情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-AL-124 No.9
2009/5/11

リアルタイムシステムの固定優先度スケジューリングに対する優先度周期探索法

橋 本 英 樹[†1]　柳 浦 睦 憲[†2]

リアルタイムシステムでは, 複数のタスクが繰り返し実行されており, 任意の時刻に発生する入力は対応する一連のタスク (パス) により処理され出力される. 入出力間の応答時間には各パスごとに要求される上限がある. 本研究ではそのようなリアルタイムシステムに対して, 応答時間の要求違反度及びシステム負荷の最小化を目的とした, タスクの固定優先度スケジューリングを設計する. 提案するアルゴリズムでは, 各タスクに対する優先度を局所探索法により探索し, 各タスクの周期は優先度から算出する. この各タスクの優先度から周期を求める部分問題に対し, 線形計画法に基づく解法を提案する. 最後に, 企業から提供された問題例に対する実験結果を報告する.

An LP-Based Algorithm for Scheduling Preemptive and/or Non-preemptive Real-time Tasks

Hideki Hashimoto[†1] and Mutsunori Yagiura[†2]

We consider a real-time system that requires the time stimuli to the system are processed through sequences of tasks to be within specified upper bounds, where the set of tasks can be the mixture of preemptive and non-preemptive tasks, and we propose an algorithm to design a static priority scheduling for the system. In the algorithm, local search is used to determine priorities of tasks, and whenever the priorities are fixed, the periods of tasks are determined. This subproblem can be described as a mathematical programming formulation and is solved via linear programming techniques. Finally, we report computational results for sample instances from a company.

†1 中央大学
Chuo University
†2 名古屋大学
Nagoya University

1. Introduction

A real-time system can be described as a computing system designed for controlling some technical facilities that have constraints on the time required from an event to the system response[2),4)]. Such systems are incorporated in various devices or machines such as mobile phones, information appliances and automobiles, and their importance becomes increasingly large.

In the literature of scheduling for a real-time system, a system is usually categorized into *preemptive* and *non-preemptive* systems and, for scheduling methods, *static priority* and *dynamic priority scheduling* are often adopted[6)]. In a preemptive system, the processing of a task can be interrupted by other tasks, while interruption is not allowed in a non-preemptive system. In a static priority scheduling, each task is assigned a static priority and a period. A task is invoked once in every period and becomes ready to execute. Then a task with the highest priority among ready tasks is chosen to be executed. In the rate monotonic scheduling, priorities are assigned in the ascending order of periods[8)]. On the other hand, in a dynamic priority scheduling, priorities of tasks are assigned dynamically during the process: For example, a task with the earliest deadline among ready tasks has the highest priority in the earliest deadline scheduling.

In real situations, a task denotes a function of the system and a stimulus to the system is processed by a sequence of tasks, which we call a *path*. A *response time of a path* is defined as the maximum time required to process the stimulus through the path. It is often required that the response time of each path should be bounded by some value determined by the requirement of applications. To our knowledge, however, none of the existing research papers explicitly treated such constraints on paths.

In this paper, we propose an algorithm to design a static priority scheduling for a real-time system having the path requirements, where the set of tasks can be the mixture of preemptive and non-preemptive ones. We consider a path-period condition, which is a sufficient condition for a path requirement, since it is difficult to treat path constraints directly. Then, a necessary and sufficient condition for a static priority scheduling to be schedulable is derived. We formulate the problem of determining periods for a given set of priorities of tasks and propose an algorithm to solve the problem by using linear programming techniques[5]. The obtained priorities and periods are schedulable but may violate path requirements to some extent if they are difficult to be satisfied. Finally, we propose an iterated local search algorithm for searching task priorities and report computational results for sample instances provided from a company.

