Adaptive Resource Allocation Control with On-Line Search for Fair QoS Level

Fumiko Harada, Toshimitsu Ushio,
Graduate School of Engineering Science
Osaka University
{harada@hopf., ushio@}sys.es.osaka-u.ac.jp

Yukikazu Nakamoto
NEC System Platforms Research Laboratories
nakamoto@ct.jp.nec.com

Abstract

Recently, applications of control theory to avoid overload conditions have been studied. But, in conventional feedback theory, control input is determined by an error between a reference value of a controlled signal and its current value. In real-time systems, a set of active tasks may be time-varying so that the reference value may change according to the set. In this paper, we proposed a novel control method for a fair resource allocation and QoS levels of tasks are used as controlled signals. The proposed adaptation controller allocates a CPU utilization factor to each task with on-line search for the fair QoS level, and the proposed control rule is based on errors between the current QoS levels and their average. Its computation time is small so that it does not yield a heavy overhead.

1. Introduction

In real-time systems, overload conditions bring up the significant decrease of system response predictability and performance[1]. To avoid the overload conditions in the real-time systems, admission controls are utilized in which a guarantee test is executed whether a task execution is allowed at its arrival. Recently methodologies are proposed where CPU utilization factors are decreased according to a QoS(quality of service) level allocated to each task to avoid the overload conditions.

Flexible applications exist in which QoS is improved as long as its computation time and resources increase[2]. In flexible applications, the QoS level indicates a satisfaction level of users for an execution result of released jobs. As an example of flexible application, there is an MPEG video playback application. In an MPEG application, I-frame playbacks, B-frame and P-frame playbacks are demanded. B-frame and P-frame playbacks handle differential frames with I-frame. As CPU time and resources for B-frame and P-frame playbacks increase, more precise images in the playback are produced. When each flexible application in a system increases the QoS level simultaneously, however, the system becomes overload conditions. To avoid the conditions, arbitrating the QoS levels of the competitive applications are required.

There are several researches to arbitrate the competitive application or tasks in real-time systems. Abdelzaher *et al.* have proposed a method where the acceptable QoS levels of tasks in overload conditions are described *a priori* and the system degrades the QoS levels of the tasks based on the QoS level description at overload conditions[3]. However, this study does not consider dynamic negotiation of QoS levels of tasks. Buttazzo *et al.* have introduced an elastic task execution model where task executions can be expanded and shrunk like an elastic spring and proposed an algorithm to degrade the task execution time to handle overload conditions[4, 5]. A software designer must model application programs as the elastic model. Rajkumar *et al.* have proposed the QoS-based resource allocation model to solve problems when applications in real-time systems have simultaneous access to multiple resource[6]. In their works, they obtained conditions where overall system utility is maximized under the constraint that each application can meet its minimum needs. Oikawa and Rajkumar have proposed software architecture and module APIs to manage resource reservation for timely guaranteed behavior of real-time programs and developed the portable kernel module based on the APIs[7]. The kernel module provides an admission control function to manage the resource reservation.

Recently, much attention has been paid to applications of the control theory to real-time systems. The control theory is applied to manage CPU usage in real-time systems in [8]. In this study, a PID(Proportional-integral-derivative) controller is used to control CPU utilization time. However, conditions of the PID controller, under which a system status is transited to an expected one, are not obtained theoretically. To maintain CPU utilization factors and deadline

miss ratios with specified values, the feedback control theories are utilized. For example, in [9], given relative reference values of task's deadline, deadline miss ratios are controlled within specified certain ranges by reserving CPU times with a PID controller based on the difference between the reference values and actual deadlines. Lu *et al*. have proposed a gain feedback control method with given reference values for CPU utilization factors and deadline miss ratios[10, 11]. Abdelzaher *et al*. use PI controllers for control of the CPU utilization factors in Web server end-systems. Moreover, recently, hybrid controllers have been proposed in [12]. All of them require reference values of controlled variables *a priori*. In real-time systems where controlled tasks changes as the time elapses, however, the reference values depend on controlled tasks and must be recalculated. In the recalculation, solving nonlinear equations requires $\mathcal{O}(n^2)$ computation time, where n is the number of the tasks, and characteristics of QoS levels of all tasks must be known.

