J Supercomput
DOI 10.1007/s11227-010-0506-z

Optimal task execution times for periodic tasks using nonlinear constrained optimization

Nasro Min-Allah · Samee Ullah Khan · Wang Yongji

Abstract Designing real-time systems is a challenging task and many conflicting issues arise in the process. Among them, the most fundamental one is the adjustment of appropriate values for task parameters such as task periods, deadlines, and computation times that directly influence the system feasibility. Task periods and deadlines are generally known at design stage and remains fixed throughout, however, task computation times fluctuates significantly. For a better quality of service or higher system utilization, higher task computation values are required, while this flexibility comes at the price of system infeasibility. To the best of our knowledge, no optimal solution exists for extracting the optimal task computation times in a given range so that the overall system remains feasible under a specific scheduling algorithm. In this paper, we present a generalized bound on the task schedulability defined as a nonlinear inequality $h_i \leq 0$ in the space of the execution times c_i. Based on this bound, the adjustment problem of tasks execution times, which determines the optimum c_i for a better system performance while still meeting all temporal requirements, is addressed by solving the standard nonlinear constrained optimization problem. Simulations on synthetic task sets are presented to compare the performance of our work with the

N. Min-Allah
COMSATS Institute of Information Technology, Islamabad 44000, Pakistan
e-mail: nasar@comsats.edu.pk

S.U. Khan (✉)
Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58108-6050, USA
e-mail: samee.khan@ndsu.edu

W. Yongji
State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100080, China
e-mail: ywang@itechs.iscas.ac.cn

Published online: 10 November 2010

most celebrated result, i.e., LL-bound by Liu and Layland in (J. ACM 20(1):40–61, 1973).

Keywords Real-time systems · Fixed-priority scheduling · Feasibility analysis · Nonlinear programming

1 Introduction

A real-time task τ_i is an executable entity of work that, at a minimum, is characterized by a worst-case execution time c_i (WCET) [26] and a time constraint [27]. In practical systems, c_i fluctuates in the range $c_i \in [c_i^{\min}, c_i^{\max}]$, while on the theoretical side to guarantee reliable system behavior in worst case scenario, c_i is kept maximum, i.e., $c_i = c_i^{\max}$. A typical timing constraint of a real-time task is the deadline d_i $(c_i \le d_i)$, which represent the time before that a task should complete its execution without causing any damage to the system [10]. Similarly, an instance/job of a task is generated every p_i units of time, called time period.

Before applying to the time critical applications, real-time systems must be designed and validated carefully to avoid undesired consequences [13, 16, 17]. Designing a real-time system [18] is a challenging task and two questions need to be addressed at this stage, i.e., (i) what criteria to use for adjusting task parameters, and (ii) which scheduling algorithm to be incorporated, so that maximum system utilization is achieved [1, 3]. In hard real-time systems, running the time critical applications with a scheduling algorithm require predictable behaviors under all possible circumstances. Even if a deadline has to be missed, it is better to miss the deadline of a less important task than missing the deadline of a critical task. As far as predictability is concerned, among available scheduling algorithms, fixed priority scheduling is the choice of modern real time systems due to its simplicity and applicability [5, 8, 11, 12, 23] and it is always known in advance that which task will miss the deadline when the system is overloaded. The most mature scheduling algorithm in static priority scheduling class is the rate-monotonic (RM) scheduling algorithm which is the focus of this paper. Task priorities in RM are based on task periods, i.e., the larger is the task period, higher is the task priority and vice versa.

Let $\Gamma = \{\tau_1, \dots, \tau_n\}$ denotes a periodic task system, where each task τ_i $(i = 1, 2, \dots, n)$ is represented by three essential parameters (c_i, p_i, d_i). For each task τ_i, its utilization is defined as: $U_i = c_i / p_i$. We define a cumulative utilization U_{lub} of periodic task system Γ as:

$$U_{\text{lub}} = \sum_{i=1}^{n} \frac{c_i}{p_i}. \tag{1}$$

Based on the task periods and deadlines, there are three major classes of task models [11]:

- Implicit-deadline: The constraint for each task $\tau_i \in \Gamma$ is $d_i = p_i$.
- Constrained-deadline: The constraint for each task $\tau_i \in \Gamma$ is $d_i \le p_i$.
- Arbitrary-deadline: For any task $\tau_i \in \Gamma$, d_i and p_i is unrelated.

The most common case is the implicit-deadline systems [29]. Since task period are usually set by the system requirement, deadlines, and execution times can be modified in order to improve system utilization. Recently, a significant contribution is made on sensitivity analysis [6, 14, 15], based on exact conditions. In contrast, the aim of this work is to provide a generalized bound for a periodic task that can be solved with standard nonlinear constrained optimization technique for attaining optimal computation demand for a given task in the context of fixed-priority preemptive uniprocessor scheduling. In the rest of the paper, the term generalized bound and polylinear bound are used interchangeably, unless stated otherwise.

The fundamental issue investigated during the past thirty years for the traditional rate monotonic scheduling algorithm can be described as: Given a task set Γ, determine whether the set Γ is RM feasible? Two questions arise from this problem formulation.

- **P-1:** If the answer is YES, then how much freedom in terms of CPU utilization do we have to increase the value of the parameter c_i in order to have a better system performance while still satisfying schedulability constraints?
- **P-2:** If the answer to the question posed by RM schedulability is NO, then how to adjust/lower the parameter c_i so that the new task set becomes schedulable?

Mainly, schedulability tests come up with a straight YES or NO answer, however, techniques do exist where the above questions are addressed with sensitivity analysis [6, 14]. Authors in [14, 15] also treats system with utilization over 100% by using the concept of slack which allows tasks computation times to be adjusted so that the system becomes schedulable or higher system utilization is achieved. In this paper, we address the same issue by applying optimization techniques. We are focusing on transforming the infeasible task set into feasible and the feasible one into the system that results in higher system utilization. The prior work of [32] enables us to represent the long list of inequalities (Eq. 9) by a single inequality, and thus provide a solution for obtaining optimal computation times for the tasks according to RM scheduling policy. This formulation is referred to as the task execution time c_i adjustment problem (C-AP), throughout this paper.

C-AP is of significance importance in the design phase of real-time and embedded systems; in application areas such as decision-making, database query processing and numerical solutions, where the task execution time varies greatly depending upon the execution precision. In such applications, the longer a task takes to execute, the better are the results produced. Similarly, the tasks execution times vary significantly with different hardware configurations, and hence the total execution cost is reduced by efficiently utilizing the available hardware resources. This variation in the execution times greatly influence the feasibility of the systems and no optimal solution exists in literature prior to this work for obtaining the best possible values.

