf}_i^3(t,\vec{F}_i)$ and $\mathsf{dbf}_i^4(t,\vec{F}_i)$ of Equation 13, respectively. Thus, we have created the LP-based algorithm for one-suspension tasks.

8 Experiments

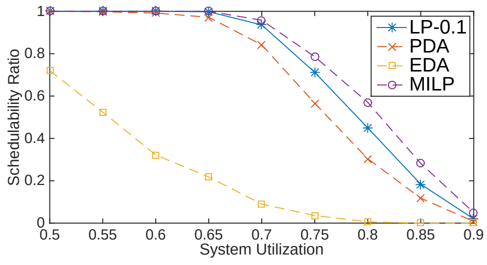
We implement our LP-based algorithms using the commercial solver GUROBI [17] in MATLAB on a 2 GHz Intel Core i5 processor and 8 GB memory machine. We compare our LP-based algorithm with the MILP algorithm [18] and its application to self-suspending tasks [9, 14] on uniprocessor systems. The algorithm LP-δ is the LP-based schedulability test given the maximum error δ of the concave programming algorithm. The algorithm n_{iter}-LP-δ limits the number of iterations to be n_{iter}. Note that we set $\delta = 0.1$, as the constant $\mu = \frac{1}{\delta} \cdot \ln\left(1 + \frac{1}{\delta}\right)$ (e.g. the exponential constants in Equation 2) will be out of range if δ is too small.

The MILP algorithm is introduced in Section 5. The algorithm EDA (equal deadline assignment [9, 3]) assigns each frame the same deadline $(D_i^k = (\mathcal{P}_i - \sum_{i=0}^{N_i-1} S_i^k)/N_i)$, and the algorithm PDA [15, 3] assigns frame deadlines proportional to frame execution time $(D_i^k = (\mathcal{P}_i - \sum_{i=0}^{N_i-1} S_i^k) \cdot E_i^k/E_i)$. Note that we use the schedulability test in the GMF model [3] with the EDA and PDA deadline assignment, since the upper bound of the maximum interval length is bounded [3]. The details of application from GMF-PA to self-suspending tasks can be found in a previous paper [18]. Comparative results on tasks with one suspension and multiple suspensions are shown in Section 8.1 and 8.2, respectively.

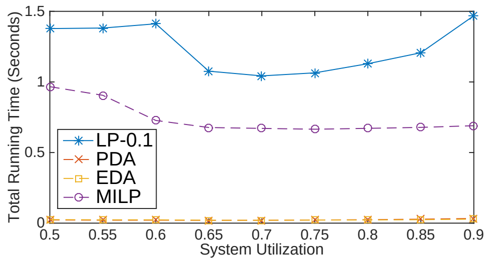
8.1 The Experiments for One-Suspension Self-Suspending Tasks

For one-suspension self-suspending tasks, we compare schedulability ratio and total running time among the algorithms in Figures 11a and 11b, respectively. Since the MILP algorithm does not scale well with an increasing number of tasks (Figure 12) and task periods, we test multiple-suspension self-suspending tasks in Figures 14a and 15a without the MILP algorithm. The schedulability ratio is the number of feasible systems over the total systems. The total running time consists of matrix building time and solver running time.

In the task systems, task periods $\mathcal{P}_i$ are randomly generated in the range $[P_{low}, P_{high}]$. P_{low} and P_{high} are the low and high bounds of the task periods. The UUniFast algorithm [5] divides the utilizations U_i of n tasks under system utilization U_{cap}. The total execution time is $E_i = \mathcal{P}_i \cdot U_i$, and the suspension delay is generated from $[S_{low} \cdot (1-U_i) \cdot \mathcal{P}_i, S_{high} \cdot (1-U_i) \cdot \mathcal{P}_i]$. S_{low} and S_{high} in suspension range $[S_{low}, S_{high}]$ are the low and high suspension index bounds, respectively. The UUniFast algorithm also divides the total execution time into frame execution times. ϵ represents the threshold in the LP-based algorithm shown in Figure 4 and is set to be 0.01. Since all algorithms perform well under small system utilization U_{cap}, we focus on the experiments whose system utilization $U_{cap} \geq 0.5$.



(a) The schedulability ratio of the algorithms at system utilization [0.5, 0.9].



(b) The average running time of the algorithms at system utilization [0.5, 0.9].

Figure 11 The comparison of our LP-based algorithm with the MILP and other polynomial-time algorithms on schedulability ratio and average running time.