2. Model of a real-time system

In this section, we describe a model of a real-time system. Let $\mathcal{T} = \{1, 2, \ldots, n\}$ be a set of tasks. The task set $\mathcal{T}$ is partitioned into a set $\mathcal{T}^{\mathrm{pre}}$ of preemptive tasks and a set $\mathcal{T}^{\mathrm{non}}$ of non-preemptive tasks. Let $\mathcal{P} = \{1, 2, \ldots, m\}$ be a set of paths, where a path $p \in \mathcal{P}$ is a sequence of tasks. We denote by $p(h)$ the hth task in path p and by l_p the number of tasks in p (duplication of counting is allowed). Each task $i \in \mathcal{T}$ and each path $p \in \mathcal{P}$ are associated with an execution time c_i and a permissible delay θ_p.

Let $S_{\mathrm{proc}} : \Re \to \{0\} \cup \mathcal{T}$ be a function representing the task processed at each time in the system, where $S_{\mathrm{proc}}(t)$ is 0 if none of the tasks is processed at time t, otherwise the index of the processed task. Let $S_{\mathrm{start}} : \Re \to \{0\} \cup \mathcal{T}$ be $i \in \mathcal{T}$ if an execution of task i starts at time t, and 0 otherwise. Note that if the execution of an interrupted task resumes at time t, $S_{\mathrm{start}}(t) = 0$. We denote by $S = (S_{\mathrm{proc}}, S_{\mathrm{start}})$ a schedule of the system. If a stimulus occurs at time t, it is processed by the tasks in a path p in the order of p along the schedule S from t. Let $\phi^p_h(t)$ (resp., $\psi^p_h(t)$) be the start (resp., completion) time of the hth task in p when the execution processes a stimulus occurs at time t, i.e.,

$$\psi^p_h(t) = \min\left\{ t' \geq \phi^p_h(t) \;\middle|\; \int_{\phi^p_h(t)}^{t'} \delta_{p(h)}(\tau)\mathrm{d}\tau \geq c_{p(h)} \right\}$$

$$\phi^p_h(t) = \begin{cases} \min\{t' \geq t \mid S_{\mathrm{start}}(t') = p(h)\}, & \text{if } h = 1 \\ \min\{t' \geq \psi^p_{h-1}(t) \mid S_{\mathrm{start}}(t') = p(h)\}, & \text{otherwise,} \end{cases}$$

where

$$\delta_i(t) = \begin{cases} 1, & \text{if } S_{\mathrm{proc}}(t) = i \\ 0, & \text{otherwise.} \end{cases}$$

For convenience, we define $\psi^p_0(t) = t$. We also define $L^p_h(t) = \psi^p_h(t) - \psi^p_{h-1}(t)$ and call it *latency*, the time it takes for task $p(h)$ after the completion of $p(h)$. Let $r_p(t)$ be the sum of the latencies

$$r_p(t) = \sum_{h=1}^{l_p} (\psi^p_h(t) - \psi^p_{h-1}(t)) = \psi^p_{l_p}(t) - t,$$

i.e., the time required to process a stimulus through path p. Then a *path requirement* is defined to be

$$\max_t r_p(t) \leq \theta_p.$$

In this paper, however, path requirements can be violated if they are difficult to be satisfied (i.e., they are considered as soft constraints).

A static priority scheduling is specified by periods, priorities and offsets of tasks. Let T_i be the period of a task i, where task i is invoked once in every T_i. Let σ denote a static priority, where $\sigma(i)$ denotes the priority of task i and task i has a higher priority than that of j if $\sigma(i) < \sigma(j)$. An instant at which a task is first invoked is called the *offset* of the task. In the static priority scheduling, an extended task is switched when its execution is completed or the processed task is preemptive and a higher priority task becomes ready. A static priority scheduling is called *schedulable* if each task is invoked at the beginning of every period and the execution is completed in the period for any offsets.