There are generally two solutions to formulate the above schedulability problems (**P-1** and **P-2**). The first is based on the sufficient conditions [4, 22, 23] such as the LL-bound, proposed by Liu and Layland in [22]. The LL-bound says that a task set

Γ consisting of n tasks is RM feasible if

$$U_{\text{lub}}(n) \leq n\bigl(2^{1/n} - 1\bigr). \tag{2}$$

Similarly, another recently proposed bound, H-bound has higher acceptance ratio than LL-bound. According to H-hound, the task set is RM-feasible if

$$\prod_{i=1}^{n}(U_i + 1) \leq 2. \tag{3}$$

The second solution is based on the necessary and sufficient condition [4, 6, 21, 24]. The first solution has two disadvantages, (i) the best value of the utilization factor cannot exceed the bounds, and (ii) it can result in the pessimistic answer, as the condition is only sufficient. The second solution is strongly recommended, however, the general algorithms developed for solving the optimization problem cannot be directly applied to it [5, 20] because of the long list of inequalities that are not only linked with logical AND (at set level) relationship but also through logical OR relationship (at task level) [5].

Considering the case of fixed values of p_i, this work inherits the fundamental idea of inequality and optimization techniques proposed in [31, 32]. The list of inequalities at task level is converted into a single inequality with the help of a mathematical transformation, and C-AP is addressed by solving the standard nonlinear programming problem.

The main contributions are as follows:

- We present a polylinear bound on task schedulability given by an inequality that helps to apply the optimization techniques to solve the problems discussed in [5, 20].
- C-AP is formulated as a nonlinear programming problem. Specifically, three distances from a point $(c_1, c_2, \ldots, c_n)$ to the schedulability boundaries are mainly investigated: (i) by how much amount any execution time c_i can be increased/decreased and keeping/reaching the schedulability for task τ_i, (ii) by how much amount all the execution times can be scaled (up or down) to hit the schedulability boundary, and (iii) by how much amount all tasks execution times can be varied while respecting schedulability region for the task set.

The advantage of using the optimization technique for solving the C-AP is threefold. First, the schedulability test problem becomes a special case of the C-AP when the task execution time c_i is fixed. Secondly, the C-AP is converted into a typical optimization problem concerning the minimization or maximization of a function, subject to different types of constraints (equality or inequality) in operation research, which can be solved by any mature algorithm for an efficient solution. Third, the solution to the problem is not only feasible, but also optimal. Any desired characteristics such as processor utilization, reward function, or the error cost function can be designed as an objective function.

The rest of the work is organized as follows. Section 2 presents the generalized bound on task schedulability with the help of a mathematical transformation. In

Sect. 3, the problem of task computation times adjustment is represented as a classical nonlinear constrained optimization problem. Experimental results are given in Sects. 4 and 5. Finally, conclusions are stated in Sect. 6.

2 A generalized bound on task schedulability

Rate Monotonic Analysis (RMA), in its exact form, is the classical way of determining RM task feasibility. The following brief discussion provides the foundation for our work.

The workload $w_i(t)$ constituted by τ_i at time t consists of its execution demand c_i as well as the interference it encounters due to higher priority tasks from τ_{i-1} to τ_1 and can be expressed mathematically as

$$w_i(t) = c_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{p_j} \right\rceil c_j. \tag{4}$$

A periodic task τ_i is feasible if we find some $t \in [0, d_i]$ satisfying

$$\min_{0<t\leq p_i} \big(w_i(t) \leq t\big). \tag{5}$$

In other words, task τ_i completes its computation requirements at time $t \in [0, p_i]$, if and only if the entire request from the $i-1$ higher priority tasks and computation time of τ_i is completed at or before time p_i. As t is a continuous variable, there are infinite numbers of points to be tested.The first attempt to limit the infinite number of points in interval $t \in [0, t]$ is made in [21]. The authors show that $w_i(t)$ is constant, except at finite number of points, where tasks are released, called RM scheduling points. Consequently, to determine whether τ_i is schedulable, $w_i(t)$ is computed only at multiples of $\tau_i \leq \tau_j$, $1 \leq j \leq i$. Specifically, let

$$S_i = \left\{ ap_b \,\middle|\, b = 1, \ldots, i;\ a = 1, \ldots, \left\lfloor \frac{p_i}{p_b} \right\rfloor \right\}. \tag{6}$$

We have the following fundamental theorem to determine whether an individual task is feasible or not.

Theorem 1 [21] *Given a set of n periodic tasks* $\tau_1, \ldots, \tau_n$, τ_i *can be feasibly scheduled for all tasks phasings using RM iff*

$$\min_{t\in S_i} \frac{w_i(t)}{t} \leq 1. \tag{7}$$

With above theorem, $w_i(t)$ is needed to be analyzed only at a finite number of points. The time demand $w_i(t)$ for task τ_i is tested at all scheduling points in the set S_i.

In the following, the schedulability of task τ_i is ensured by solving an inequality $g_i(c_1, c_2, c_i) \leq 0$. The implicit function $g_i \leq 0$ is called the generalized bound of τ_i. Extending the work in [5, 6, 21], the following discussion leads us to the generalized bound on task schedulability represented as a polylinear surface in the space of the task execution times c_i.

Definition 1 ($\mathbf{F_i}$) The schedulability region $\mathbf{F_i}$ is defined as:

$$F_i(c_1, c_2, \dots, c_i) = \big\{(c_1, c_2, \dots, c_i) \mid \tau_i \text{ is schedulable by RMA}, \\ c_j \geq 0,\ j = 1, 2, \dots, i\big\}. \tag{8}$$

Definition 2 (Generalized bound) The generalized bound of task τ_i is an inequality $g_i(c_1, c_2, \dots, c_i) \leq 0$ such that:

- Any point in the region $\mathbf{F_i}$ satisfies $g_i(c_1, c_2, \dots, c_i) \leq 0$.
- Any point violates $g_i(c_1, c_2, \dots, c_i) > 0$ is not in the region $\mathbf{F_i}$.

2.1 Schedulability region

According to the Definition 1, region $\mathbf{F_i}$ which guarantees the schedulability of τ_i is obtained with the following theorem.

Theorem 2 *The schedulability region* $\mathbf{F_i}$ *for task* τ_i *is given by*:

$$\mathbf{F_i}(c_1, c_2, \dots, c_i) = \left\{(c_1, c_2, \dots, c_i) \bigvee_{t \in S_i} \sum_{j=1}^{i} \left\lceil \frac{t}{p_j} \right\rceil c_j - t \leq 0, \\ c_j \geq 0,\ j = 1, 2, \dots, i \right\} \tag{9}$$

where "$\bigvee$" denotes logic OR relationship.