In Figures 11a and 11b, the x-axes represent the system utilization $U_{cap} \in [0.5, 0.9]$ with a step size of 0.05. Each task system contains five tasks. The task configuration parameters are $P_{low} = 10$, $P_{high} = 100$, $S_{low} = 0.3$, and $S_{high} = 0.6$. The y-axes represent the schedulability ratio and total running time in Figures 11a and 11b, respectively. The data are the average numbers of 500 runs on each U_{cap}. Figure 11a shows that our LP-δ is better than PDA and EDA algorithms in terms of schedulability ratio. The iteration numbers of all tested LP-δ algorithm are smaller than five. The multiple runs of the LP algorithm make the LP-δ algorithm take slightly longer than the MILP algorithm shown in Figure 11b. The MILP can be relatively efficient for small enough task systems; however, as the number of tasks/frames increases, the MILP running time increases exponentially. Note that in Figure 11, we focus on a small system where we can gauge the effectiveness of the LP in comparison with the MILP and other algorithms. With $U_{cap} = 0.5$, Figure 12 shows that the execution time of the MILP algorithm increases dramatically when the number of tasks increases. Multiple input dimensions affect the execution time of the MILP algorithm, e.g., the task periods. Task periods directly affect the number of integer variables of the MILP algorithm and the running time is longer with higher task periods even when the number of tasks in the system is small. The running time of the LP-based algorithm scales relatively well.

Since we use the concave programming algorithm to guide the LP-based algorithm and have not proved a speed-up factor for the LP-based algorithm, we perform experiments on $\mathcal{L}$ value and maximum error ($\sqrt{\mathcal{L}/\mathcal{L}_{\text{MILP}}} - 1$ by the transformation of Theorem 9). $\mathcal{L}$ shows how close the value of the heuristic algorithm is to the MILP algorithm. $\mathcal{L}$ indicates the minimization of the maximum demands over all tested intervals. E.g., assume there exist two heuristic algorithms that generate $\mathcal{L} = 0.2$ and 0.9, respectively. Both algorithms

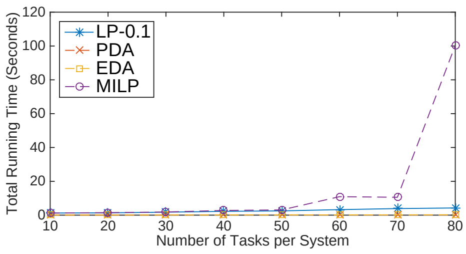
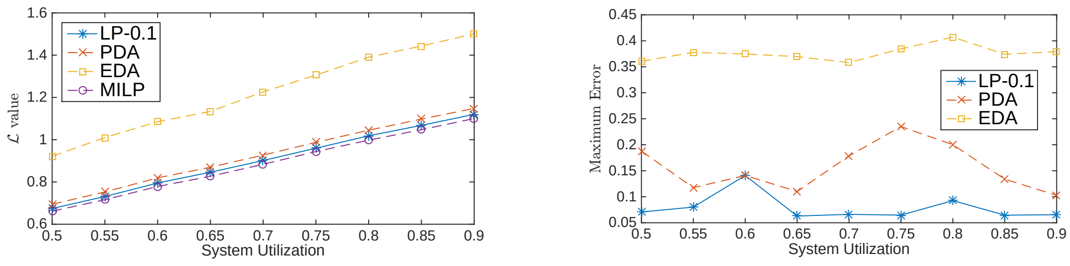


Figure 12 The average running time of the algorithms as the number of tasks increases.

will give successful schedules in the schedulability ratio test, but the one with $\mathcal{L} = 0.2$ is a tighter schedule compared to the other one. If $\mathcal{L} > 1$, the system is not schedulable. We also compare the maximum error of the LP-δ algorithm since the error can be larger than δ.

Figure 13a shows the average $\mathcal{L}$ value of the algorithms among all system utilization points. The LP-0.1 algorithm returns the closest values to the MILP algorithm. The maximum error values shown in Figure 13b take the maximum values among 500 runs in each utilization point. Our LP-based algorithm returns the smallest error across all algorithms.



(a) The $\mathcal{L}$ value of the algorithms at system utilization [0.5, 0.9].

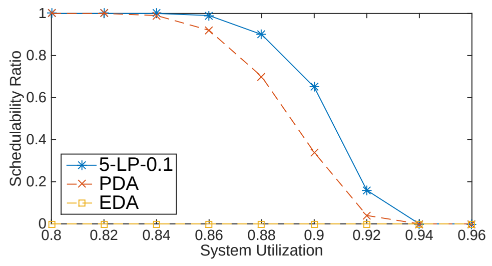
(b) The maximum error of the algorithms compared with the MILP algorithm.

Figure 13 The quality of the LP-based algorithm on the $\mathcal{L}$ value and the maximum system error.