When CPU utilization factors are decreased with the same ratio for all tasks to avoid overload conditions or to arbitrate the QoS levels of competitive tasks, the quality of services of tasks varies and the deviation of the QoS levels could occur. To prevent unfairness of QoS levels, this paper proposes a QoS adaptive control algorithm to equalize the QoS levels of all active tasks under a constraint where the total CPU utilization factor is constant. The proposed controller obtains a QoS level of each task as feedback data through a monitor and allocates a CPU utilization factor to each task adaptively. The tasks adapt their jobs' parameters to complete the jobs using the allocated CPU utilization factors and release their jobs. A novelty of the proposed method is that it searches a desirable value on-line without increase of computation time in the controller.

The paper organizes as follows: Section 2 introduces a normalized QoS level and define a fairness of QoS levels. Section 3 proposes a resource resource allocation architecture and QoS adaptation controller to achieve fairness of QoS. In Section 4, we describe about analysis and design of the QoS adaptation controller. Section 5 shows the results of simulation experiment of our method. Finally, Section 6 concludes this paper.

2. Task model and QoS fairness

In this paper, we deal with a real-time system with a set of periodic independent tasks $\{\tau_1, \tau_2, \ldots, \tau_N\}$ and one CPU resource. Each task τ_i releases the ℓ-th job J_i^ℓ with a CPU utilization factor r_i^ℓ at time t_i^ℓ, periodically, and it is assumed that every job J_i^ℓ is executed and completed such that the allocated CPU utilization factor r_i^ℓ is satisfied. We introduce a QoS adaptation controller which allocates a CPU utilization factor r_i^ℓ to task τ_i. Shown in Fig. 1 is an illustration of the control architecture whose details will be shown in the next section.

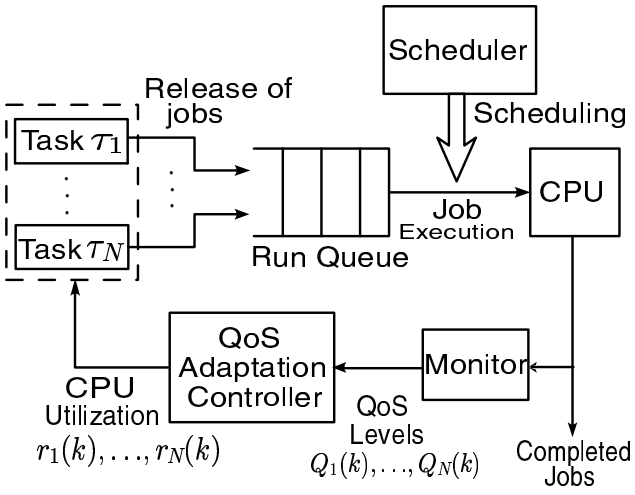

Figure 1. Real-time systems with QoS adaptation control.

An execution result of every job released by the tasks is evaluated as a QoS level. Denoted by QoS_i^ℓ is a QoS level of job J_i^ℓ. Obviously, QoS_i^ℓ depends on both r_i^ℓ and a QoS characteristics of task τ_i. Each task has a minimal and a maximal QoS requirement $QoS_i^{\min}$ and $QoS_i^{\max}$, respectively, where $QoS_i^{\min}$ represents the worst QoS level acceptable to execute τ_i and the $QoS_i^{\max}$ the best QoS level τ_i requires. In order to achieve $QoS_i^{\min}$ and $QoS_i^{\max}$, task τ_i needs specified CPU utilization factors $r_i^{\min}$ and $r_i^{\max}$, respectively. In this paper, we assume the QoS level $Q_i(r)$ of task τ_i is modeled by a real-valued function of the CPU utilization factor r which is continuous and monotonically increasing.

In multi-task real-time systems, resource allocation may cause unfairness in the following sense: we consider two tasks τ_1 and τ_2, and both tasks release jobs with the same CPU utilization factor r. If $r = r_1^{max} = r_2^{min}$ holds, one is executed with its maximal QoS level and the other with its minimal QoS level. This situation is not preferable. In order to evaluate such unfairness, we introduce a normalized QoS level ϕ_i for task τ_i as follows:

$$\phi_i(r) := \begin{cases} 0 & \text{if } r_i < r_i^{\min}, \\ \frac{Q_i(r) - QoS_i^{\min}}{QoS_i^{\max} - QoS_i^{\min}} & \text{if } r_i^{\min} \le r \le r_i^{\max}, \\ 1 & \text{if } r_i > r_i^{\max}. \end{cases} \quad (1)$$