Proof It directly follows from Eqs. 7 and 8 which defines F_i. □

It should be observed that Eq. 9 consist of a set of inequalities with logic OR relations. The question arises here is how to express such a list of inequalities with a single inequality. In the following, a mathematical transformation is proposed to remove logic OR relationships among the inequalities, which enables us to derive the generalized bound.

2.2 Construction of generalized bound

Lemma 1 [32] *Suppose* $g_1 \leq 0$, $g_2 \leq 0, \dots, g_m \leq 0$ *are constraints with logic OR relationships, then* $\forall j = 1, \dots, m$, $g_1 \leq 0 \vee g_2 \leq 0 \vee \cdots \vee g_m \leq 0$ *can be determined by* $(\Delta v - \sum_{j=1}^{m} (\sqrt{g_j^2} - g_j)) \leq 0$ *as a small positive value* $\Delta v \to 0$.

By Lemma 1, the constraints with logic OR relationships are turned into one general inequality.

Example 1 An equality $x \leq 3$ and inequalities $x \geq 5$ can be determined by

$$\Delta v - \left(\left(\sqrt{(x-3)^2} - (x-3)\right) + \left(\sqrt{(5-x)^2} - (5-x)\right)\right) \leq 0.$$

Note that in Lemma 1, $g_1 = 0 \vee g_2 = 0 \vee \cdots \vee g_m = 0$ is just barely determined by $(\Delta v - \sum_{j=1}^{m} (\sqrt{g_j^2} - g_j)) \leq 0$ because when $\Delta v \to 0^+$, we have that $\exists j, g_j \to 0^-$.

From Lemma 1, we have the following theorem.

Theorem 3 *The schedulability region* $\mathbf{F_i}$ *can be determined by*:

$$\begin{aligned}
&\mathbf{F_i}(c_1, c_2, \ldots, c_i) \\
&\quad = \left\{ (c_1, c_2, \ldots, c_i) \,\middle|\, \Delta v - \sum_{j=1}^{k_i} \left(\sqrt{\left(\sum_{m=1}^{i} \left\lceil \frac{S_{ij}}{p_m} \right\rceil c_m - S_{ij} \right)^2} \right.\right. \\
&\qquad \left.\left. - \left(\sum_{m=1}^{i} \left\lceil \frac{S_{ij}}{p_m} \right\rceil c_m - S_{ij} \right) \right) \leq 0,\ c_m \geq 0,\ m = 1, 2, \ldots, i \right\}
\end{aligned} \tag{10}$$

where S_{ij} *is the* jth *element of* S_i, k_i *is the number of elements in* S_i *and* $c_m \geq 0$ $(m = 1, 2, \ldots, i)$.

Proof It directly follows from Lemma 1 and Eq. 10 which defines $\mathbf{F_i}$. □

Theorem 4 *The generalized bound of a task* τ_i *can be determined by*

$$\Delta v - \sum_{j=1}^{k_i} \left(\sqrt{\left(\sum_{m=1}^{i} \left\lceil \frac{S_{ij}}{p_m} \right\rceil c_m - S_{ij} \right)^2} - \left(\sum_{m=1}^{i} \left\lceil \frac{S_{ij}}{p_m} \right\rceil c_m - S_{ij} \right) \right) \leq 0. \tag{11}$$

Proof It directly follows from Theorem 3 and Definition 2. □

Example 2 Consider a set of 3 tasks with periods $p_1 = 3,\ p_2 = 4$, and $p_3 = 5$. From Eq. 9, we have

$$\begin{aligned}
F_3 = \{ &(c_1, c_2, c_3) \mid c_1 + c_2 + c_3 \leq 3 \vee 2c_1 + c_2 + c_3 \leq 4 \vee 2c_1 + 2c_2 + c_3 \leq 5, \\
&c_j \geq 0,\ j = 1, 2, 3 \}.
\end{aligned}$$

The generalized bound of task τ_3 is given by

$$\begin{aligned}\Delta v &- \big(\sqrt{(c_1+c_2+c_3-3)^2}-(c_1+c_2+c_3-3)\\ &+\sqrt{(2c_1+c_2+c_3-4)^2}-(2c_1+c_2+c_3-4)\\ &+\sqrt{(2c_1+2c_2+c_3-5)^2}-(2c_1+2c_2+c_3-5)\big)\\ &\le 0.\end{aligned}$$

One advantage of using an inequality $g_i(c_1, c_2, \ldots, c_i) \le 0$ to express the task generalized bound is that other existing utilization bounds can be considered as a special case of the generalized bound, the other one is to develop a nonlinear programming formulation for solving *C-AP*.

2.3 Representation of the generalized bound

For better understanding and description, we show that the generalized bound can be represented as a polylinear surface in the task execution times space, denoted as the *c-space* from now on. In such a space, a point $c = \{c_1, c_2, \ldots, c_n\}$ represents a periodic task set whose tasks execution times are $c_1, c_2, \ldots,$ and c_n, respectively.

Lemma 2 *Given a line $g(x, y) = ax + by + c = 0$ in a 2-D space where $g(x, y)$ is an implicit function of $\mathbf{x} = (x, y) \in R^2$. The region below the line can be represented as $g(x, y) < 0$ and the region above the line can be represented as $g(x, y) > 0$ when $a > 0$ and $b > 0$.*

Proof Suppose a point $\mathbf{x} = (x_1, y_1)$ is on the line, then it satisfies $ax_1 + by_1 + c = 0$. Suppose a point $\mathbf{y} = (x_2, y_2)$ is below the line, that means $x_1 = x_2$, $y_1 > y_2$, then we have $ax_2 + by_2 + c = ax_1 + by_2 + c < ax_1 + by_1 + c = 0$. This proves that the region below the line can be represented as $g(x, y) < 0$ when $a > 0$ and $b > 0$. Similarly, we can prove that the region above the line can be represented as $g(x, y) > 0$ when $a > 0$ and $b > 0$. $\square$

The meaning of Lemma 2 is illustrated in Fig. 1: only one inequality can express three regions, (a) on the line, (b) over the line, and (c) below the line. In a broader

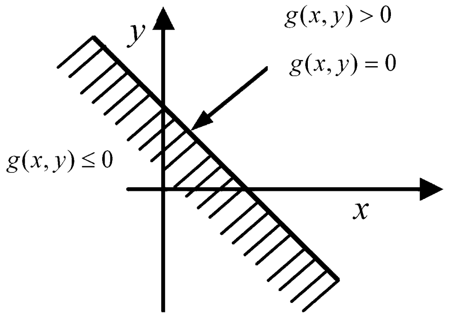


Fig. 1 Lemma 2 interpretation

perspective, it can be divided into two regions, Region-1: ($g(x, y) \geq 0$) and Region-2: ($g(x, y) < 0$). Since the feasibility problem has also two regions, i.e., feasible and infeasible, Lemma 2 can be easily extended to accommodate feasibility phenomena, which allows us to present the generalized bound of a task τ_i:

Theorem 5 *The generalized bound of task τ_i can be represented by a polylinear surface in the c-space.*

Proof The generalized bound of task τ_i is the bound of region $\mathbf{F_i}$ defined by Eq. 9. Note that all the schedulability constraints of task τ_i are linear. Since the task periods p_i are fixed, each constraint is represented by an i-dimensional plane (hyperplanes in higher dimensions) in the *c-space*. The collection of all the i-dimensional planes construct an i-dimensional polylinear surface intersecting each axis in 1. Points below the i-dimensional polylinear surface represent that task τ_i is RM feasible, whereas the region above the surface shows infeasible task τ_i. Hence, the generalized bound of task τ_i can be represented by a polylinear surface in the *c-space*. $\square$

In the following, we refer to the generalized bound of tasks as the polylinear bound or P-bound for short, as it can be represented by a polylinear surface in the *c-space*. It should be observed that the P-bound can also be represented as a polylinear surface in the task utilization space because c_i is proportional to U_i.

Considering Example 2, the region F_3 (Eq. 9) is delimited by three planes, which constructs a $3D$ polylinear surface in the *c-space*, as depicted in Fig. 2. Thus, the generalized bound of task τ_3 is represented as this $3D$ polylinear surface. Similarly, Fig. 3 shows the LL bound, the hyperbolic bound and the P-bound in the task utilization space for $n = 2$. The LL bound is represented by a line intersecting both axis in $U_{\text{lub}}(2) = 2(\sqrt{2} - 1)$. The hyperbolic bound is described as a hyperbola where the P-bound is a polyline. The hyperbolic bound and the P-bound intersect each axis

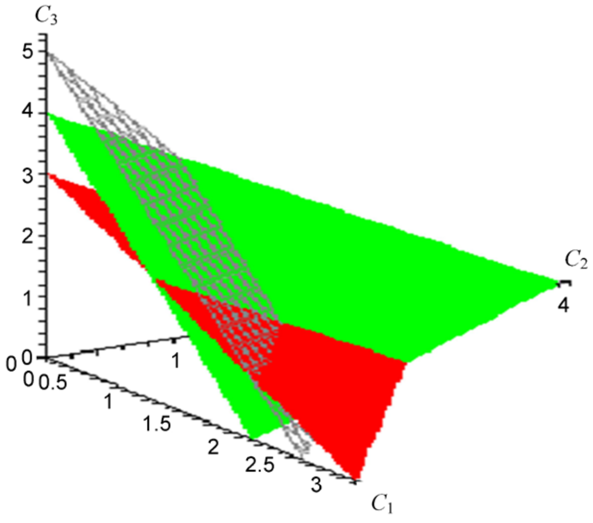


Fig. 2 Polylinear surface illustration for a task set with $p_1 = 3$, $p_2 = 4$, $p_3 = 5$

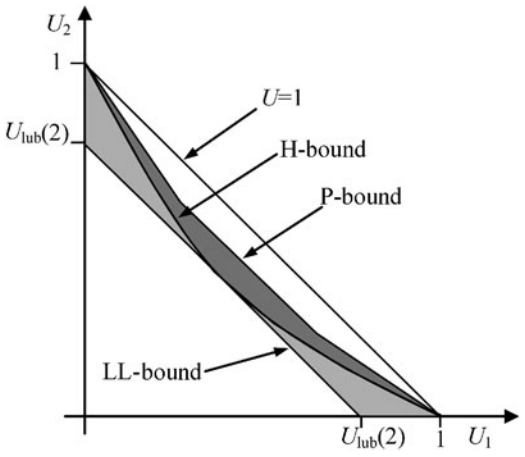


Fig. 3 Schedulability bounds for RM in the utilization space

in 1. The schedulability region below the P-bound is larger than that below the hyperbolic bound which is larger than that below the LL bound. It can be seen that none of RM-bounds (LL and H-bound)) can exceed the EDF-bound ($U = 1$ in Fig. 3) [22], which is understandable, as EDF is the optimal scheduling algorithm for any implicit deadline system. The figure is sketched for $n = 2$, it can be easily concluded that region below both the LL-bound and H-bound will reduce with larger n, however, P-bound (due to polylinear surface) in Fig. 3 would be definitely more advantageous over these solution, as it is exact condition.

3 Task execution times adjustment based on the P-bound

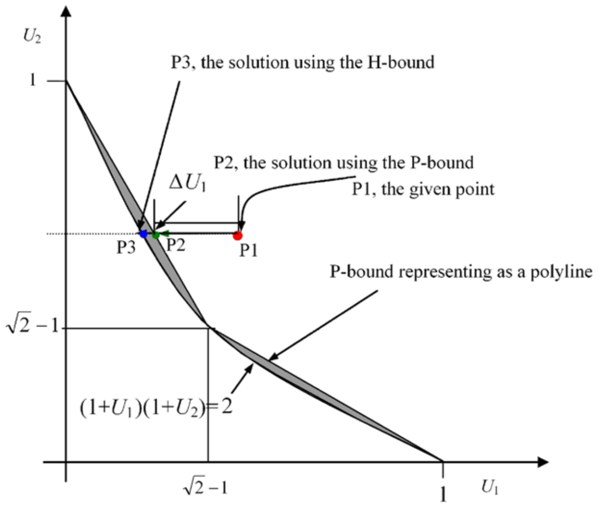
In this section, we show how the task execution times adjustment problem can be integrated with the P-bound. We assume that task periods are fixed and the tasks execution times can be well tuned to improve the system performance. In a very general framework, c_i is the design variable to represent the possible choices of the designer at the stage of the system design, the P-bound is the constraints to ensure the task schedulability, and the objective function is designed as a measure of the overall system performance. Consequently, C-AP can be formulated as nonlinear programming optimization problems, which enables us to determine the optimum c_i by applying the classical nonlinear optimization approaches.

3.1 Modifying a task

We are interested in knowing the amount of execution time variation Δc_k to task τ_k such that the generic task τ_i remains schedulable while maximizing the processor utility. Two situations arise here:

- If $i < k$, the task τ_i is not affected by any variation Δc_k because it has a higher priority than τ_k.