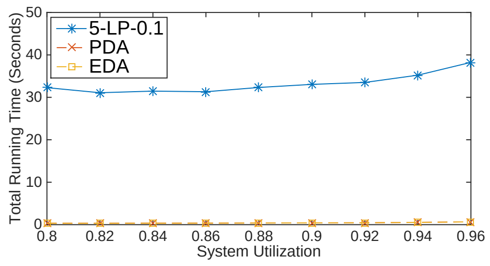
8.2 The Experiments for Multiple-Suspension Self-Suspending Tasks

Among the shown experiments on self-suspending tasks with one suspension frame, the average number of iterations of the LP-based algorithm is smaller than five among all system utilization U_{cap}. Since we believe that the algorithms can approach local optimal with a small number of iterations, we fix the number of iterations to five and test on multiple-suspending tasks. In Figures 14 and 15, the data for each system utilization point is based on 100 runs. Each run of the system contains 30 tasks and each task contains six execution frames separated by five suspending frames (11 frames in total). $P_{low} = 10$ and $P_{high} = 100$. Since the MILP-based approach in this setting takes much longer than the LP-based algorithm, we do not include the MILP-based approach in this experiment. The MILP-based approach takes more than $1.5 * 10^3$ (respectively, $3.0 * 10^3$) seconds with optimality gap (the gap between the lower and upper objective bounds) which is larger than 10% (respectively, 5%).

In Figure 14a, the system utilization $U_{cap} \in [0.8, 0.96]$ with step size of 0.02 is shown on the x-axis. Figure 14a has the suspension range with $S_{low} = 0.1$ and $S_{high} = 0.3$. In Figure 15a, the system utilization $U_{cap} \in [0.5, 0.9]$ with a step size of 0.05 is shown on the x-axis. Figure 15a has the suspension range with $S_{low} = 0.3$ and $S_{high} = 0.6$. Figures 14a and 15a show that our LP-δ is the best among all polynomial-time algorithms in terms of



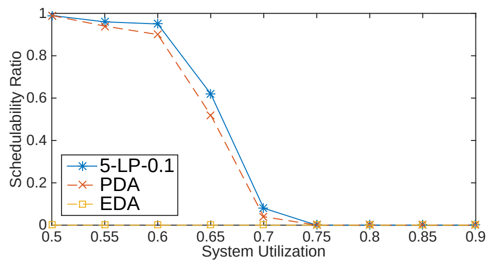
(a) The schedulability ratio of the algorithms at system utilization [0.8, 0.96].



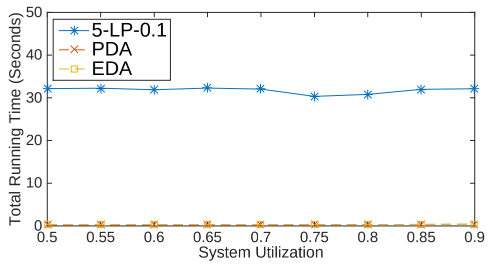
(b) The average running time of the algorithms at system utilization [0.8, 0.96].

Figure 14 Comparison of our LP-based algorithm with other polynomial-time algorithms on the schedulability ratio and average running time.

schedulability ratio. The running times in Figures 14b and 14b reveal that LP-δ also scales well. The improvements for low suspension range $[0.1, 0.3]$ are better than the one with long range $[0.3, 0.6]$. The reason is that when the system specification has more slack time (small frame execution time and short suspending length), the LP-based algorithms can be "trained" to get near optimal parameters during the five iterations. In other words, e.g., the frames deadlines will be equal to their corresponding execution times if there are no slacks for all tasks, and all algorithms will return identical frame deadlines.



(a) The schedulability ratio of the algorithms at system utilization [0.5, 0.9].



(b) The average running time of the algorithms at system utilization [0.5, 0.9].

Figure 15 The comparison of our LP-based algorithm with other polynomial-time algorithms on schedulability ratio and average running time.

Our LP-based algorithm always yields higher schedulability ratio compared to other polynomial-time algorithms. The average running time is competitive overall even when compared with non-mathematical-programming based algorithms such as EDA/PDA.

9 Conclusions

In this paper, we propose a concave programming approximation algorithm and prove its speed-up factor (can approach one) compared to the optimal MILP algorithm. Under the guidance of the tunable small speed-up factor, we present the general LP-based scheme to schedule GMF-PA tasks. We further optimize the LP-based algorithm and apply it to schedule one-suspension tasks. Extensive experiments show that our algorithms improve the schedulability ratio and have competitive running time compared to the previous results.

References

1 B. Andersson. Schedulability analysis of generalized multiframe traffic on multihop-networks comprising software-implemented ethernet-switches. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, April 2008.