Figure 1 shows an example of a static priority scheduling, where $\mathcal{T} = \{1, 2, 3\}$, $c_1 = c_2 = 1, c_3 = 2$, $T_1 = 4$, $T_2 = 5$, $T_3 = 6$, $\sigma = (1, 2, 3)$ and the offsets are all zero. In the figure, an upward arrow represents an invocations of a task and a rectangle represents an execution of a task. For example, a stimulus at at time 8 (resp., 12) is processed through diagonally striped (resp., shaded) tasks of path $(1, 2, 3)$ and is completed at time 15 (resp., 20), and $\lim_{\epsilon \to +0} r_{(1,2,3)}(8 + \epsilon) = 12$.

3. Scheduling

In this section, we describe two conditions which our algorithm is based on: a path-period condition, which is a sufficient condition for the path requirements, and a necessary and sufficient condition for a static priority scheduling to be schedulable.

Although a path requirement is difficult to be certified without simulating the sched-

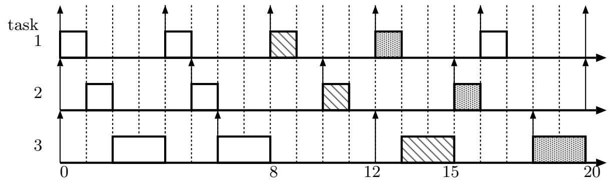

Fig. 1 An example of schedule

ule, the following path-period condition gives us a sufficient condition which is easy to check.

Theorem 3.1. *For a periodic scheduling S, the condition*

$$\sum_{h=1}^{l_p} 2T_{p(h)} \le \theta_p,$$

which we call a path-period condition, is a sufficient condition for a path requirement $\max_t r_p(t) \le \theta_p$ to be satisfied.

Proof. We prove that $r_p(t) \le \sum_{h=1}^{l_p} 2T_{p(h)}$ holds for any t.

Let us consider the latency $L_h^p(t)$ of the hth task in p. The latency $L_h^p(t)$ is no more than $2T_{p(h)}$, i.e., twice the period, because for any t', the hth task must be invoked and its execution must be completed in the time span $[t', t' + 2T_{p(h)}]$. Hence $r_p(t) = \sum_{h=1}^{l_p} L_h^p(t) \le \sum_{h=1}^{l_p} 2T_{p(h)}$ holds. □

Schedulability conditions for a static priority scheduling has been investigated for a preemptive system and a non-preemptive system, respectively[6)]. However, for a system with both preemptive and non-preemptive tasks, such a condition has not been clearly stated in the literature to the best of our knowledge. The following theorem enables us to determine the schedulability by checking the condition in pseudo polynomial time as is the case of the known schedulability conditions.

Theorem 3.2. *A set $\mathcal{T}$ of tasks with execution times c_i $(i \in \mathcal{T})$ is schedulable by a static priority scheduling with a priority σ and periods T_i $(i \in \mathcal{T})$ if and only if there exist t_i for all $i \in \mathcal{T}$ satisfying the following inequalities:*

$$c_i \le t_i \le T_i, \quad \forall i \in \mathcal{T} \tag{1}$$

$$\max_{i<j, j \in \mathcal{T}^{\mathrm{non}}} c_j + \sum_{j=1}^{i-1} c_j \left\lceil \frac{t_i - c_i}{T_j} \right\rceil \le t_i - c_i, \quad \forall i \in \mathcal{T}^{\mathrm{non}} \tag{2}$$

$$\max_{i<j, j \in \mathcal{T}^{\mathrm{non}}} c_j + c_i + \sum_{j=1}^{i-1} c_j \left\lceil \frac{t_i}{T_j} \right\rceil \le t_i, \quad \forall i \in \mathcal{T}^{\mathrm{pre}}, \tag{3}$$

where it is assumed, without loss of generality, that the indices of the tasks in $\mathcal{T}$ are ordered by the priority (i.e., $\sigma(i) = i$, $\forall i$).

Proof. Let us consider a *response time for a task*, which is the time from the moment the task is invoked until its execution is completed. In the literature, a *critical instant* for a task is an instant when the maximum response time for the task is realized among any combination of offsets. If the response time for the task is within its period, the task can be scheduled for any offsets, and vice versa.

