Geometric Programming Based Optimization of Multiple Periodic Resources in Hierarchical Scheduling

Man-Ki Yoon*, Jung-Eun Kim*, Richard Bradford†, and Lui Sha*
*University of Illinois at Urbana-Champaign, Urbana, IL 61801
Email: {mkyoon, jekim314, lrs}@illinois.edu
†Rockwell Collins, Cedar Rapids, IA 52498
Email: rmbradfo@rockwellcollins.com

Abstract

Hierarchical scheduling of periodic resources has been increasingly applied to a wide variety of real-time systems due to its ability to accommodate various applications on a single system through strong temporal isolation. This leads to the question of how one can optimally design the resource parameters while satisfying the timing requirements of real-time applications. A great deal of research has been devoted to deriving the analytic model for the bounds on the design parameter of a single resource as well as its optimization. The optimization for multiple periodic resources, however, requires a holistic approach due to the conflicting requirements of the limited computational capacity of a system among resources. Thus, this report addresses a holistic optimization of multiple periodic resources with regard to minimum system utilization. We extend the existing analysis on the parameter bounds of a single resource in order for the variable interferences among resources to be captured in the resource bound, and then solve the problem with Geometric Programming (GP). The experimental results show that the proposed method can find a solution very close to the one optimized via an exhaustive search and that it can explore more solutions than a known heuristic method.

I. Introduction

As the processing power of processors has grown, there has been an increasing trend toward integrating many real-time applications on a single system and thus efficiently utilizing the system by allowing the applications to share common hardware devices. In such systems, temporally partitioned hierarchical scheduling [1]–[5] has been widely adopted because of its strong isolation among sets of real-time applications, which either are independently developed or have different functionalities or criticalities. For example, in IMA (Integrated Modular Avionics) architecture [6], applications are often grouped into different partitions according to their design-assurance levels in order to protect high-criticality applications from the faulty behavior of other applications and guarantee their timing requirements. Partitioned resource scheduling can also be used to implement resource reservation to prevent aperiodic tasks from being starved [7], [8].

In such a temporally partitioned hierarchical scheduling, one important question is how much of the computational resource needs to be allocated to each partitioned resource in order for the system to be optimized for a certain metric. For instance, it is desirable in system design process to minimize the system utilization while guaranteeing the timing requirements of both resources and their applications. This is true since a lower-utilized system can be more utilized by accommodating additional workload or, alternatively, the same workload can be implemented by a lower-speed system, which can reduce the unit cost of production.

For a single resource case, the optimized *resource design parameters*, that is, *period* and *execution length*, can be obtained by a method based either on an exact schedulability test [4], [5] or on resource supply and demand functions [1]–[3]. However, it is often intractable to find the optimal set of resource parameters mainly because the local optimality of each resource does not necessarily lead to the global optimal solution [4], [5]. Accordingly, each design parameter cannot be chosen independently. Thus, the optimal selection requires a brute-force search, which is only practical when some parameters are fixed and/or the number of resources is small.

Thus, in this report, we are interested in finding a sub-optimal set of resource design parameters that minimizes the schedulable system utilization; both resources and their tasks are schedulable. Specifically, we consider the periodic resource model introduced in [1]–[5]; each resource $\mathcal{R}$ is periodically released at every T and supplies an execution amount of L to its tasks. For global and local scheduling, we consider fixed-priority scheduling with the assumption that priorities are pre-assigned. The results on the resource parameter bound in previous work were derived by calculating the lower-bound on a resource supply that can satisfy the worst-case demand of the workload. When other resource parameters are unknown, however, a pessimistic assumption on the minimum supply needs to be made; each resource suffers the maximum possible delay. We tackle this problem by parameterizing the worst-case resource supply with the unknown parameters of other resources that can be holistically optimized via *Geometric Programming (GP)* [9], [10]. GP is a non-linear optimization method that can solve a specially formed non-convex problem by transforming it into a convex one through a logarithmic transformation, thus finding the optimal solution efficiently. We present a GP formulation as the solution to the design parameter optimization of multiple periodic resources. As will be shown later, our method can find a solution that is close to the one that can be found by an exhaustive search, and it can explore more solutions than a known heuristic method [5].

The remaining sections of this report are organized as follows: Section II summarizes the related work, and then Section III introduces the system model we consider and then formally describes the parameter optimization problem of multiple periodic resources. In Section IV, we review the previous literature on the analysis of single resource bounds, and then extend these findings to multiple resources in Section V. In Section VI, we explain how to formulate and transform the considered optimization problem to geometric programming. The evaluation results are given in Section VII. Finally, Section VIII concludes this report.

II. Related Work

Shin *et al.* [1] proposed the periodic resource model in a hierarchical scheduling that facilitates the schedulability analysis of the workload of tasks (child) under a periodic resource supply (parent). The authors presented the exact schedulability analysis of a workload set in a periodic resource under RM and EDF scheduling and derived the utilization bounds. In [2], Almeida *et al.* analyzed a similar periodic server model by introducing the server availability function. They also developed a heuristic algorithm for server (resource) parameter optimization for the minimum system utilization, in which the search space is reduced to a set of *deadline points*. Lipari *et al.* [3] also considered the server parameter optimization problem in a hierarchical scheduling system with a different approach of schedulability analysis. These three works all used linear models of resource supply to represent the resource supply (supply bound function, availability function, characteristic function, respectively) and considered a single resource. In contrast, Davis *et al.* [4], [5] presented the exact worst-case response time analysis of tasks under deferrable server, periodic server, and sporadic server. The authors also addressed the parameter selection optimization of multiple servers and provided a greedy algorithm. Through an empirical investigation, the authors claimed that the optimal parameter selection for multiple resources is a holistic problem. In Section VII, we compare the heuristic method in [5] with our GP-based optimization method. Additionally, in [11], Saewong *et al.* developed a response time analysis for real-time guarantees of tasks under sporadic server and deferrable server.

In [12], Easwaran introduced a generalized periodic resource model called *Explicit Deadline Periodic* (EDP) resource model, and proposed an exact algorithm for determining the optimal resource parameter that minimizes the ratio of length to period of an EDP resource. The same problem for periodic resource model was addressed by Shin et al. [13], in which the authors presented a polynomial-time sufficient algorithm. Both problems were addressed by Dewan *et al.* [14] and Fisher [15] by proposing fully-polynomial-time approximation algorithms that improve both the optimality and time complexity. None of these papers, however, consider the problem of optimizing the parameters of multiple resources.

Geometric Programming [9], [10] has been widely applied to a broad range of non-linear, non-convex optimization problems such as digital circuit gate sizing [16], resource allocation in communication systems [17], information theory [18], etc. An extensive discussion of geometric programming can be found in [10].

III. Problem Description

A. System Model

We consider a uniprocessor consisting of a set of independent periodic resources $\mathfrak{R} = \{\mathcal{R}_i | i = 1, \ldots, N^{\mathfrak{R}}\}$. Each resource $\mathcal{R}_i$ is characterized by an unknown tuple of (T_i, L_i), where T_i and L_i are the period and the execution length of the resource, respectively. In each resource $\mathcal{R}_i$, a set $\mathbf{\Gamma_i} = \{\tau_j | j = 1, \ldots, N^{\mathbf{\Gamma_i}}\}$ of tasks run in a fixed-priority preemptive schedule such as Rate Monotonic [19]. Each task τ_j is represented by $\tau_j := (e_j, p_j, d_j)^1$, where e_j is the worst-case execution time, p_j is the minimum inter arrival time between successive releases, and d_j is the

^{1}More precisely, each task should be represented as $\tau_{i,j}$ if the task belongs to resource i. For the simplicity of notations, however, we use the abbreviation τ_j.

relative deadline. In this report, we assume that $d_j = p_j$. We then further assume that there is no synchronization or precedence constraints among tasks, and task releases are not bound to the release of resources [5].

The resources $\mathfrak{R}$ are also scheduled in a fixed-priority manner and we assume their deadline, D_i, is equal to the period. In addition, we consider that resource priorities are given, assuming, for example, the priorities are assigned according to criticalities. We note that the optimization method in this report cannot be applied to cases when resource priorities are not given. Additionally, a resource is idled if there is no task ready to execute. We also assume that there is no resource release jitter.