Note that $\phi_i(r)$ is a monotonically increasing function from 0 to 1 with respect to r, and represents the rate of achievement of the QoS level in the sense that $\phi_i(r_i^{\min}) = 0$ means that task τ_i releases a job with its minimal QoS level

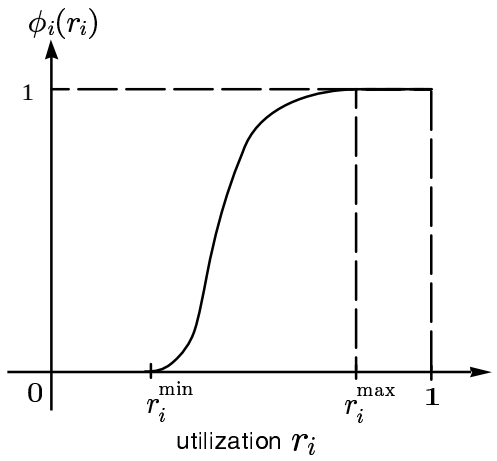

Figure 2. Illustration of function ϕ_i.

QoS_i^{min} while $\phi_i(r_i^{\max}) = 1$ means that the job is executed with its maximal QoS level QoS_i^{max}.

Definition 1 (QoS Fairness) *It is called that a fair resource allocation is achieved by a CPU utilization factors $r_i (i = 1, 2, \ldots, N)$ if the normalized QoS levels ϕ_i satisfy the following equation:*

$$\phi_1(r_1) = \phi_2(r_2) = \cdots = \phi_N(r_N). \tag{2}$$

When every job of task τ_i is released with the CPU utilization factor r_i that satisfies Eq. (2), all tasks can be executed and completed with the same degree of performance.

Since QoS_i is monotonically increasing, the shape of $\phi_i(r)$ can be illustrated as Fig. 2. In the following, a normalized QoS level will be called a QoS level for short. We assume ϕ_i is unknown except the following conditions:

- $\phi_i(r)$ is differentiable in $r \in (r_i^{\min}, r_i^{\max})$.
- For any $r \in (r_i^{\min}, r_i^{\max})$, $0 < \dfrac{d\phi_i}{dr} \leq D_i$.
- In order to guarantee that every job is completed by its deadline, $\sum_{i=1}^{N} r_i^{\min} \leq R \leq \sum_{i=1}^{N} r_i^{\max}$, where R is the desired total CPU utilization factor for which the task set is schedulable.

3. QoS adaptation control

3.1. Architecture

In order to avoid an overload condition and to achieve the fair allocation of CPU utilization factors, we propose an adaptation resource allocation control architecture with on-line search for the fair QoS level shown in Fig. 1, which consists of a basic scheduler, a QoS adaptation controller, and a monitor.

The monitor evaluates a normalized QoS level of each completed job and feeds it back to the QoS Adaptation controller.

The QoS adaptation controller activates periodically or aperiodically, and allocates the CPU utilization factors to each task with searching a fair QoS level on-line. This allocation is based on the normalized QoS levels fed back from the monitor. A control rule used in the QoS adaptation controller will be dealt with in the next section.

The basic scheduler works with a specified scheduling algorithm (e.g., EDF, RM, or DM algorithm). Released jobs are scheduled based on the CPU utilization factor allocated to these corresponding tasks by the QoS adaptation controller. We assume that the least upper bound of the total CPU utilization factor U^{lub} for the basic scheduler is known *a priori*. The allocation by the QoS adaptation controller is done such as the total CPU utilization factor is less than or equal to U^{lub}. The desired total CPU utilization factor R is less than or equal to U^{lub}.

The following notations will be used in this paper:

- t_k: the time of k-th activation of the QoS adaptation controller.
- $r_i(k) = r_i(t_k)$: the CPU utilization factor allocated to task τ_i by the QoS adaptation controller activated at time t_k.
- $Q_i(k) = \phi_i(r_i(t_k))$: the normalized QoS level of the completed job of task τ_i whose CPU utilization factor is equal to $r_i(t_k)$.