Here we consider a critical instant for a task i, and we assume that i at the critical instant is invoked immediately after time x_0 so that an execution of a lower priority task can be started at x_0 and that the execution of i is completed at time x_1. From the maximality of a critical instant, neither a higher priority task nor a lower priority non-preemptive task can be processed before x_0, because if such a task exists, we can take a smaller value as x_0. In the time span $[x_0, x_1]$, tasks that can be processed are i, the higher priority tasks and one lower priority non-preemptive task. We consider an upper bound on the execution time spent for each of such tasks in the time span. The task i and one lower priority non-preemptive task can be executed only once and the upper bounds are c_i and $\max_{i<j, j \in \mathcal{T}^{\mathrm{non}}} c_j$, respectively. For a higher priority task k, the time at which k is first invoked is not earlier than x_0 and the upper bound is $c_k \lceil (x_1 - x_0)/T_k \rceil$ if i is preemptive and $c_k \lceil (x_1 - x_0 - c_i)/T_k \rceil$ if i is non-preemptive. Hence, if the inequalities (1)–(3) in the statement are satisfied, the response times for all tasks are less than or equal to their periods, respectively, and the static priority scheduling is schedulable. On the other hand, each upper bound on the response time of the task are realized when the offsets of all higher priority tasks are zero and the offset of the lower priority non-preemptive task is a very little less value than 0, and

the inequalities (1)–(3) must hold if the static priority scheduling is schedulable. □

4. Design of the real-time system

In this section, we propose an algorithm to design a static priority scheduling for the real-time system. We discuss the evaluation of priority σ in Section 4.1, and then, we describe the local search used to determine priorities of tasks in Section 4.2.

4.1 Evaluation of priority $\boldsymbol{\sigma}$

In this subsection, we propose an algorithm to determine periods of tasks for a static priority σ. For convenience, in this subsection, we assume $\sigma = (1, \ldots, n)$.

The problem of asking the periods of tasks so that the schedulability condition for the static priority scheduling is satisfied and the violated amount (measured by the ratio to the upper bound) of path-period conditions is minimized is formally described as follows:

$$\begin{aligned}
\min\ & \sum_{i=1}^{n} \frac{c_i}{T_i} + \lambda \\
\text{s.t.}\ & \max_{i<j, j\in\mathcal{T}^{\text{non}}} c_j + \sum_{j=1}^{i-1} c_j \left\lceil \frac{t_i - c_i}{T_j} \right\rceil \le t_i - c_i, \quad \forall i \in \mathcal{T}^{\text{non}} \\
& \max_{i<j, j\in\mathcal{T}^{\text{non}}} c_j + c_i + \sum_{j=1}^{i-1} c_j \left\lceil \frac{t_i}{T_j} \right\rceil \le t_i, \quad \forall i \in \mathcal{T}^{\text{pre}} \\
& \sum_{h=1}^{l_p} 2T_{p(h)} \le (1+\lambda)\theta_p, \quad \forall p \in \mathcal{P} \\
& \lambda \ge 0 \\
& c_i \le t_i \le T_i, \quad \forall i \in \mathcal{T}.
\end{aligned}$$

It is not easy to tackle this formulation directly, because there exist the ceiling terms. Hence, we remove these terms from the formulation by replacing a term $\lceil x \rceil$ with $x+1$. Then the following restricted formulation is derived:

$$\begin{aligned}
\min\ & \sum_{i=1}^{n} \frac{c_i}{T_i} + \lambda \\
\text{s.t.}\ & \max_{i<j, j\in\mathcal{T}^{\text{non}}} c_j + \sum_{j=1}^{i-1} c_j \left(\frac{t_i - c_i}{T_j} + 1 \right) \le t_i - c_i, \quad \forall i \in \mathcal{T}^{\text{non}} \\
& \max_{i<j, j\in\mathcal{T}^{\text{non}}} c_j + c_i + \sum_{j=1}^{i-1} c_j \left(\frac{t_i}{T_j} + 1 \right) \le t_i, \quad \forall i \in \mathcal{T}^{\text{pre}} \\
& \sum_{h=1}^{l_p} 2T_{p(h)} \le (1+\lambda)\theta_p, \quad \forall p \in \mathcal{P} \\
& \lambda \ge 0 \\
& c_i \le t_i \le T_i, \quad \forall i \in \mathcal{T}.
\end{aligned}$$

Note that, by this restriction, some solutions are rejected even if they satisfy the original schedulability condition. The inequalities of the revised formulation can be viewed as linear constraints of variables T_i, $1/T_i$, $1/t_i$, $1/(t_i - c_i)$, and λ. The only remaining constraints are the inverse relations of T_i and $1/T_i$, i.e., $T_i(1/T_i) = 1$. In this revised formulation, they can be replaced by inequalities $T_i \cdot (1/T_i) \ge 1$ and the problem turns out to be a convex nonlinear optimization problem[3].

The inverse inequality $x \cdot y \ge 1$ can be treated by infinite linear inequalities, e.g., $x + \mu^2 y \ge 2\mu, \forall \mu \ge 0$. We solve the problem via linear programming imposing the infinite linear inequalities implicitly. In order to achieve this, we initially solve the problem ignoring the inverse inequalities, and while the obtained solution violates some inverse inequalities we repeat solving linear programming problems by adding those constraints that eliminate the current solution. Note that the problem can also be solved by semidefinite programming[3] by expressing the inverse inequalities as Shur complements[7].

4.2 Local search for $\boldsymbol{\sigma}$

The search space of the proposed local search (LS) is the set of all permutations $\boldsymbol{\sigma}$[1]. The LS starts from an initial solution $\boldsymbol{\sigma}$ and repeats replacing $\boldsymbol{\sigma}$ with a better solution in its neighborhood $N(\boldsymbol{\sigma})$ until no better solution is found in $N(\boldsymbol{\sigma})$. We use the swap neighborhood, which is one of the representative neighborhoods for permutations. A swap operation of priorities j and k exchanges the priorities of the two tasks:

$$\sigma'(i) = \begin{cases} \sigma(k), & \text{if } i = j \\ \sigma(j), & \text{if } i = k \\ \sigma(i), & \text{otherwise,} \end{cases}$$

where σ' is the priority after applying the swap operation.

Because only one iteration of LS may not be sufficient to find a good solution, we use the iterated local search[9], which iterates LS many times from initial solutions generated by perturbing good solutions obtained so far. As a perturbation, we use a random 3-cyclic exchange operation:

$$\sigma'(i) = \begin{cases} \sigma(l), & \text{if } i = j \\ \sigma(j), & \text{if } i = k \\ \sigma(k), & \text{if } i = l \\ \sigma(i), & \text{otherwise,} \end{cases}$$

where j, k and l are randomly selected and σ' is the priority after applying the perturbation. Note that a 3-cyclic exchange operation cannot be attained by one swap operation and it prevents LS from returning right back to the initial solution.

5. Experiments

We conduct computational experiments on sample instances provided from a company. The algorithm was coded in the C language and run on a PC (Intel Core2 Duo CPU E6750, 2.66GHz, 2 GB memory). Linear programming problems are solved by using GLPK 4.33. [⋆1] Furthermore, in order to evaluate the obtained schedules, we simulate them for the time span $[0, L]$, where we set $L = \alpha \max_{p \in \mathcal{P}} \sum_{h=1}^{l_p} 2T_{p(h)}$ with $\alpha = 4$ in the experiments. With this setting, each path is executed at least α times.