Finally, there is no strict assumption on the smallest time unit of resource parameters, i.e., $T_i, L_i \in \mathbf{R}^+$ for all i. However, we also consider cases when the parameters are constrained to integers, i.e., T_i or $L_i \in \mathbf{N}$. As will be shown later, the integrality constraint makes the parameter optimization much harder to solve.

B. Problem Description

Given a set of resources $\{\mathcal{R}_i\}$ and the corresponding task sets $\{\mathbf{\Gamma_i}\}$, our problem is to find the set of the resource parameters, $\{(T_i, L_i)\}$ for $i = 1, \ldots, N^{\mathfrak{R}}$, which minimizes the overall system utilization, U_s, while guaranteeing the schedulabilities of the resources and the tasks. Here, the overall system utilization can be represented by

$$U_s = \sum_{i=1}^{N^{\mathfrak{R}}} \frac{c_1 \cdot \delta + c_2 \cdot L_i}{T_i}, \tag{1}$$

where δ is the resource context-switch overhead, and c_1, c_2 are weights given by the system designer [3]. In this report, we set both c_1 and c_2 to 1 and assume that each resource will consume a context-switch overhead of δ at each release.

In summary, task sets and their temporal characteristics, i.e., $\{(e_j, p_j, d_j) | \forall \tau_j \in \mathbf{\Gamma_i}\}$ for all $\mathcal{R}_i \in \mathfrak{R}$, are given as input, and we will find the optimal set of resource parameters $\{(T_i, L_i) | \forall \mathcal{R}_i \in \mathfrak{R}\}$ for a given c_1, c_2, and δ.

IV. Parameter Bounds of Single Resources

In this section, we summarize the previous literature on the analysis of single resource bounds. The analysis presented in this section is primarily based on the periodic resource model introduced in [1]. It should be noted, however, that the periodic server model in [2] can be similarly used without loss of generality.

A. Sufficient Resource Bound for Task Schedulability

In a partitioned resource whose period T_i and length L_i are unknown, we can derive the lower-bound of L_i (or the upper-bound of T_i) with respect to T_i (or L_i) that makes τ_j in $\mathcal{R}_i$ schedulable by using the periodic resource model introduced in [1], [2]. Informally speaking, the key idea of previous work is that a task can be schedulable if the minimum resource supply ($\mathbf{sbf_\Gamma}(t)$ in [1] or $A\_s(t)$ in [2]) can match the maximum workload demand generated by τ_j and its higher-priority tasks during a time interval t.

In fixed-priority global scheduling, the minimum supply of periodic resource i is delivered to tasks when its $(k-1)^{th}$ execution has just been finished at time 0 with minimum interferences from higher-priority resources. Then, the subsequent executions from the k^{th} release is maximally delayed by higher-priority resources. Since no assumption is made on the periods and lengths of other resources, we assume that the worst-case occurs when the resource suffers zero interference in the $(k-1)^{th}$ release and $T_i - L_i$ thereafter, as depicted in Figure 1. For this worst-case minimum supply we can derive the linear lower-bound supply function $\mathbf{lsbf}_{\mathcal{R}_\mathbf{i}}(t)$ as in [1], which is defined as follows:

$$\mathbf{lsbf}_{\mathcal{R}_\mathbf{i}}(t) = \frac{L_i}{T_i} \cdot (t - 2 \cdot (T_i - L_i)). \tag{2}$$

Note that it is identical to $A'\_s(t)$ with $\alpha = \frac{L_i}{T_i}$ and $\Delta = T_i - L_i$ in [2].

Now, let us consider task τ_j in resource $\mathcal{R}_i$ whose period T_i is fixed. Then, let us define $L_i^{min}(\tau_j, T_i)$ as the minimum required length of $\mathcal{R}_i$ that guarantees to schedule τ_j. In order to derive $L_i^{min}(\tau_j, T_i)$, we can consider the situation in which τ_j barely meets its deadline at time $t = d_j$ with the worst-case interference from higher-priority tasks. Since we make no assumption on task offsets, the worst-case response time of τ_j occurs when τ_j and the tasks with higher priority than τ_j are released simultaneously at the end of $\mathcal{R}_i$'s execution and then suffers the worst-case

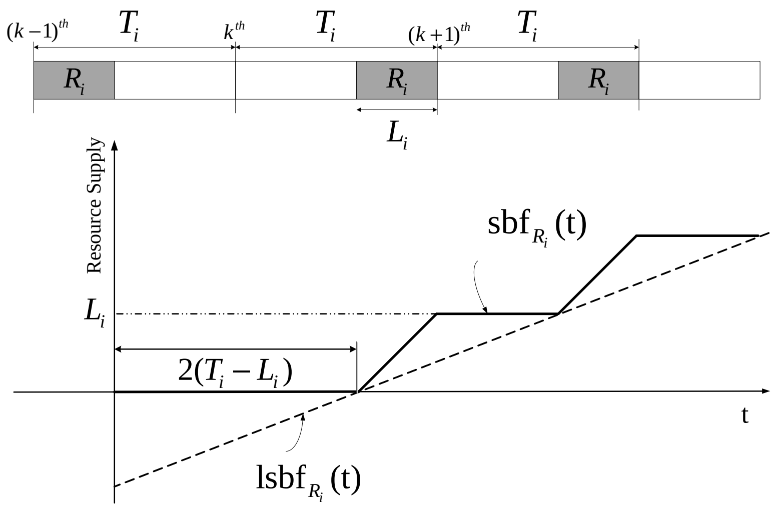

Fig. 1: The worst-case release pattern of a periodic resource $\mathcal{R}_i$ when the parameters of higher-priority resources are unknown.

preemptions from the higher-priority tasks from k^{th} release and thereafter, which we define as *the critical instant*.[2] Now, let us denote I_j as the worst-case workload generated by τ_j and the higher-priority tasks from the critical instant to the deadline of τ_j as follows:

$$I_j = e_j + \sum_{\tau_h \in hp(\tau_j)} \left\lceil \frac{d_j}{p_h} \right\rceil \cdot e_h,$$

where $hp(\tau_j)$ is the set of tasks with higher priority than τ_j. In order to guarantee τ_j's schedulability, the minimum supply delivered by the resource has to be greater than or equal to the worst-case workload during the time interval d_j. Thus,

$$\mathbf{lsbf}_{\mathcal{R}_\mathbf{i}}(d_j) = \frac{L_i}{T_i} \cdot (d_j - 2 \cdot (T_i - L_i)) \geq I_j. \tag{3}$$

Accordingly, the minimum required resource length, i.e., $L_i^{min}(\tau_j, T_i)$, for task τ_j with a given resource period T_i can be obtained by solving the quadratic inequality in Eq. (3), which results in

$$L_i^{min}(\tau_j, T_i) = \frac{-(d_j - 2T_i) + \sqrt{(d_j - 2T_i)^2 + 8I_jT_i}}{4}. \tag{4}$$

Note that Eq. (4) is equivalent to Eq. (23) in [1] and to Eq. (12) with $\beta = 1$ in [2]. It is also important to note that Eq. (3) is only sufficient and not necessary condition; τ_j can be schedulable if and only if there exists a time instant $t \leq d_j$ such that $\mathbf{lsbf}_{\mathcal{R}_\mathbf{i}}(t) \geq I_j$. In this report, we use the sufficient condition in Eq. (3) since the presence of time in the necessary condition makes the proposed optimization method not applicable to the problem under consideration.