Fig. 4 Task execution time adjustment for $p_2 < 2p_1$



U_2
1
P3, the solution using the H-bound
P2, the solution using the P-bound
ΔU_1
P1, the given point
P3
P2
P1
P-bound representing as a polyline
$\sqrt{2}-1$
$(1+U_1)(1+U_2)=2$
$\sqrt{2}-1$
1
U_1

- If $i \geq k$, we obtain that

$$\begin{aligned}&\text{Maximize: } U_1 + U_2 + \cdots + U_i + \Delta c_k / p_k\\&\text{Subject to \{task } \tau_i \text{ is schedulable\}.}\end{aligned} \tag{12}$$

The objective function can be replaced by Δc_k since U_1, U_2, U_i, and p_k are all fixed. Further, applying the P-bound ensuring the schedulability of task τ_i. The above optimization problem can be written more formally as:

$$\text{Maximize } \Delta c_k \tag{13}$$

$$\Delta v - \sum_{j=1}^{k_i}\left(\sqrt{\left(\sum_{\substack{m=1\\m\neq k}}^{i}\left\lceil\frac{S_{ij}}{p_m}\right\rceil c_m + \left\lceil\frac{S_{ij}}{p_k}\right\rceil(c_k+\Delta c_k) - S_{ij}\right)^2}\right.$$
$$\left. - \left(\sum_{\substack{m=1\\m\neq k}}^{i}\left\lceil\frac{S_{ij}}{p_m}\right\rceil c_m + \left\lceil\frac{S_{ij}}{p_k}\right\rceil(c_k+\Delta c_k) - S_{ij}\right)\right) \leq 0. \tag{14}$$

In order to understand the formulation of P-bound for modifying one task, we use a simple example in the task *U-space* for illustration. In Fig. 4, assuming a task τ_2 is infeasible with (U_1, U_2) (representing as point P1 in the *U-space*). Our concern is to make task τ_2 schedulable by adjusting the execution time of task τ_1. The best solution is to move the given point P1 to the point P2 on the polyline where task τ_2 becomes schedulable. It is easy to see that the solution is better than that obtained using the H-bound, which is represented as point P3 in the *U-space*, i.e., ΔU_1 is the minimum distance from the given infeasible point to the P-bound, as $\Delta U_1 = \Delta c_1/p_1$.

Example 3 Consider task set $\Gamma = \{\tau_1, \tau_2\}$ with $c_1 = 15$, $p_1 = 30$ and $c_2 = 20$, $p_2 = 40$, where task τ_2 is unschedulable by RMA. By solving problem 13, we get $c_1 = 10$, which means that if the execution time of τ_1 is decreased from 15 to 10, task τ_2 becomes RM-feasible.

3.2 Scaling execution times of the task set

To reduce energy consumption of the real-time system, dynamic voltage scaling techniques [9] are being applied to real-time scheduling [25], where focus is made on lowering the system speed so that no deadline is ever missed. Suppose we have a schedulable task set and we want to compute the minimum processor speed, denoted by α, such that the task set is still schedulable at that speed α. The speed reduction can be considered as scaling the execution times c_i by a factor α. If we replace all the c_i with those obtained after a speed modification of α, the P-bound for τ_i becomes:

$$\Delta v - \sum_{j=1}^{k_i}\left(\sqrt{\left(\sum_{m=1}^{i}\left\lceil\frac{S_{ij}}{p_m}\right\rceil\frac{c_m}{\alpha} - S_{ij}\right)^2} - \left(\sum_{m=1}^{i}\left\lceil\frac{S_{ij}}{p_m}\right\rceil\frac{c_m}{\alpha} - S_{ij}\right)\right) \leq 0. \tag{15}$$

We can now find the minimum speed α, which is given by:

$$\text{Minimize } \alpha \tag{16}$$

$$\text{Subject to Inequality 15}, \quad \forall i = 1, \dots, n. \tag{17}$$

The constraints ensure that all the tasks are schedulable. The above optimization problem is also applicable to the cases when we have an unschedulable task set and we want to compute the minimum processor speed α, such that the task set becomes schedulable at that speed.

Example 4 Consider a set of four tasks with execution times $c_1 = 10$, $c_2 = 10$, $c_3 = 10$, and $c_4 = 10$, and periods $p_1 = 50$, $p_2 = 80$, $p_3 = 120$, and $p_4 = 200$. The task set is schedulable by RMA since $U_{\text{lub}} = 0.4583$ is less than the LL-bound. By solving the above problem, we get $c_1 = 20$, $c_2 = 20$, $c_3 = 20$, and $c_4 = 20$ and the total utilization equals 0.9166 at full speed. On the other hand, the original task set remains feasible even if the processor speed is reduced by half.

Example 5 Consider a set of four tasks with execution times $c_1 = 30$, $c_2 = 30$, $c_3 = 30$, and $c_4 = 30$, and periods $p_1 = 80$, $p_2 = 120$, $p_3 = 150$, and $p_4 = 210$. We have $U_{\text{lub}}(n) = 0.9679$. From LL-bound, the task set is unschedulable by RMA. By applying our technique, we get that a schedulable set when the processor speed is increased by a factor of $8/7$. In other words, the computation times are adjusted as $c_1 = c_2 = c_3 = c_4 = 26.25$ and the system schedulability is guaranteed.

3.3 Finding a generalized schedulable task set

Among many designing factors, the most vital issue is of getting exact value for how long the task will take at run-time. Considering many practical problems such as the source code, compiler optimization, system architecture, operating system, and so on, the *WCET* can only be estimated. To make the things worst, the time needed to find *WCET* of tasks depends also on the machine architectures which is again a complicated activity (interested reader is referred to [2, 19, 28]). The point that we are tying to make here is that even an estimate on *WCET* is tedious. Assuming a good estimate is made of the source code, there is still uncertainty involved at run time due to data cache, etc. Another extreme case would be the best case execution time (*BCET*). The difference between *BCET* and *WCET* can be as large as 80% [30]. The P-bound can be used in this situation, by first putting $c_i = BCET$ and then if the task τ_i is schedulable the value of c_i can be increased such that $c_i = WCET$. In case the task is infeasible, the code for τ_i can be readjusted and making *WCET* lowered accordingly. Sometimes, a task may have primary and alternative version of the same task. The primary version has a higher *WCET* than the alternative one. In such situations, completing either version results in the task being completed. Though the primary version offering high quality of service is preferred, alternative version of the task with acceptable quality may be executed when overload occurs. Another remedy to infeasibility is dividing c_i into two parts (i) Mandatory (m_i) and (ii) Optional (o_i) [28]. Tasks of these types are known as increased reward increased service (*IRIS*) tasks; the longer they run, the better would be their output results. Using the P-bound, initially put $c_i = m_i$ and if the task is still feasible, the value of c_i can be increased by adding o_i, until the system becomes barely schedulable, i.e., $c_i = m_i + x_i$, where $0 \le x_i \le o_i$.