To perform the fair QoS adaptation control, this controller dynamically allocates $r_i(k)$ for achieving

$$Q_1(k) = Q_2(k) = \cdots = Q_N(k) \tag{3}$$

under

$$\sum_{i=1}^{N} r_i(k) = R. \tag{4}$$

Since ϕ_i is continuous and monotonically increasing, a set of the fair utilization factors $\{r_1^f, \ldots, r_N^f\}$, which satisfies Eqs. (3) and (4), is uniquely determined under the total CPU utilization factor R and denoted by Q^f is a QoS level where the fair QoS allocation is achieved. If we know Q^f, it is easy to achieve a fair resource allocation by using the conventional control theory. In order to obtain Q^f, however, we have to identify the characteristics of ϕ_i exactly, and it may be a heavy overload to calculate Q^f on-line since ϕ_i is nonlinear and we have to solve a set of nonlinear algebraic equations. So resource allocation method using Q^f is unrealistic and we propose a novel architecture where the QoS adaptation controller allocates a CPU utilization to each task with avoiding the overload condition, searches

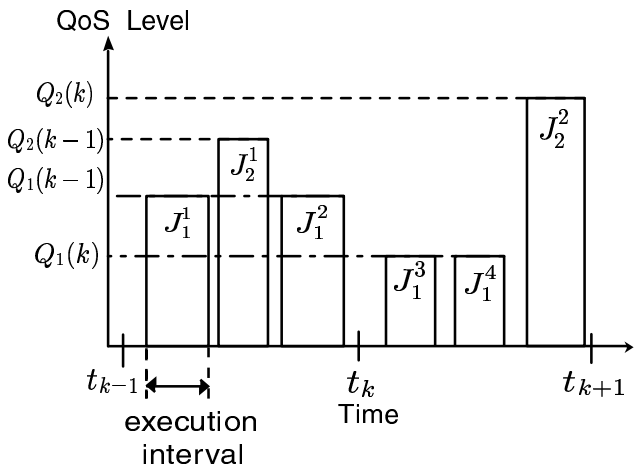

Figure 3. Variation of QoS levels.(J_i^k: the k-th job of task τ_i, $i = 1, 2$)

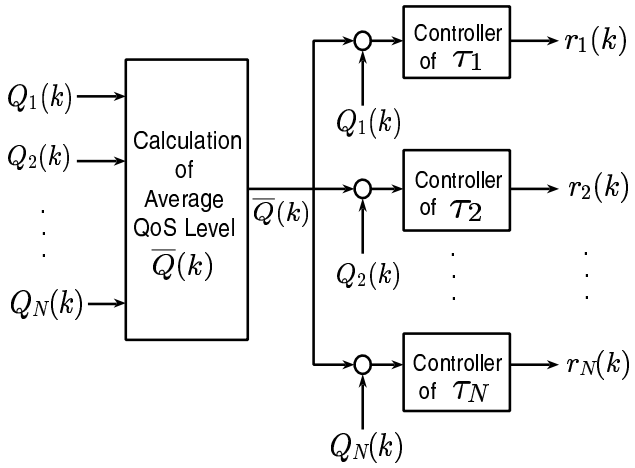

Figure 4. QoS adaptation controller.

the fair QoS level on-line, and achieves a fair resource allocation finally.

3.2. Activation of controller

As described in the previous section, the controller activates at discrete times $t_k (k = 1, 2, \ldots)$. When the QoS adaptation controller activates at the time $t = t_k$, it updates $r_1(k), \ldots, r_N(k)$ based on QoS levels $Q_1(k-1), \ldots, Q_N(k-1)$, which are fed back. The CPU utilization factors of all jobs released by task τ_i in every time interval $[t_k, t_{k+1})$ are equal to $r_i(k)$. After these jobs are completed, the QoS level $Q_i(k)$ is fed back to the QoS adaptation controller through the monitor. Since the allocated CPU utilization factors to all jobs released by τ_i in the interval are same, we can assume that their QoS levels are the same in the interval. Variation of $Q_i(k)$ is illustrated as Fig. 3.

In order to update the QoS levels of all tasks after the activation of the QoS adaptive controller, we assume that the interval of activation of the QoS adaptation controller is large enough for at least one job of each task to be released and completed in $[t_k, t_{k+1})$.

3.3. Control rule

Conventional feedback control rules such as PID control are based on an error between a reference value and its current value in general, and the previous studies on applications of control theory to real-time systems assume that the reference values (depending on control specifications) are given *a priori* [8, 10].

In this paper, control specifications are that all QoS levels of tasks are equal to the fair QoS level Q^f. But, since it depends on a set of active tasks which will be time varying, it is difficult to use the conventional feedback control rules for the control specifications. So this paper proposes a novel control rule which uses only the current QoS levels $Q_1(k),\ Q_2(k),\ \ldots,\ Q_N(k)$.