B. Optimization of Single Resource Bound

As seen in the previous section, the required resource length that can guarantee the schedulability of τ_j in $\mathcal{R}_i$ is lower-bounded by Eq. (4). Figure 2 shows $L_i^{min}(\tau_j, T_i)$ for an example periodic resource consisting of $\{\tau_1 = (5, 20), \tau_2 = (10, 100), \tau_3 = (15, 150)\}$, where $d_j = p_j$ for all j. In order to find the minimum required length of a resource i for a given period T_i, we take the maximum of the bounds $L_i^{min}(\tau_j, T_i)$ over all tasks in $\mathbf{\Gamma_i}$, which therefore can be defined as follows:

$$L_i^{min}(T_i) = \max_{\tau_j \in \mathbf{\Gamma_i}} \Big(L_i^{min}(\tau_j, T_i) \Big). \tag{5}$$

[2]In this report, we do not consider task jitters. However, without loss of generality, the presented analyses in this report can be similarly applied to cases with jitters. For example, the worst-case situation of τ_j is when all the higher-priority tasks have experienced their maximum jitters and are released at the same time with τ_j.

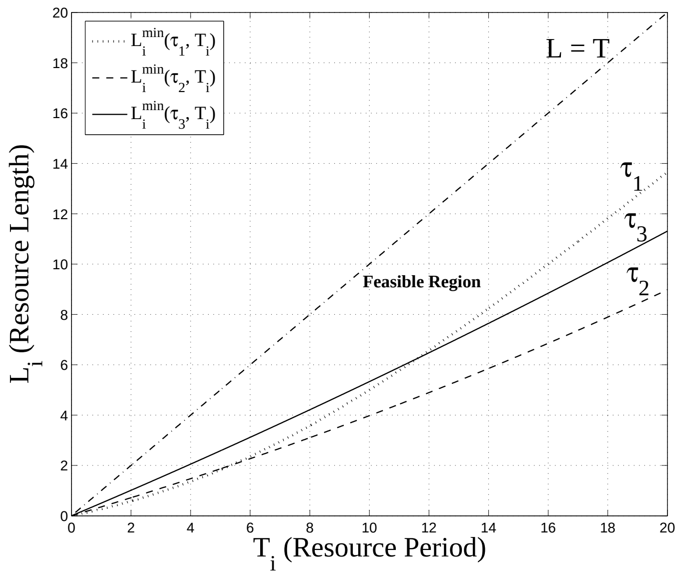

Fig. 2: Minimum required resource length $L_i^{min}(\tau_j, T_i)$ for $\{\tau_1 = (5, 20), \tau_2 = (10, 100), \tau_3 = (15, 150)\}$.

Thus, if we take L_i from the feasible region, that is, $L_i^{min}(T_i) \leq L_i \leq T_i$, all $\tau_j \in \mathbf{\Gamma_i}$ are guaranteed to meet their deadlines. For example, we can see from Figure 2 that $L_i^{min}(T_i)$ is lower-bounded by τ_3 until around $T_i = 11.6$. For $T_i > 11.6$, $L_i^{min}(\tau_1, T_i)$ becomes the new bound. If $T_i = 15$, the resource length L_i has to be longer than approximately 9.12 in order to guarantee that the tasks meet their deadlines.

Although the main consideration of this report is the optimization of multiple resources, we briefly address the effects of various constraints for the case of a single resource. In Figure 3, we drew the resource utilization function

$$U_i(T_i) = \frac{\delta}{T_i} + \frac{L_i^{min}(T_i)}{T_i}$$

for $T_i \in [1, 20]$ of the example used in Figure 2: (a) no context-switch overhead ($\delta = 0$), (b) $\delta = 1$, (c) $\delta = 4$, and (d) $\delta = 1$ and integrality constraint on L_i^{min}. First of all, if context-switch overhead is not a consideration, the minimum resource utilization is achieved at the minimum possible period because

$$\frac{\partial}{\partial T_i}\Big(\frac{L_i^{min}(\tau_j, T_i)}{T_i}\Big) \geq 0$$

if $I_j \leq d_j$. Note that if $I_j = d_j$, $L_i^{min}(\tau_j, T_i)$ becomes T_i, which means that a dedicated processor needs to be allocated to the resource in order to make the task schedulable. When a context-switch overhead is considered, in contrast, the resource utilization function is no longer monotonically increasing with T_i, and the optimal period appears at a longer period due to the hyperbolic nature of $\frac{\delta}{T_i}$. If an integrality constraint on resource length L_i is enforced, the graph becomes a sawtooth function and the optimal period in such a case is not necessarily identical to the one obtained with real-valued L_i. The optimization of a single resource has been extensively studied in previous literatures. Interested readers can refer to [1]–[5], [12]–[15], [20].

V. Parameter Bounds of Multiple Resources

In this section, we extend the analysis of the single resource bound in the previous section to the case of multiple resources via a parameterization of unknown resource parameters.

A. Lower-bound Supply Function Considering Unknown Parameters of Higher-Priority Resources

The bound for single resources explained in Section IV was derived with the assumption that each resource experiences no interference in the $(k-1)^{th}$ release and then suffers the delay of $T_i - L_i$ from the k^{th} release. This is a pessimistic assumption, since, in reality, high priority resources would suffer no (if they are the highest ones)

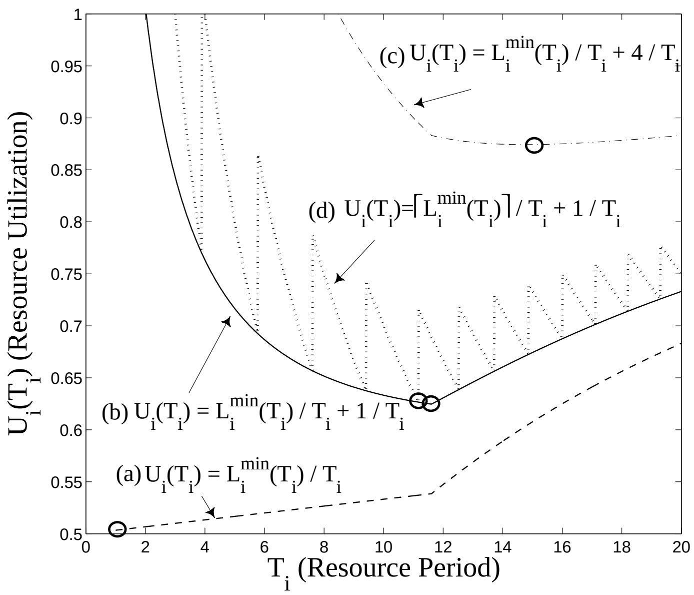

Fig. 3: Resource Utilization $U_i(T_i)$ for $T_i \in [1, 20]$ with (a) $\delta = 0$ (no context-switch overhead), (b) $\delta = 1$, (c) $\delta = 4$, and (d) $\delta = 1$ and integrality constraint on $L_i^{min}(T_i)$. Each circle represents the minimum $U_i(T_i)$ for each case.

or only a few preemptions. Thus, an exact method is required [4], [5], which is not useful for an optimization of multiple resource parameters due to its high time complexity. This necessitates holistic optimization of multiple resource parameters.

Thus, we now parameterize the linear lower-bound supply function $\mathbf{lsbf}_{\mathcal{R}_i}(t)$ (Eq. (2)) with the periods and execution lengths of the higher-priority resources, $hp(\mathcal{R}_i)$. The worst-case release pattern of $\mathcal{R}_i$ occurs when $\mathcal{R}_i$ and $hp(\mathcal{R}_i)$ are released simultaneously. The worst-case busy period of $\mathcal{R}_i$, denoted as $w_{\mathcal{R}_i}$, is the maximum time duration that $\mathcal{R}_i$ can take to execute L_i when it is released simultaneously with the higher-priority resources at the k^{th} release, which can be obtained by the traditional exact analysis:

$$w_{\mathcal{R}_i}^{k+1} = L_i + \sum_{\mathcal{R}_h \in hp(\mathcal{R}_i)} \left\lceil \frac{w_{\mathcal{R}_i}^k}{T_h} \right\rceil \cdot L_h,$$

where $w_{\mathcal{R}_i}^0 = L_i$ and the worst-case busy period of $\mathcal{R}_i$ is $w_{\mathcal{R}_i}$ when it converges, i.e., $w_{\mathcal{R}_i} = w_{\mathcal{R}_i}^k = w_{\mathcal{R}_i}^{k+1}$ for some k. Thus, the worst-case delay at the k^{th} release and also thereafter (called the *initial latency* in [2]) can be represented as

$$\Delta_{\mathcal{R}_i} = \sum_{\mathcal{R}_h \in hp(\mathcal{R}_i)} \left\lceil \frac{w_{\mathcal{R}_i}}{T_h} \right\rceil \cdot L_h. \tag{6}$$