Here, we extend the above conditions to a more general one by assuming that the execution times c_i $(i = 1, 2, \dots, n)$ vary within a range, i.e., they are described by $c_i^{\min} \le c_i \le c_i^{\max}$. The two issues namely, **P-1** and **P-2**, can be combined into a single optimization problem which can be formulated formally as follows: Given a set of n periodic tasks $\Gamma = \{\tau_1, \tau_2, \dots, \tau_n\}$ where the task execution time c_i varies in a range with a lower bound $c_i^{\min}$ and an upper bound $c_i^{\max}$, and p_i is known a priori, find a set of the execution times c_i under the RM schedulability constraints such that a system performance index is maximized. Suppose the task system is optimal in the sense that the total processor utilization is maximized. Thus, the extended problem can be expressed as a maximization problem:

$$\text{Maximize} \quad f(c_1, c_2, \dots, c_n) \tag{18}$$

$$\text{Subject to} \quad \text{Eq. 15} \quad \forall i = 1, \dots, n, \tag{19}$$

$$c_i^{\min} \le c_i \le c_i^{\max}. \tag{20}$$

In the following example, we show that the P-bound always produce higher task computation times than the LL-bound and H-bound.

Example 6 Suppose we have a system where the highest priority task τ_1 is an interrupt handler. τ_1 has a period $p_1 = 12$ and a worst-case execution time $c_1 = 1$. On top of this task, there are two tasks: the higher priority one has a period $p_2 = 30$. The other one has a period $p_3 = 50$. The two tasks are approximation algorithms. The longer they run the better are the results. The total approximation error of the two algorithms is given by

$$f(c_2, c_3) = p_2/c_2 + p_3/c_3 \tag{21}$$

which is then inversely proportional to the execution times c_i. The problem is to find the values of c_2 and c_3 such that the total error is minimized and all the tasks are schedulable.

The schedulability of τ_1 is ensured because $c_1 < p_1$.

The P-bound of τ_2 is given by

$$\begin{aligned}
&\Delta v - \big(\sqrt{(c_1 + c_2 - 12)^2} - (c_1 + c_2 - 12) \\
&\quad + \sqrt{(2c_1 + c_2 - 24)^2} - (2c_1 + c_2 - 24) \\
&\quad + \sqrt{(3c_1 + c_2 - 30)^2} - (3c_1 + c_2 - 30)\big) \\
&\leq 0.
\end{aligned} \tag{22}$$

Further, we have

$$\begin{aligned}
&\Delta v - \big(\sqrt{(c_2 - 11)^2} - (c_2 - 11) + \sqrt{(c_2 - 22)^2} - (c_2 - 22) \\
&\quad - \sqrt{(c_2 - 27)^2} - (c_2 - 27)\big) \\
&\leq 0.
\end{aligned} \tag{23}$$

The P-bound of τ_3 is given by

$$\begin{aligned}
&\Delta v - \big(\sqrt{(c_1 + c_2 + c_3 - 12)^2} - (c_1 + c_2 + c_3 - 12) \\
&\quad + \sqrt{(2c_1 + c_2 + c_3 - 24)^2} - (2c_1 + c_2 + c_3 - 24) \\
&\quad + \sqrt{(3c_1 + c_2 + c_3 - 30)^2} - (3c_1 + c_2 + c_3 - 30) \\
&\quad + \sqrt{(3c_1 + 2c_2 + c_3 - 36)^2} - (3c_1 + 2c_2 + c_3 - 36) \\
&\quad + \sqrt{(4c_1 + 2c_2 + c_3 - 48)^2} - (4c_1 + 2c_2 + c_3 - 48) \\
&\quad + \sqrt{(5c_1 + 2c_2 + c_3 - 50)^2} - (5c_1 + 2c_2 + c_3 - 50)\big) \\
&\leq 0.
\end{aligned} \tag{24}$$

Also, we have

$$\begin{aligned}
&\Delta v - \big(\sqrt{(c_2 + c_3 - 11)^2} - (c_2 + c_3 - 11) \\
&\quad + \sqrt{(c_2 + c_3 - 22)^2} - (c_2 + c_3 - 22) \\
&\quad + \sqrt{(c_2 + c_3 - 27)^2} - (c_2 + c_3 - 27) \\
&\quad + \sqrt{(2c_2 + c_3 - 33)^2} - (2c_2 + c_3 - 33) \\
&\quad + \sqrt{(2c_2 + c_3 - 44)^2} - (2c_2 + c_3 - 44) \\
&\quad + \sqrt{(2c_2 + c_3 - 45)^2} - (2c_2 + c_3 - 45)\big) \\
&\leq 0.
\end{aligned} \tag{25}$$

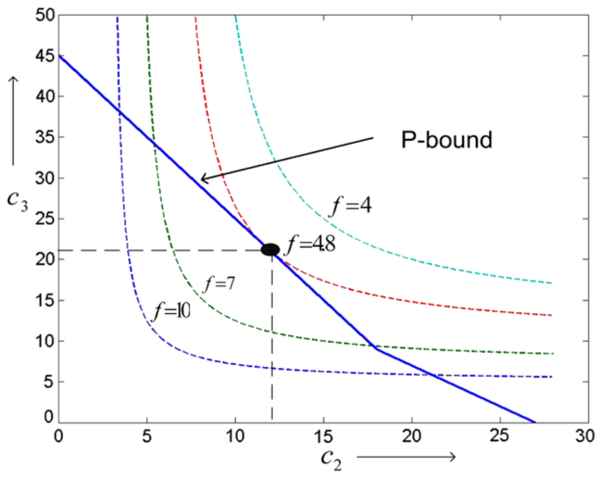


Fig. 5 Optimal computation times for the task set in Example 6

Therefore, the optimization problem can be expressed as follows:

$$\text{Minimize } f(c_2, c_3) = 30/c_2 + 50/c_3 \tag{26}$$

$$\text{Subject to Inequality 23 and 25} \tag{27}$$

All the points below the P-bound are feasible, however, the solution obtained through our formulation is optimal. Be solving problem 26, we get $c_2 = 11.7624$, $c_3 = 21.4751$ and $f(c_2, c_3) = 4.8788$. Figure 5 depicts the P-bound of τ_3 given by Eq. 25 in the *c-space*. The curves qualitatively show the cost function of Eq. 25, and the black dot is the solution to the problem 26. We now highlight the supremacy of our technique over existing bounds with the same example. It can be easily seen that the task is RM schedulable by P-bound with modified task computation times ($c_1 = 1$, $c_2 = 11.76$, and $c_3 = 21.47$). In contrast, both LL-bound and H-bound fail; computation times of tasks has to be further reduced to satisfy these bound. In other words,