The proposed control rule is based on an error between $Q_i(k)$ and the average QoS level $\overline{Q}(k)$ of all QoS levels $Q_i(k)$ defined by

$$\overline{Q}(k) = \frac{1}{N} \sum_{i=1}^{N} Q_i(k). \tag{5}$$

If the error is zero, all QoS levels take the same value and the fair resource allocation is achieved.

So, in order for the error to converge to zero, we propose the following control rule for the allocation of the CPU utilization factor $r_i(k)$:

$$r_i(k+1) = r_i(k) + \alpha(\overline{Q}(k) - Q_i(k)), \tag{6}$$

where the real number α is a control parameter. Shown in Fig. 4 is a block diagram of the adaptation controller based on the above control rule. The main objective of this rule is to achieve the fair resource allocation, and this rule is based on discrete-time I(Integral)-control. Note that this control rule is simple and its calculation requires $\mathcal{O}(n)$ computation time. Eq. (6) indicates that if the error between the average QoS level $\overline{Q}(k)$ and $Q_i(k)$ becomes zero, $r_i(k)$ takes a constant value, which means a fair resource allocation is achieved. If $\overline{Q}(k) - Q_i(k) > 0$, then the allocated CPU utilization factor $r_i(k+1)$ at the next activation of the QoS adaptation controller is increased so that released jobs of τ_i will be completed at a larger QoS level. On the other hand, if

$\overline{Q}(k) - Q_i(k) < 0$, then $r_i(k+1)$ is decreased so that its released job will be completed at a smaller QoS level. Thus, all QoS levels are expected to converge to the same value. Moreover, Eq. (6) guarantees that, at each activation time of the QoS adaptation controller, the total CPU utilization factor $\sum_{i=1}^{N} r_i(k)$ is always constant. This is derived from the following equation:

$$\begin{aligned}\sum_{i=1}^{N} r_i(k+1) &= \sum_{i=1}^{N} r_i(k) + \alpha \sum_{i=1}^{N} \left(\overline{Q}(k) - Q_i(k)\right) \\ &= \sum_{i=1}^{N} r_i(k) \\ &= \sum_{i=1}^{N} r_i(k-1) = \cdots = \sum_{i=1}^{N} r_i(0). \end{aligned} \quad (7)$$

Thus, if the initial resource allocation $\{r_1(0), \ldots, r_N(0)\}$ satisfies $\sum_{i=1}^{N} r_i(0) = R$, then $\sum_{i=1}^{N} r_i(k) = R$ for any k.

In Eq. (6), the selection of the gain parameter α is important. It must be selected such that the following two conditions are satisfied:

- **[Feasibility condition]** Each $r_i(k) (r = 1, 2, \ldots, N)$ is positive for every k.
- **[Stability condition]** Every CPU utilization factor r_i converges to the fair resource allocation.

In the next section, we will deal with the above two conditions.

Remark: Many studies use PI- or PID-control[8, 9]. From the control theoretical point of view, our control specification is a kind of servo with a piecewise-constant reference, which is unknown *a priori*, and, by the internal model principle, we need I-control for eliminating offset[13]. Fortunately, we can show in the next section that the above two conditions hold simultaneously without PD-control if we select the control parameter α appropriately.

4. Analysis of the control rule

4.1. Feasibility condition

If the gain parameter α of the control rule (6) is too large, $r_i(k)$ will cause a large variation even if the error $|\overline{Q}(k) - Q_i(k)|$ is small. So $r_i(k)$ may take a negative value, which is infeasible for scheduling. The following lemma guarantees the feasibility of the control rule.

Lemma 1 (Feasibility condition) *Assume that* $r_i(0) \geq 0$ *and* $\sum_{i=1}^{N} r_i(0) = R$. *If*

$$0 < \alpha \leq \frac{1}{\max_i D_i}, \quad (8)$$

then $r_i(k) \geq 0$ *for every* k.

Proof:
Since $d\phi_i(r_i)/dr_i \leq D_i$ and $\phi_i(0) = 0$, we have $Q_i(k) \leq D_i r_i(k)$.