However, this iterative method can only be applicable to brute-force optimization. Thus, we take a different approach; we approximate $\Delta_{\mathcal{R}_i}$. During a time interval of T_i, the maximum workload generated by $\mathcal{R}_i$ and $hp(\mathcal{R}_i)$ can be represented by:

$$w_{\mathcal{R}_i} = L_i + \sum_{\mathcal{R}_h \in hp(\mathcal{R}_i)} \left\lceil \frac{T_i}{T_h} \right\rceil \cdot L_h.$$

Note that with this equation, we can avoid iterative calculation by assuming the number of invocations of higher-priority resources during T_i, not during the exact busy period of $\mathcal{R}_i$. Also note that it is a safe bound as long as $\mathcal{R}_i$ meets its deadline, i.e., $D_i = T_i$. Now, we remove the ceiling in order to linearize $w_{\mathcal{R}_i}$, which results in

$$w_{\mathcal{R}_i} = L_i + \sum_{\mathcal{R}_h \in hp(\mathcal{R}_i)} \left(\frac{T_i}{T_h} + 1 \right) \cdot L_h,$$

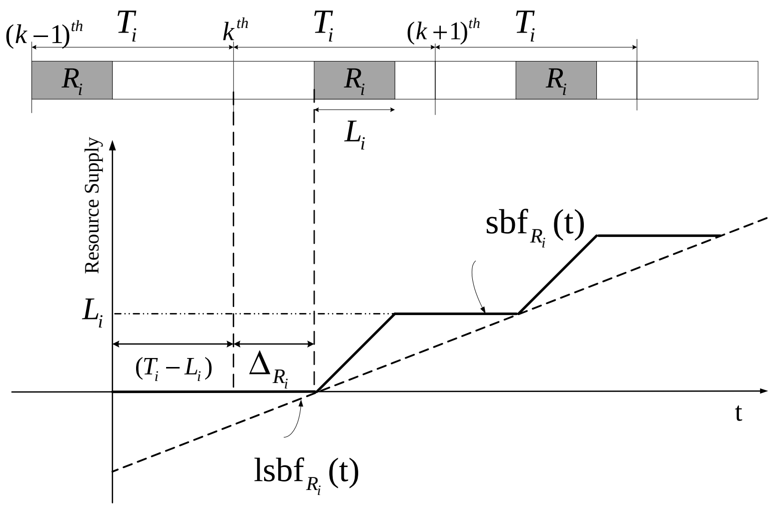

Fig. 4: The worst-case release pattern of $\mathcal{R}_i$ considering the periods and execution lengths of higher-priority resources.

because $\lceil x \rceil \leq x+1$. Then, the new linear lower-bound supply function of $\mathcal{R}_i$ during a time interval t parameterized with $hp(\mathcal{R}_i)$ can be presented as follows:

$$\mathbf{lsbf}_{\mathcal{R}_\mathbf{i}}(t) = \frac{L_i}{T_i} \cdot (t - (T_i - L_i) - \Delta_{\mathcal{R}_i}), \tag{7}$$

where

$$\Delta_{\mathcal{R}_i} = \sum_{\mathcal{R}_h \in hp(\mathcal{R}_i)} \left(\frac{T_i}{T_h} + 1 \right) \cdot L_h. \tag{8}$$

If the resource periods are harmonic with each other, we can use $\Delta_{\mathcal{R}_i} = \sum_{\mathcal{R}_h \in hp(\mathcal{R}_i)} \left(\frac{T_i}{T_h} \right) \cdot L_h$ instead.

Now, the minimum supply of $\mathcal{R}_i$ is delivered to tasks when it experiences no interference in the $(k-1)^{th}$ execution and the maximum preemption delay from its higher-priority resources thereafter, that is, $\Delta_{\mathcal{R}_i}$ (Figure 4). Now, the sufficient resource bound constraint for task schedulability, i.e., Eq. (3), is refined as follows:

$$\mathbf{lsbf}_{\mathcal{R}_\mathbf{i}}(d_j) = \frac{L_i}{T_i} \cdot (d_j - (T_i - L_i) - \Delta_{\mathcal{R}_i}) \geq I_j. \tag{9}$$

Accordingly, the minimum required resource length for task τ_j with a given resource period T_i, i.e., $L_i^{min}(\tau_j, T_i)$ in Eq. (4), becomes

$$L_i^{min}(\tau_j, T_i) = \frac{-(d_j - T_i - \Delta_{\mathcal{R}_i}) + \sqrt{(d_j - T_i - \Delta_{\mathcal{R}_i})^2 + 4 I_j T_i}}{2}. \tag{10}$$

Again,

$$L_i^{min}(T_i) = \max_{\tau_j \in \mathbf{\Gamma_i}} \Big(L_i^{min}(\tau_j, T_i) \Big). \tag{11}$$

Although the bound presented here is not exact and may incur approximation error, it enables us to optimize multiple resources holistically with high efficiency, as will be described in Section VI.

B. Non-convexity of Multiple Resource Optimization

We present a simple example of two resources in order to show the non-convexity of the optimization problem of multiple resources. Let us consider Figure 5, which shows the utilization functions, U_1 and U_2, of two randomly generated resources, $\{\mathcal{R}_1, \mathcal{R}_2\}$, and the system utilization function, $U_s = U_1 + U_2$, over the period in $[1, 140]$. In this example, the resources have the same period, i.e., $T_1 = T_2$, for simplicity of representation, and δ is set to 1. $\mathcal{R}_1$ has a higher priority than $\mathcal{R}_2$, thus $\Delta_{\mathcal{R}_1} = 0$ and $\Delta_{\mathcal{R}_2} = 2 \cdot L_1$. From the graphs, we can first see that the system utilization function U_s is not convex (and neither is U_1), which is shown by the straight line drawn between

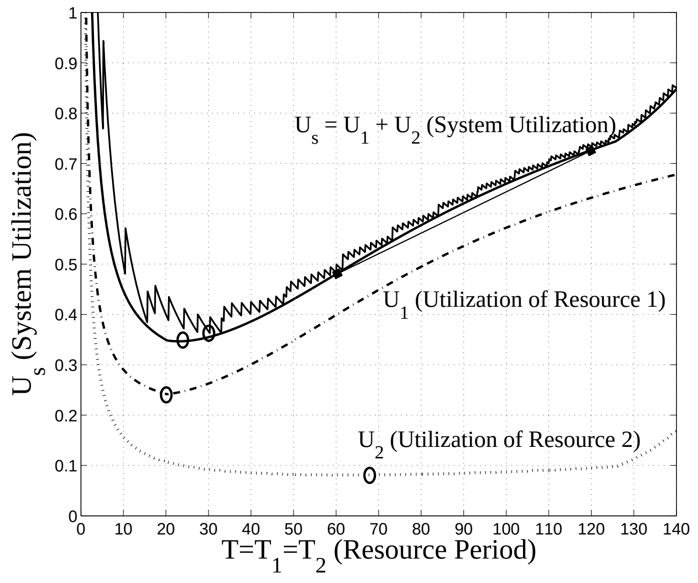

Fig. 5: System utilization of two resources, $\{\mathcal{R}_1, \mathcal{R}_2\}$, and its non-convexity. The sawtooth-shaped graph represents the system utilization when the resource execution lengths are constrained to integers. Circles represent the minimum utilization of the system and the resources.

$T = 60$ and $T = 120$:

$$U_s(90) \not\leq \frac{U_s(60) + U_s(120)}{2}.$$

Furthermore, while the resources achieve minimum utilization at $T_1 = 20.3$ and $T_2 = 63.9$, respectively, these do not necessarily lead to the global optimality which occurs at $T = T_1 = T_2 = 23.1$; this issue is addressed also in [4]. Additionally, as in the single resource case in Figure 3, if L_is are to be discrete, the optimal solution occurs at a different point ($T = 30.3$). In fact, when we consider such integrality constraints, determining the optimal solution requires extensive branch and bound searches.