$$\begin{aligned}
&\sum_{i=1}^{n} \frac{c_i}{p_i} \leq n\big(2^{1/n} - 1\big),\\
&\frac{1}{12} + \frac{11.76}{30} + \frac{21.47}{50} \leq 3\big(2^{1/3} - 1\big),\\
&0.904 \leq 0.779
\end{aligned} \tag{28}$$

which shows LL-bound fails with modified computation times derived by P-bound. Similarly,

$$\begin{aligned}
&\prod_{i=1}^{n}\left(1 + \frac{c_i}{p_i}\right) \leq 2,\\
&\left(1 + \frac{1}{12}\right) \times \left(1 + \frac{11.76}{30}\right) \times \left(1 + \frac{21.47}{50}\right) \leq 2,\\
&(2.13) \leq 2.
\end{aligned} \tag{29}$$

Again, H-bound also fails here.

Table 1 Improvement over LL-bound for $U_{\text{lub}} \leq n(2^{1/n} - 1)$

Number of tasks	Average processor utilization		
	U-LL (%)	U-PB (%)	Improvement (%)
3	77.98	86.38	8.40
5	74.35	81.35	7.00
10	71.77	78.19	6.42
20	70.53	76.18	5.65
30	70.12	76.08	5.96
40	69.92	75.88	6.08
50	69.80	75.88	6.08

4 Analysis of the P-bound

The complexity of the proposed technique is pseudo-polynomial in nature. For each task τ_i, there could be (p_i/p_1) operations involved and there are n number of tasks in total in the system. Thus, the time complexity of P-bound is $O(np_n/p_1)$, which is in agreement with necessary and sufficient condition [21].

We now compare our work with existing bounds in terms of system utilization and acceptance ratio. Results are obtained by evaluating bounds over synthetic task sets. Specifically, task periods p_i are randomly extracted from $[10, 10^6]$ with uniform distribution. Simulations runs are made for task sets with tasks $n = \{3, 5, 10, 20, 30, 40, 50\}$ in increasing order. For each task set of size n, 1,000 simulation runs are performed with different random task sets. The optimization problem is solved by the sequential quadratic programming algorithm [7]. Tasks execution times c_i are randomly extracted from $[0, p_i]$ with uniform distribution. In Table 1, we generated the task set with utilization less than or equal to $n(2^{1/n} - 1)$. The column heading U-LL means the system utilization achievable with LL-bound, while U-PB represents the utilization achievable through P-bound. It can be seen in Table 1 that the U-LL column lists descending values. This is due to the implicit behavior of the LL-bound, i.e., $U_{\text{lub}} \simeq \ln(2)$ for larger n. In comparison, the decrease in U-PB column is quite descent.

In Table 2, we generated a task set having a higher utilization $U_{\text{lub}} \simeq 1$. With such higher utilization, there is a high probability that the task set is RM-infeasible. In this case, either the computation time of tasks should be reduced or the system speed should be increased to make the set RM-feasible. We study the effect of speed in Table 2, by keeping the utilization intact, while the speed is increased so that the task set becomes RM-feasible. Let α_i be the associated speed for executing task τ_i. τ_i misses its deadline d_i running at α_i. As $c_{\alpha,i} = c_i/\alpha_i$, increasing α_i makes τ_i feasible. Table 2 indicates the appropriate speed for Γ in column α_r that produces RM-feasible set when executed with α_r, where $\alpha_r \geq \alpha_i$.

5 Comparison by acceptance ratio

The stage is now ready for evaluating our proposed technique. We judge the P-bound from the perspective of (i) the time it takes to produces the results and (ii) the accu-

Table 2 Improvement over LL-bound for $U_{\text{lub}} \leq 1$

Number of tasks	Average processor utilization			α_r
	U-LL (%)	U-PB (%)	Improvement (%)	
3	77.98	88.04	10.06	1.135
5	74.35	83.35	9.00	1.199
10	71.77	80.47	8.70	1.242
20	70.53	77.76	7.23	1.286
30	70.12	76.89	6.77	1.300
40	69.92	76.56	6.64	1.306
50	69.80	76.57	6.77	1.306

racy of the results produced. It can be easily concluded that both LL-bound and H-bound have polynomial complexity while the P-bound exhibits pseudo-polynomial complexity, as discussed earlier. Though the complexity of P-bound is higher, its practicability compensates the higher complexity very decently as the test is run at design time where accuracy is of higher importance than the complexity associated. The criteria we adapted here for determining the accuracy of generalized bound is the acceptance ratio; the number of tasks set with respect to those accepted by EDF test. Let X denotes a set of randomly generated tasks under the condition $U_{\text{lub}} \leq 1$, using the same criteria as discussed earlier in Sect. 4. Naturally, with this restraint on system utilization, the task set X is EDF-feasible. The RM-feasibility of the task set X is subject to the condition $U_{\text{lub}} \leq n(2^{1/n} - 1)$ or $\prod_{i=1}^{n}(U_i + 1) \leq 2$, however, for higher utilization, there exist a subset of X, which in RM-infeasible. We represent this set by X^f such that $X^f \subseteq X$. Let X^f_{LL}, X^f_{HB}, X^f_{PB} represent the task set which is RM-feasible by LL-bound, H-bound, and P-bound, respectively. For a particular utilization, the acceptance ratio is defined by

$$\text{Acceptance}(B) = \frac{X}{X^B} \tag{30}$$

where B represent a bound. The experimental results are drawn in Fig. 6, where RM-feasibility is influenced by the number of tasks, the decrease in acceptance ratio of a bound with increasing n is understandable. Both LL bound and H-bound are declining rapidly with n, as these are only sufficient conditions, however, generalized bound shows better results because it is both necessary and sufficient condition. It can be seen that $X^f_{\text{LL}} \subseteq X^f_{\text{HB}} \subseteq X^f_{\text{PB}}$, which is the intended purpose of this analysis; if any other test can determine the RM-feasibility of X, so will the generalized bound.