We will use the induction method. Suppose that $r_i(k) \geq 0$ and $\sum_{i=1}^{N} r_i(k) = R$ for k. Then

$$\begin{aligned} r_i(k+1) &= r_i(k) + \alpha(\overline{Q}(k) - Q_i(k)) \\ &\geq \left(\frac{1}{D_i} - \frac{1}{\max_j D_j}\right) Q_i(k) + \alpha \overline{Q}(k) \\ &\geq 0. \end{aligned}$$

□

4.2. Stability condition

Assume that $\sum_{i=1}^{N} r_i(0) = R$, which keeps the total CPU utilization factor to R, from Eq. (7). A fair resource allocation $r_i(k)$ with $Q_1(k) = Q_2(k) = \cdots = Q_N(k) = Q^f$ is a fixed point of Eq. (6). We will derive a stability condition for the fair resource allocation. Let

$$r(k) := \begin{bmatrix} r_1(k) \\ r_2(k) \\ \vdots \\ r_{N-1}(k) \end{bmatrix}, \qquad Q(k) := \begin{bmatrix} Q_1(k) \\ Q_2(k) \\ \vdots \\ Q_N(k) \end{bmatrix}. \quad (9)$$

Note that $r_N(k)$ is excluded in the vector $r(k)$ since it is determined by $r_N(k) = R - \sum_{i=1}^{N-1} r_i(k)$.

By Eq. (6) and $Q_i(k) = \phi_i(r_i(k))$, the real-time system is modeled by the following difference equation:

$$\begin{aligned} r(k+1) &= r(k) + \begin{bmatrix} \overline{Q}(k) \\ \vdots \\ \overline{Q}(k) \end{bmatrix} - Q(k) \\ &= r(k) + \\ &\quad \begin{bmatrix} \frac{1-N}{N} & \frac{1}{N} & \cdots & \frac{1}{N} & \frac{1}{N} \\ \frac{1}{N} & \frac{1-N}{N} & \cdots & \frac{1}{N} & \frac{1}{N} \\ \vdots & & \ddots & & \vdots \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1-N}{N} & \frac{1}{N} \end{bmatrix} \\ &\quad \times \begin{bmatrix} \phi_1(r_1(k)) \\ \vdots \\ \phi_{N-1}(r_{N-1}(k)) \\ \phi_N(R - \sum_{i=1}^{N-1} r_i(k)) \end{bmatrix}. \end{aligned} \quad (10)$$

Now let

$$\Delta r(k) := r(k) - \begin{bmatrix} r_1^f \\ \vdots \\ r_{N-1}^f \end{bmatrix}, \quad (11)$$

$$\Delta \phi_i(r_i) := \phi(r_i - r_i^f) - Q^f. \quad (12)$$

Note that $r_N(k) - r_N^f = -\sum_{i=1}^{N-1} \Delta r_i(k)$ and $0 < \frac{d\Delta\phi_i}{d\Delta r_i} \leq D_i$. Then, Eq. (10) can be rewritten as follows:

$$\begin{aligned}\Delta r(k+1) &= \Delta r(k) + \\ &\alpha \begin{bmatrix} \frac{1-N}{N} & \frac{1}{N} & \cdots & \frac{1}{N} & \frac{1}{N} \\ \frac{1}{N} & \frac{1-N}{N} & \cdots & \frac{1}{N} & \frac{1}{N} \\ \vdots & & \ddots & & \vdots \\ \frac{1}{N} & \frac{1}{N} & \cdots & \frac{1-N}{N} & \frac{1}{N} \end{bmatrix} \\ &\times \begin{bmatrix} \Delta\phi_1(\Delta r_1(k)) \\ \vdots \\ \Delta\phi_{N-1}(\Delta r_{N-1}(k)) \\ \Delta\phi_N(-\sum_{i=1}^{N-1} \Delta r_i(k)) \end{bmatrix} \end{aligned} \quad (13)$$

The linearized equation of Eq. (13) around the origin is given by

$$\Delta r(k+1) = A\Delta r(k) := (a_{ij})\Delta r(k), \quad (14)$$

$$a_{ij} = \begin{cases} 1 - \frac{\alpha}{N}\{(N-1)d_i + d_N\} & \text{if } i = j, \\ \frac{\alpha}{N}(d_j - d_N) & \text{otherwise,} \end{cases} \quad (15)$$

where

$$d_i = \frac{d\Delta\phi_i(0)}{d\Delta r_i} = \frac{d\phi_i(r_i^f)}{dr_i},\ i = 1, 2, \ldots, N. \quad (16)$$

A stability condition of the origin of Eq. (14) is given by the following lemma.