In this example, the resources have the same period. The optimization of multiple resource parameters will be harder to solve once we consider a higher number of resources and arbitrary resource periods.

VI. Holistic Optimization of Resource Parameters via Geometric Programming

In this section, we formulate the parameter optimization problem of multiple periodic resources with Geometric Programming (GP) [9], [10].

A. Geometric Programming Formulation

A non-linear, non-convex optimization problem can be solved by geometric programming if the problem can be formulated in a special form as follows [10]:

$$\begin{array}{ll} \text{Minimize} & f_0(\mathbf{x}) \\ \text{Subject to} & f_i(\mathbf{x}) \leq 1,\ i = 1, \ldots, n_p, \\ & g_j(\mathbf{x}) = 1,\ j = 1, \ldots, n_m, \end{array}$$

where f and g are *posynomial* and *monomial* functions, respectively, and $\mathbf{x}$ are the optimization variables. A function $g_j(\mathbf{x})$ is monomial if it can be represented as:

$$g_j(\mathbf{x}) = c_j \prod_{k=1}^{n_j} x_k^{a_k},$$

where $c_j \in \mathbf{R}^+$ and $a_k \in \mathbf{R}$. A posynomial function is a sum of monomials, and thus can be expressed as:

$$f_i(\mathbf{x}) = \sum_{k=1}^{n_i} c_k x_1^{a_{1k}} x_2^{a_{2k}} \cdots x_n^{a_{nk}},$$

where $c_k \in \mathbf{R}^+$ and $a_{jk} \in \mathbf{R}$. Also, f/g is a posynomial and f^{a_x} is also a posynomial[3] if $a_x \in \mathbf{R}^+$.

In summary, the objective function and the inequality constraints must be in posynomial forms, and the equality constraints can only be in monomial forms.

We now formulate the optimization problem of the multiple resource parameters in a GP form. As previously stated in Section III-B, we are given a set of periodic resources $\{\mathcal{R}_i\}$ with unknown parameters, T_i and L_i, their task sets $\{(e_j, p_j, d_j) | \forall \tau_j \in \mathbf{\Gamma_i}\}$, and the resource context-switch overhead are knownδ. Thus, the optimization variables are $\mathbf{T} = (T_1, \ldots, T_{N^{\mathfrak{R}}})$ and $\mathbf{L} = (L_1, \ldots, L_{N^{\mathfrak{R}}})$.

Objective Function

The objective function (1) in Section III-B is already in a posynomial form, thus it can be represented as follows:

$$f_o(\mathbf{T}, \mathbf{L}) = \sum_{i=1}^{N^{\mathfrak{R}}} (c_1 \delta + c_2 L_i) \cdot T_i^{-1}, \tag{12}$$

where $c_1, c_2, \delta \geq 0$.

Resource Bound Constraint

The resource bound for each resource $\mathcal{R}_i$ is constrained by Eq. (9) for each $\tau_j \in \mathbf{\Gamma_i}$, which can be reexpressed as follows:

$$\frac{T_i \cdot (L_i + I_j) + \Delta_{\mathcal{R}_i} \cdot L_i}{L_i \cdot (L_i + d_j)} \leq 1, \tag{13}$$

where $\Delta_{\mathcal{R}_i} = \sum_{\mathcal{R}_h \in hp(\mathcal{R}_i)} \left(\frac{T_i}{T_h} + 1 \right) \cdot L_h$ (i.e., Eq. (8)), d_j is the relative deadline of τ_j, and I_j is the worst-case workload generated by the task itself and the higher-priority tasks during the time interval of d_j, both of which are constants for a given input. However, the above inequality does not conform to a posynomial form because of the posynomial term in the denominator, i.e., $L_i \cdot (L_i + d_j) = L_i^2 + L_i \cdot d_j$ (recall that a denominator must be monomial). Observe, however, that $L_i + d_j$ can be approximated with a monomial by the following geometric mean approximation [17]. Let us first denote it as

$$g_i(L_i) = u_1(L_i) + u_2(L_i),$$

where $u_1(L_i) = L_i$ and $u_2(L_i) = d_j$. Then, we now approximate $g_i(L_i)$ with

$$\tilde{g}_i(L_i) = \left(\frac{u_1(L_i)}{\gamma_1} \right)^{\gamma_1} \cdot \left(\frac{u_2(L_i)}{\gamma_2} \right)^{\gamma_2}, \tag{14}$$

where

$$\gamma_1 = \frac{u_1(x_0)}{g_i(x_0)} \text{ and } \gamma_2 = \frac{u_2(x_0)}{g_i(x_0)}$$

where $x_0 \in \mathbf{R}^+$ is a constant that satisfies $\tilde{g}_i(x_0) = g_i(x_0)$. The approximated monomial $\tilde{g}_i(L_i)$ then can be rewritten as:

$$\tilde{g}_i(L_i) = \left(\frac{L_i}{\gamma_1} \right)^{\gamma_1} \cdot \left(\frac{d_j}{\gamma_2} \right)^{\gamma_2},$$

with

$$\gamma_1 = \frac{x_0}{x_0 + d_j} \text{ and } \gamma_2 = \frac{d_j}{x_0 + d_j}.$$

[3]If α_x is allowed to be a non-integer, the form is called *Generalized Geometric Program (GGP)*, which can be transformed to GP.

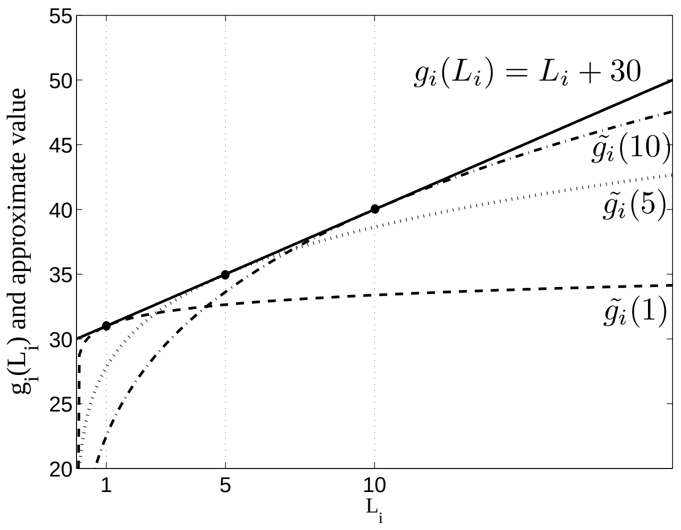

Fig. 6: $g_i(L_i) = L_i + 30$ and its approximated monomial $\tilde{g}_i(L_i)$ at $x_0 = 1, 5$, and 10. $\tilde{g}_i(x_0)$ is tangent to $g_i(L_i)$ at $L_i = x_0$.

Finally, Eq. (13) can be formulated as the following posynomial constraint:

$$(T_i \cdot (L_i + I_j) + \Delta_{\mathcal{R}_i} \cdot L_i) \cdot (L_i \cdot \tilde{g}_i(L_i))^{-1} \leq 1, \tag{15}$$

where

$$\Delta_{\mathcal{R}_i} = \sum_{\mathcal{R}_h \in hp(\mathcal{R}_i)} \big((T_i + T_h) \cdot T_h^{-1} \cdot L_h\big).$$

Note that the approximation quality of $\tilde{g}_i(L_i)$ depends on the choice of x_0, as shown in Figure 6. Thus, in the optimization procedure, we iteratively approximate $\tilde{g}_i(L_i)$ by updating γ_1 and γ_2 according to the intermediate solution of L_i. That is, until the objective value converges, we use L_i at k^{th} step as x_0 at $(k+1)^{th}$ step. In our experiment, the initial value of x_0 was chosen as 1, and the objective value converged within one or two iterations.