In Table 1, the computational results for 11 instances are shown. The time limit of our algorithm is 300 seconds for each instance. Column "instance" denotes the instance name, column "util" denotes the utility $\sum_{i=1}^{n} c_i/T_i$ of the CPU by the obtained schedule and column "λ" denotes the violated amount for the path-period conditions. Column "r/θ" denotes the worst ratio $\max_{p \in \mathcal{P}} \max_{0 \le t \le L} r_p(t)/\theta_p$ obtained by simulating the schedule (i.e., the priorities and the periods). Note that, if a value of λ is zero, the path-period conditions (and hence the path requirements) are satisfied, and if a value of r/θ is less than one, the path requirements are satisfied at least in the span the schedules are simulated. From Table 1, we can observe that the obtained schedules satisfy the path requirements for instances f200, f500, f800 and f1000, and for instance n7, the path requirements are satisfied at least in the simulated span. For the other instances, though the path requirements are violated, the utilities of the obtained schedules are at most 0.79. This is attributed to the limitation of the static priority scheduling ability.

Table 1 Computational results on sample instances from a company

instance	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{T}^{\mathrm{non}}\rvert$	$\lvert\mathcal{P}\rvert$	util	λ	r/θ
n7	7	7	4	0.72	0.31	0.62
n18	18	18	10	0.79	2.41	1.48
n23	23	23	13	0.78	2.35	1.52
n200	200	200	2942	0.75	4.22	2.48
n500	500	500	26905	0.73	4.32	2.40
n800	800	800	46126	0.73	4.37	2.50
n1000	1000	1000	45314	0.73	4.31	2.46
f200	200	95	2942	0.71	0.00	0.46
f500	500	237	26905	0.71	0.00	0.46
f800	800	385	46126	0.73	0.00	0.47
f1000	1000	494	45314	0.73	0.00	0.48

6. Conclusion

We considered a real-time system that requires a bound on the time a stimulus to the system is processed by a sequence of tasks, and propose an algorithm that can treat such path requirements, where the mixture of preemptive and non-preemptive tasks was considered. In our algorithm to design a static priority scheduling for the system, local search is used to search priorities of tasks, and after fixing the priority, the periods of tasks are determined. This subproblem is described as a mathematical programming formulation based on path-period conditions and a schedulability condition for

[⋆1] http://www.gnu.org/software/glpk/

the static priority scheduling, and is solved by using linear programming techniques. Finally, we reported the computational results on sample instances from a company.

Acknowledgment

The authors would like to thank Taichiroh Sugiyama for helping us to simulate the solutions.

References

1) Aarts, E. H.L. and Lenstra, J.K.(eds.): *Local Search in Combinatorial Optimization*, John Wiley and Sons (1997).
2) Baruah, S. and Goossens, J.: Scheduling Real-Time Tasks: Algorithms and Complexity, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis* (Leung, J. Y.-T., ed.), Chapman Hall/CRC Press, chapter28 (2004).
3) Ben-Tal, A. and Nemirovski, A.: *Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications*, Society for Industrial and Applied Mathematics (2001).
4) Błażewicz, J., Ecker, K.H., Pesch, E., Schmidt, G. and Węglarz, J.: Scheduling in Hard Real-Time Systems, *Handbook on Scheduling: From Theory to Applications*, Springer, pp.243–269 (2007).
5) Chvátal, V.: *Linear Programming*, W. H. Freeman (1983).
6) George, L., Rivierre, N. and Spuri, M.: Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling, Technical report, Institut national de recherche en informatique et en automatique (1996).
7) Horn, R. A. and Johnson, C. R.: *Matrix Analysis*, Cambridge University Press (1985).
8) Liu, C.L. and Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the ACM*, Vol.20, No.1, pp.46–61 (1973).
9) Lourenço, H.R., Martin, O.C. and Stützle, T.: Iterated Local Search, *Handbook of Metaheuristics* (Glover, F. and Kochenberger, G. A., eds.), Kluwer Academic Publishers, pp.321–353 (2003).