6 Conclusions

Considering the specific characteristics of the RM necessary and sufficient schedulability condition for a periodic task, which consists of a set of inequalities linked through logical or relationships. We have shown how to use one inequality to express

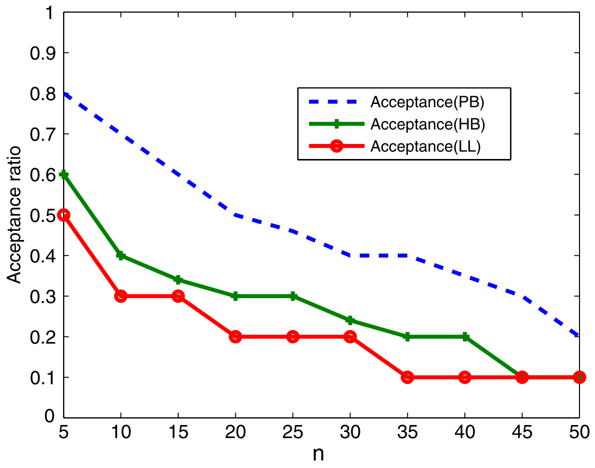


Fig. 6 Acceptance ratios of all bounds with respect to EDF-bound

the task schedulability constraint. The problem of determining the optimal tasks execution times in the space of RM schedulability is solved using optimization techniques. It is shown that the $C\text{-}AP$ is a natural extension of the RM schedulability test which typically assumes that the timing parameters are fixed. Computational results confirm that this new method is quite competitive. Our formulation improves the results obtained by any other bound such as the LL-bound. By applying the optimization technique, the P-bound founds the best design solution, even for large scale problems. Furthermore, one advantage of our work is that any system performance index can be designed as the objective functions of the nonlinear programming formulation.

Acknowledgements The authors would like to extend their thanks to the anonymous reviewers and Junxiang Liu for their valuable suggestions on the initial draft of this paper.

References

1. Akl SG (2004) Superlinear performance in real-time parallel computation. J Supercomput 29(1):89–111
2. Aydin H, Melhem RG, Mossé D, Mejía-Alvarez P (2001) Optimal reward-based scheduling for periodic real-time tasks. IEEE Trans Comput 50(2):111–130
3. Bhalla S (2004) Parallel concurrency control activity for transaction management in real-time database systems. J Supercomput 28(3):345–369
4. Bini E, Buttazzo GC, Buttazzo G (2003) Rate monotonic analysis: the hyperbolic bound. IEEE Trans Comput 52(7):933–942
5. Bini E, Buttazzo GC (2004) Schedulability analysis of periodic fixed priority systems. IEEE Trans Comput 53(11):1462–1473
6. Bini E, Natale MD, Buttazzo G (2008) Sensitivity analysis for fixed-priority real-time systems. Real-Time Syst 39(1–3):5–30
7. Boggs PT, Tolle JW (1995) Sequential quadratic programming. In: Acta numerica. Cambridge University Press, Cambridge, pp 1–51
8. Buttazzo GC (2005) Rate monotonic vs. edf: judgment day. Real-Time Syst 29(1):5–26
9. Chandrakasan AP, Brodersen RW (1995) Low power design. Kluwer Academic, Dordrecht
10. Colom PM (2002) Analysis and design of real-time control systems with varying control timing constraints. PhD dissertation, Automàtica i Informàtica Industrial, Barcelona

11. Davis R, Zabos A, Burns A (2008) Efficient exact schedulability tests for fixed priority real-time systems. IEEE Trans Comput 57:1261–1276
12. George L, Riverre N, Spuri M (1996) Preemptive and non-preemptive real-time uniprocessor scheduling. INRIA, France, Tech Rep 2966
13. Ha HCR, Liu JWS (2003) Experimental analysis of timing validation methods for distributed real-time systems. J Supercomput 25(1):73–94
14. Harbour MG, García JG, Gutiérrez JP, Moyano JD (2001) Mast: modeling and analysis suite for real time applications. In: Proceedings of the 13th Euromicro conference on real-time systems. IEEE Computer Society, New York, p 0125
15. Mast: modeling and analysis suite for real-time applications. University of Cantabria, Tech Rep 2010. http://mast.unican.es/
16. Izadi BA, Özgüner F (2004) An augmented *k*-ary tree multiprocessor with real-time fault-tolerant capability. J Supercomput 27(1):5–17
17. Jia W, Han B, Zhang C, Zhou W (2004) Delay control and parallel admission algorithms for real-time anycast flow. J Supercomput 29(2):197–209
18. Kaouane L, Akil M, Grandpierre T, Sorel Y (2004) A methodology to implement real-time applications onto reconfigurable circuits. J Supercomput 30(3):283–301
19. Krishna CM, Shin KG (1997) Real time systems, vol 1. McGraw-Hill, New York
20. Lee C, Sha L, Peddi A (2004) Enhanced utilization bounds for qos management. IEEE Trans Comput 53(2):187–200
21. Lehoczky JP, Sha L, Ding Y (1989) The rate monotonic scheduling algorithm: exact characterization and average case behavior. In: Proceedings of the IEEE real-time system symposium, pp 166–171
22. Liu CL, Layland JW (1973) Scheduling algorithms for multiprogramming in a hard real-time environment. J ACM 20(1):40–61
23. Min-Allah N, Ali I, Xing J, Wang Y (2010) Utilization bound for periodic task set with composite deadline. J Comput Electr Eng. doi:10.1016/j.compeleceng.2010.04.003
24. Min-Allah N, Yong-Ji W, Jian-Sheng X, Jiu-Xiang L (2007) Revisiting fixed priority techniques. In Embedded and ubiquitous computing, LNCS, vol. 4808. Springer, Berlin, pp 134–145
25. Pillai P, Shin KG (2001) Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proceedings of 18th ACM symposium on operating systems principles, pp 89–102
26. Puschner P, Koza C (1989) Calculating the maximum execution time of real-time programs. Real-Time Syst 1(2):159–176
27. Ramamritham K (1996) Where do time constraints come from and where do they go? International Journal of Database Management 7(2)
28. Shih WK, Liu JWS, Chung JY (1991) Algorithm for scheduling tasks to minimize total error. SIAM J Comput 20:537–552
29. Stankovic JA, Spuri M, Ramamritham K, Buttazzo GC (1998) In: Stankovic JA (ed) Deadline scheduling for real-time systems: EDF and related algorithms. Kluwer Academic, Dordrecht
30. Wegener J, Mueller F (2001) A comparison of static analysis and evolutionary testing for the verification of timing constraints. Real-Time Syst 21(3):241–268
31. Yongji W, Cartmell M, Tao Q, Liu H (2005) A generalized real-time obstacle avoidance method without the c-space execution. J Comput Sci Technol 20(6):774–787
32. Yongji W, Lane DM (2002) Solving a generalized constrained optimization problem with both logic and or relationships by a mathematical transformation and its application to robot path planning. IEEE Trans Syst Man Cybern, Part C, Appl Rev 30(4):525–536