Resource Schedulability Constraint

Each resource must be schedulable, that is, $L_i + \Delta_{\mathcal{R}_i} \leq T_i$, which can be expressed as the following posynomial constraint:

$$\Big(L_i + \sum_{\mathcal{R}_h \in hp(\mathcal{R}_i)} \big((T_i + T_h) \cdot T_h^{-1} \cdot L_h\big)\Big) \cdot T_i^{-1} \leq 1. \tag{16}$$

B. Mixed-Integer Geometric Programming for Integrality Constraints

In a real system, the resource periods and execution lengths are multiples of the smallest time unit because of scheduling granularity. In this case, we can think of integrality constraints on T_i and L_i values, however this makes the optimization problem much harder to solve as illustrated in Section IV-B. A GP is called *Mixed-Integer Geometric Programming (MIGP)* [10] if one or more variables are constrained to be integers. A branch and bound method [21] can be used and often finds a global optimal solution; however, it cannot be scalable with problem size. In this report, we use a heuristic for rounding fractional variables. The heuristic is not a globally optimal method but can efficiently find a near-optimal solution. We note that the choice of a branching or rounding method is orthogonal to the optimization presented in the previous subsection.

The key idea of the rounding heuristic is that we first find the optimal solution without any integrality constraint, which is called *GP Relaxation*. Then, we round each variable up or down to its nearest integer value at each step. Here we assume the integrality constraint on resource execution lengths, $\mathbf{L}$; however, this can be similarly applied to $\mathbf{T}$. Now, let us denote $\{\mathbf{T}^*, \mathbf{L}^*\}$ as the optimal solution found with the relaxed GP.[4] For each $L_i^* \in \mathbf{L}^*$, we

[4]The problem itself is infeasible if no solution exists for the relaxed GP.

TABLE I: Evaluation parameters.

Parameter	Value
Number of resources, $N^{\mathfrak{R}}$	$\{2,3,4,5\}$
Number of tasks per resource, $N^{\mathbf{\Gamma_i}}$	$[2,8]$
Task execution time, e_j	$[1,30]$
Task period, p_j	$[50,2000]$
Context-switch overhead, δ	1

calculate the distance between L_i^* and its nearest integer as follows:

$$\varepsilon(L_i^*) = \min\left(|L_i^{*\uparrow}|, |L_i^{*\downarrow}|\right),$$

where $x^{\uparrow} = \lceil x \rceil - x$ and $x^{\downarrow} = x - \lfloor x \rfloor$. Then, we find the least fractional L_i^* such that

$$L_i^* = \arg\min_{L_i^* \in \mathbf{L}^*} (\varepsilon(L_i^*)).$$

Once we have found such L_i^*, we then solve the GP by adding the following monomial constraint to the original GP:

$$L_i \cdot \widetilde{L_i^*}^{-1} = 1,$$

where $\widetilde{L_i^*}$ is $L_i^* + L_i^{*\uparrow}$ or $L_i^* - L_i^{*\downarrow}$ depending on $|L_i^{*\uparrow}|$ and $|L_i^{*\downarrow}|$.

Now let us consider the case when both $\mathbf{T}$ and $\mathbf{L}$ are to be integers. The rounding process is similar to that described above; however, in each iteration we round one T_i and one L_j up or down, where i is not necessarily equal to j. The heuristic works as follows. We first find the relaxed optimal solution $\{\mathbf{T}^*, \mathbf{L}^*\}$. Then, we round the least fractional T_i^* up or down to its nearest integer, $\widetilde{T_i^*}$. We then again solve the modified GP and then find the least fractional L_j^*. If T_j^* is an integer we round L_j^* up. Otherwise, we find its nearest integer as described above. In the next iteration, we find the least fractional T_i^* among the remaining fractional $\mathbf{T}^*$. However, this time we check if L_i^* is already an integer. If this is the case, we round T_i^* down. In summary, the heuristic iterates $N^{\mathfrak{R}}$ times, and during each iteration we update one pair of fractional T_i^* and L_j^*; however, T_i^* (L_j^*) can only be rounded down (up) if its corresponding L_i^* (T_j^*) is already an integer. The reason is that for each T_i^* found by GP, its corresponding L_i^* is the lower-bound of L_i. Thus, if L_i^* is already an integer, T_i^* cannot be rounded up even if it is closer to $\lceil T_i^* \rceil$ because this will make the problem infeasible by violating the resource bound constraint (15). A similar argument is applied to L_j^*. Note that we need to update $\tilde{g}_i(L_i)$ (Eq. (14)) as well in each iteration.

We note that this rounding heuristic is based on the hope that a better solution of parameter pairs could be found by independently rounding each parameter of a resource; updating both T_i^* and L_i^* of a resource at the same time may increase the possibility of local optima by moving radically in the feasible region. As previously mentioned, however, the presented rounding heuristic does not guarantee the optimality of the solution.

VII. Evaluation

In this section, we evaluate the proposed optimization method presented in Section V and VI.

A. Evaluation Method

Table I summarizes the experimental parameters used for the evaluations. We consider the cases with $2, 3, 4,$ and 5 resources, and for each case, we generated 100 random input sets with the parameters. The number of tasks per resource, the task execution time and period are uniformly randomly chosen in the given range. For the simplicity of evaluations, the context-switch overhead was fixed to 1. With these parameters, we compare the following methods:

- Exhaustive Search: From the highest priority resource to the lowest one, we recursively assign each resource period from 1 to T_{max} with a step size of s. For each period T_i, L_i^{min} is determined by Eq. (11) and (10) with $\Delta_{\mathcal{R}_i}$ calculated by Eq. (6). Recall that the system utilization obtained with this exhaustive search is still not the exact globally optimal solution as explained in Section V.
- GP with the upper-bound on T_{max}: The GP optimization method presented in Section VI with an additional set of constraints on the upper-bound on resource periods, that is, $T_i \cdot T_{max}^{-1} \leq 1$.
- GP without the upper-bound on T_{max}: Identical to the above except that there is no upper-bound on T_{max}. Note that in our GP-based optimization, the upper-bound of the resource period is unnecessary.

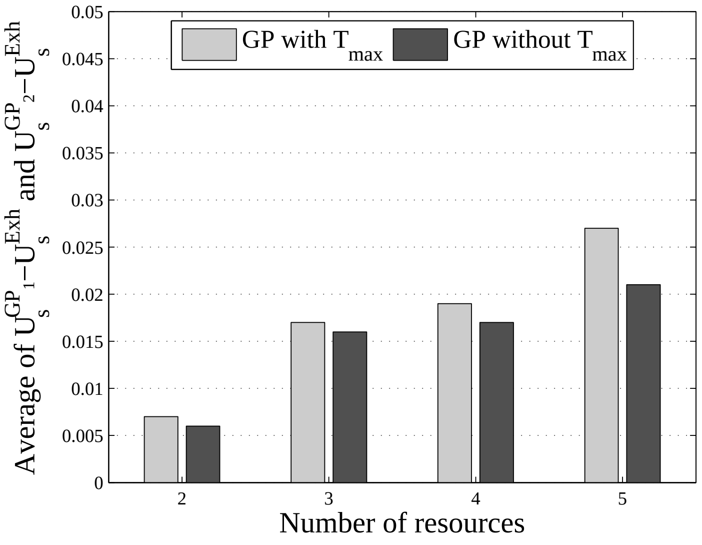

Fig. 7: The average differences of $U_s^{GP_1}$ and $U_s^{GP_2}$ to U_s^{Exh} with different numbers of resources.

The priority of each resource is assigned according to the utilization sum of tasks in each resource; the higher the utilization is, the higher its priority. This assumption is only for the evaluation purpose. Readers interested in priority optimization can refer to [4], [5]. The priority of each task in each resource is assigned based on Rate Monotonic priority assignment [19]. The GPs were solved using GGPLAB [22].