2 S. Baruah. Dynamic- and Static-priority Scheduling of Recurring Real-time Tasks. *Real-Time Syst.*, 24(1):93–128, January 2003.

3 S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized Multiframe Tasks. *Real-Time Systems*, pages 5–22, 1999.

4 S. Baruah and N. Fisher. The Partitioned Multiprocessor Scheduling of Sporadic Task Systems. In *Proceedings of the 26th Real-Time Systems Symposium*, pages 321–329, 2005.

5 E. Bini and G. C. Buttazzo. Measuring the Performance of Schedulability Tests. *Real-Time Systems*, pages 129–154, 2005.

6 G. C. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic Scheduling for Flexible Workload Management. *IEEE Transactions on Computers*, pages 289–302, March 2002.

7 D. Buttle. Real-Time in the Prime-Time. In *Proceedings of the 24th Euromicro Conference on Real-Time Systems*, pages xii–xiii, July 2012. doi:10.1109/ECRTS.2012.7.

8 T. Chantem, X. Wang, M.D. Lemmon, and X.S. Hu. Period and Deadline Selection for Schedulability in Real-Time Systems. In *Proceedings of the Euromicro Conference on Real-Time Systems (ECRTS)*, pages 168–177, July 2008.

9 J. J. Chen and C. Liu. Fixed-Relative-Deadline Scheduling of Hard Real-Time Tasks with Self-Suspensions. In *Proceedings of the Real Time Systems Symposium (RTSS)*, December 2014.

10 J. J. Chen, G. von der Bruggen, W. H. Huang, and C. Liu. State of the Art for Scheduling and Analyzing Self-Suspending Sporadic Real-Time Tasks. In *Proceedings of the Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2017.

11 S. Ding, H. Tomiyama, and H. Takada. Scheduling Algorithms for I/O Blockings with a Multi-frame Task Model. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2007.

12 P. Ekberg and W. Yi. Uniprocessor Feasibility of Sporadic Tasks Remains coNP-complete Under Bounded Utilization. In *Proceedings of the 36th IEEE Real-Time Systems Symposium (RTSS)*, 2015.

13 M. R. Garey, D. S. Johnson, and Ravi Sethi. The Complexity of Flowshop and Jobshop Scheduling. *Math. Oper. Res.*, 1(2):117–129, May 1976. doi:10.1287/moor.1.2.117.

14 W.H. Huang and J.J. Chen. Self-Suspension Real-Time Tasks under Fixed-Relative-Deadline Fixed-Priority Scheduling. In *Proceedings of the Design, Automation, and Test in Europe (DATE)*, March 2016.

15 J. Liu. Real-Time Systems. *Prentice Hall*, 2000.

16 A.K. Mok and D. Chen. A multiframe model for real-time tasks. In *Proceedings of the 17th IEEE Real-Time Systems Symposium*, pages 22–29, December 1996.

17 Gurobi Optimization. GUROBI: The state-of-the-art mathematical programming solver. URL: http://www.gurobi.com/.

18 B. Peng and N. Fisher. Parameter Adaptation for Generalized Multiframe Tasks and Applications to Self-Suspending Tasks. In *Proceedings of the 22nd Embedded and Real-Time Computing Systems and Applications (RTCSA)*, August 2016.

19 B. Peng, N. Fisher, and T. Chantem. MILP-based deadline assignment for end-to-end flows in distributed real-time systems. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, RTNS '16, pages 13–22, New York, NY, USA, 2016. ACM. doi:10.1145/2997465.2997498.

20 Bo Peng and Nathan Fisher. Parameter adaptation for generalized multiframe tasks: schedulability analysis, case study, and applications to self-suspending tasks. *Real-Time Systems*, 2017.

21 F. Ridouard, P. Richard, and F. Cottet. Negative results for scheduling independent hard real-time tasks with self-suspensions. In *Proceedings of the 25th Real-Time Systems Symposium*, pages 47–56, December 2004. `doi:10.1109/REAL.2004.35`.

22 J. M. Rivas, J. J. Gutiérrez, J. C. Palencia, and M. G. Harbour. Schedulability Analysis and Optimization of Heterogeneous EDF and FP Distributed Real-Time Systems. In *Proceedings of the 23rd Euromicro Conference on Real-Time Systems (ECRTS)*, pages 195–204, July 2011. `doi:10.1109/ECRTS.2011.26`.

23 M. Stigge, P. Ekberg, N. Guan, and W. Yi. The Digraph Real-Time Task Model. In *Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 71–80, April 2011. `doi:10.1109/RTAS.2011.15`.

24 Martin Stigge and Wang Yi. Graph-based models for real-time workload: a survey. *Real-Time Systems*, 2015.