B. *Evaluation Metric*

We compare the methods above in terms of the minimum system utilization, i.e., Eq (1). We denote the solution of each method as U_s^{Exh}, $U_s^{GP_1}$, and $U_s^{GP_2}$, respectively. For each input, we calculate the difference of U_s^{Exh} from $U_s^{GP_1}$ and $U_s^{GP_2}$, that is,

$$U_s^{GP_1} - U_s^{Exh} \text{ and } U_s^{GP_2} - U_s^{Exh},$$

respectively, and then take the average of 100 random input sets for each setting. It should be noted that we do not compare the solving time of each method because while GP can solve a problem within a few seconds, the exhaustive search normally takes 10–60 minutes or more depending on the problem size and the choices of T_{max} and s.

C. *Evaluation Results*

Figure 7 compares the minimum system utilization found by the exhaustive search and our GP method increasing the number of resources from 2 to 5. It should be noted that the exhaustive search takes a longer time to solve cases with six or more resources. Thus, we evaluated cases with 2, 3, 4, and 5 resources. It took about 30 minutes – 1 hour to solve one input consisting of 5 resources with a step size of 0.5. T_{max} and the step size were set to 100 and 0.5, respectively, and no integrality constraint was posed. As we can see from the result, the average difference of $U_s^{GP_1}$ and U_s^{Exh}, i.e., $U_s^{GP_1} - U_s^{Exh}$, increases with the number of resources. This is mainly because of the approximation error in $\Delta_{\mathcal{R}_i}$ described in Section V. Recall that while the exhaustive search calculates the minimum supply of each resource by using the iterative equation (Eq. (6)), our GP method takes the ceiling off and calculates the interference from higher-priority resources during the interval of its period (Eq. (8)). When there are only two resources, the error of the approximation is small; however as the number of resources increases, the error accumulates from higher-priority resources to lower-priority ones. Nevertheless, we can see that the differences between the two methods are quite small, indicating that our method can find a solution that is close to the one that can be found by the exhaustive search; 0.007–0.027 average difference compared to the solutions of the exhaustive search. Another interesting observation is that GP_2, a GP without the upper-bound on T_{max}, can find better solutions than GP_1; the error of GP_2 compared to the exhaustive search is only 0.006–0.021 on average.

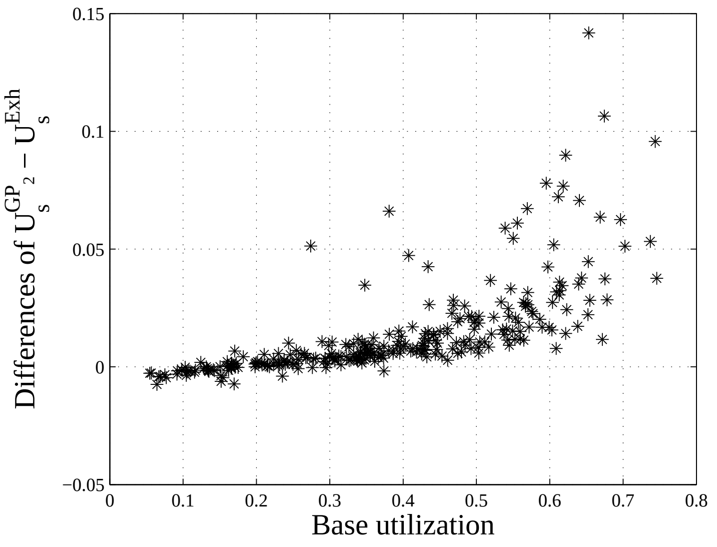

Fig. 8: The differences of $U_s^{GP_2}$ and U_s^{Exh} with various base utilizations.

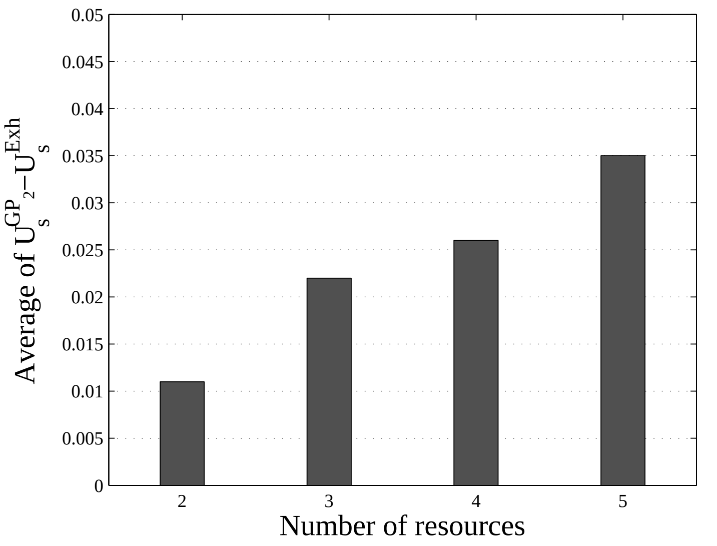

Fig. 9: The average differences of $U_s^{GP_2}$ and U_s^{Exh} when integrality constraints on resource parameters are posed.

This result might be expected because of the limited search space of GP_1: when the workload of a resource is significantly lower than the other resources, its optimal period may appear beyond T_{max}. In fact, for some input sets, $U_s^{GP_2}$ were lower than U_s^{Exh}. One can find better optimal solutions with the exhaustive search by setting T_{max} higher, however, this can be limited by the input size. Hereafter we compare $U_s^{GP_2}$ with U_s^{Exh}.

With the same inputs, we evaluated the differences of $U_s^{GP_2}$ to U_s^{Exh} with various base utilizations, i.e., the sum of all task utilizations, as shown in Figure 8. From the graph, we can see that the differences increase with the base utilizations. A similar argument as above can be used to explain this correlation. That is, a higher base utilization implies that there exist resources with higher utilizations, and thus those tend to have shorter periods and longer execution lengths. We attribute this, again, to the approximation error of $\Delta_{\mathcal{R}_i}$ in Eq. (8).

Next, we evaluate our method when both resource periods and execution lengths are be integers. For this

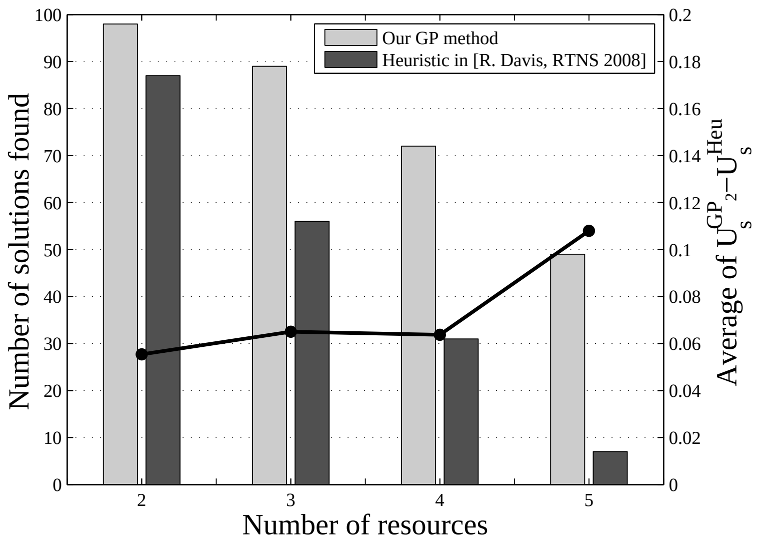

Fig. 10: The number of solutions found by the proposed GP method and the heuristic in [5] (Bars), and the average difference of $U_s^{GP_2}$ and U_s^{Heu} (Line).

evaluation, we used the same input as above. In the exhaustive search, the resource periods are recursively assigned from 1 to 100 with a step size of 1. Then, the ceiling of the minimum execution length for each resource period, $\lceil L_i^{min}(T_i) \rceil$ (Eq. (11)), was used in Eq. (6). For our GP, the rounding heuristic explained in Section VI-B was used. As can be seen from Figure 9, the difference between the two methods increases with the number of resources for the same reasons as above. However, this time the difference became bigger with the constraints compared to the results in Figure 7. One can easily expect that the increased difference arises from the simplicity of the rounding heuristic used. A more accurate method could be combining an estimation-based selection of rounding variables. Note that resource utilization functions are in different shapes depending on the priorities and workload of tasks, as can be seen in Figure 5. In the example, $\mathcal{R}_2$ is more flexible in choosing its parameters since its utilization function, $\frac{L_i^{min}(T_i)}{T_i}$, is almost flat during a wide range of its period. In such a case, it would be better to round off $\mathcal{R}_2$ first and then proceed to higher utilization resources in order to avoid local optima. However, it is still difficult to find the global optimal solution with such a rounding-based approach due to the sawtooth nature of the curves. Thus, one may want to consider using a class of branch-and-bound methods to enhance the quality of the solution.

Lastly, we compare our GP method with the heuristic proposed in [5]. The method finds the optimal parameters for each resource in turn from the highest to the lowest resources; for each resource, it iterates over a range of periods and for each period, it finds the optimal resource length by a binary search. When the optimal pair of period and length is found, the same process is applied to the next priority resource. For the comparison, we used the same inputs as above. In the heuristic, each resource period is assigned from 1 to 1000 with a step size of 0.1, and each resource length was found at the granularity of 0.1. For our GP method, both T_{max} and integrality were not assumed. Figure 10 shows i) the numbers of solutions found by each method and ii) the average difference of the minimum utilization between the two methods. As can be seen from the bars, our GP method finds more solutions than the heuristic, and in the experiment, all input sets for which a solution was found by the heuristic were also solved by our method. We can also see that as the number of resources increase, the gap also increases; with 5 resources, our method found 49 solutions among 100 input sets, but only 7 solutions were found by the heuristic.[5] This follows from the greedy nature of the heuristic; the parameters for a high-priority resource were locally optimized without considering the feasibilities of lower-priority resources. In contrast, although our method is not a globally optimal method either, it can explore more solutions due to its ability to take into account the

[5]It should be noted that the exact number of feasible solutions is unknown as this requires a true optimal method. Among 100 input sets for each case, some input sets may not be feasible in the first place. Also, the main reason that each method finds fewer number of solutions as the number of resources increase is because the base system utilization also increases. For example, the average base utilizations of 100 input sets with 2 and 5 resources are 0.281 and 0.663, respectively.

variable interferences among resources simultaneously in the GP optimization process. However, the qualities of the solutions found by our method are worse than those found by the heuristic, as the line plot in Figure 10 shows. Each marker on the line is the average of the difference of the minimum system utilization found by our method, i.e., $U_s^{GP_2}$, to that found by the heuristic, i.e., U_s^{Heu}, for the input sets that the heuristic found; with $2, 3$ and 4 resources, $U_s^{GP_2} - U_s^{Heu}$ are between 0.055 and 0.065 in average, and with 5 resources, the difference is 0.108. The spike at 5 resources could be explained by the low number of solutions found. Although our method achieved lower system utilization for some input sets, the heuristic could find better solutions in most cases. Such differences mainly arise from the optimality of the analysis used by the heuristic. That is, when the parameters of higher-priority resources and the period of the resource under analysis are fixed, the (local-)optimal resource length is found by the binary search which is based on the exact analysis [5]. On the other hand, as explained Section V, our analysis considers the worst-case scenarios that are sufficient but not necessary, and it is also based on the approximation of $\Delta_{\mathcal{R}_i}$, both of which lead to schedulability loss. From this evaluation, we can conclude that there is a trade-off between the solution feasibility (our GP method) and the solution quality (the heuristic of [5]).

VIII. Conclusion

In this report we addressed the problem of design parameter optimization of multiple periodic resources in hierarchical scheduling. We extended the existing analysis on a single resource in order for our resource supply model to be able to capture the variable parameters of higher-priority resources. In order to solve the problem, we formulated it via Geometric Programming and provided a heuristic method for integrality constraints. The presented analysis on the resource supply model and its optimization is not a globally optimal method due to the approximation error of worst-case resource interference. However, we believe that one can benefit from the presented optimization method in designing a hierarchical system with a large number of partitioned resources due to its ability to yield a high-quality solution with a high scalability. For future work, we will investigate the possibility of applying the presented analysis and optimization method to a hierarchical system under non-preemptive global scheduling such IMA (Integrated Modular Avionics) scheduling.

References

[1] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *Proceedings of the 24th IEEE Real-Time Systems Symposium*, 2003, pp. 2–13.

[2] L. Almeida and P. Pedreiras, "Scheduling within temporal partitions: response-time analysis and server design," in *Proceedings of the 4th ACM international conference on Embedded software*, 2004, pp. 95–103.

[3] G. Lipari and E. Bini, "Resource partitioning among real-time applications," in *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, 2003, pp. 151–158.

[4] R. I. Davis and A. Burns, "Hierarchical fixed priority pre-emptive scheduling," in *Proceedings of the 24th IEEE Real-Time Systems Symposium*, 2005, pp. 389–398.

[5] R. Davis and A. Burns, "An investigation into server parameter selection for hierarchical fixed priority pre-emptive systems," in *Proccedings of Real-Time and Network Systems, RTNS*, 2008.

[6] *ARINC Specification 651: Design Guidance for Integrated Modular Avionics*, ser. ARINC report. Airlines Electronic Engineering Committee (AEEC) and Aeronautical Radio Inc, Nov. 1991.

[7] B. Sprunt, L. Sha, and J. Lehoczky, "Aperiodic task scheduling for hard-real-time systems," *Journal of Real-Time Systems*, vol. 1, pp. 27–60, 1989.

[8] M. Spuri and G. Buttazzo, "Scheduling aperiodic tasks in dynamic priority systems," *Journal of Real-Time Systems*, vol. 10, pp. 179–210, 1996.

[9] R. J. Duffin and E. Peterson and C. Zener, *Geometric Programming - Theory and Application*. John Wiley, New York, 1967.

[10] S. P. Boyd, S.-J. Kim, L. Vandenberghe, and A. Hassibi, "A tutorial on geometric programming," *Optimization and Engineering*, vol. 8, pp. 67–127, 2007.

[11] S. Saewong, R. R. Rajkumar, J. P. Lehoczky, and M. H. Klein, "Analysis of hierarhical fixed-priority scheduling," in *Proceedings of the 14th Euromicro Conference on Real-Time Systems*, 2002, pp. 152–160.

[12] A. Easwaran, "Compositional schedulability analysis supporting associativity, optimality, dependency and concurrency," *PhD thesis, Computer and Information Science, University of Pennsylvania*, 2007.

[13] I. Shin and I. Lee, "Compositional real-time scheduling framework with periodic model," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 3, pp. 30:1–30:39, May 2008.

[14] F. Dewan and N. Fisher, "Approximate bandwidth allocation for fixed-priority-scheduled periodic resources," in *Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium*, 2010, pp. 247–256.

[15] N. Fisher, "An FPTAS for interface selection in the periodic resource model," in *Proceedings of the 17th International Conference on Real-Time and Network Systems*, 2009, pp. 127–136.

[16] S. P. Boyd, S.-J. Kim, D. D. Patil, and M. A. Horowitz, "Digital circuit optimization via geometric programming," *Operations Research*, vol. 53, pp. 899–932, 2005.

[17] M. Chiang, "Geometric programming for communication systems," *Commun. Inf. Theory*, vol. 2, pp. 1–154, Jul. 2005.

[18] M. Chiang and S. P. Boyd, "Geometric programming duals of channel capacity and rate distortion," *IEEE Trans. Inform. Theory*, vol. 50, pp. 245–258, 2004.

[19] C. L. Liu and J. W. Layland, “Scheduling algorithms for multiprogramming in a hard real-time environment,” *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, January 1973.
[20] A. Easwaran, M. Anand, I. Lee, and O. Sokolsky, “On the complexity of generating optimal interfaces for hierarchical systems,” in *Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, 2008.
[21] O. K. Gupta and A. Ravindran, “Branch and bound experiments in convex nonlinear integer programming,” *Management Science*, vol. 31, pp. 1533–1546, 1985.
[22] “GGPLAB: A Simple Matlab Toolbox for Geometric Programming,” http://www.stanford.edu/~boyd/ggplab/.