Lemma 2 (Stability condition) *If*

$$0 < \alpha \leq \frac{N}{(N-1)\max_i d_i + d_N}, \quad (17)$$

then the origin of Eq. (14) is asymptotically stable.

Proof:
Denoted by $\| \cdot \|$ is the L_1 matrix norm[14]. Since $\|\Delta r(k+1)\|_1 \leq \|A\|_1 \|\Delta r(k)\|_1$, it is sufficient to prove that $\|A\|_1 < 1$, which leads to $\lim_{k\to\infty} \Delta r(k) = 0$. Without loss of generality, assume that

$$d_N \leq d_i,\ i = 1, 2, \ldots, N-1. \quad (18)$$

Then, using Eqs. (17) and (18), we have

$$\begin{aligned}\|A\|_1 &= \max_j \sum_{i=1}^{N-1} |a_{ij}| \\ &= \max_j \Big[1 - \frac{\alpha}{N}\{(N-1)d_j + d_N\} \\ &\quad + \sum_{i\neq j, i=1}^{N-1} \frac{\alpha}{N}(d_j - d_N)\Big] \\ &= \max_j \Big[1 - \frac{\alpha}{N}\{(N-1)d_j + d_N\} \\ &\quad + \frac{N-2}{N}\alpha(d_j - d_N)\Big] \\ &= \max_j \Big[1 - \frac{\alpha}{N}\{d_j + (N-1)d_N\}\Big] \\ &< 1. \end{aligned}$$

task	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
period	500	400	700	600	300	600
phase	0	40	30	50	10	0
D_i	2.50	6.28	1.90	2.00	3.14	2.36
$r_i^{\min}$	0	0	0.029	0	0	0
$r_i^{\max}$	0.40	0.250	0.857	0.50	0.50	0.333
ϕ_i	(I)	(IV)	(II)	(I)	(III)	(III)

1*: every period is equal to its corresponding relative deadline.

Table 1. Task parameters for simulation.

□

4.3. Selection of α

According to Lemmas 1 and 2, Eqs. (8) and (17) give sufficient conditions to achieve a fair resource allocation control.

Since

$$\max_i D_i \geq d_j,\ j = 1, 2, \ldots, N, \quad (19)$$

Eq. (8) implies Eq. (17). Thus, even if d_i is unknown, we can use a sufficient condition for both the feasibility and the stability to hold simultaneously shown in the following theorem.

Theorem 1 (Fair resource allocation condition) *Assume that $r_i(0) \geq 0$ and $\sum_{i=1}^N r_i(0) = R$. If Eq. (8) holds, each $Q_i(k)$ converges to Q^f with satisfying that $0 \leq r_i(k) \leq R$ and $\sum_{i=1}^N r_i(k) = R$ for every k.*

5. Simulation

We show simulation experiments to evaluate the performance of the proposed QoS adaptation control. We consider a periodic task set $\{\tau_1, \tau_2, \ldots, \tau_6\}$ shown in Table 1. Each task has a relative deadline equal to its period. Each ϕ_i is one of the following functions:

$$(\mathrm{I}) Q_i(k) = (r_i(k) - r_i^{\min})/(r_i^{\max} - r_i^{\min}) \quad (20)$$

$$(\mathrm{II}) Q_i(k) = \sin\frac{\pi(r_i(k) - r_i^{\min})}{2(r_i^{\max} - r_i^{\min})} \quad (21)$$

$$(\mathrm{III}) Q_i(k) = 0.5 + 0.5\sin\frac{\pi\left(r_i(k) - \frac{r_i^{\max} - r_i\min}{2}\right)}{r_i^{\max} - r_i^{\min}} \quad (22)$$

$$(\mathrm{IV}) Q_i(k) = 1 + \sin\frac{\pi(r_i(k) - r_i^{\max})}{2(r_i^{\max} - r_i^{\min})} \quad (23)$$

The QoS adaptation controller activates every 2000 unit times, namely $t_k = 2000 \times k$. This guarantees that at least

one job is released and completed for each task in the interval of activations $[t_k,\ t_{k+1}]$. The gain parameter α is set to be $1/6.28(\approx 0.159)$, which satisfies Eq. (8).

Two typical scheduling algorithms for basic scheduler are used in simulations.

A: We use the *Earliest Deadline First*(EDF) algorithm. We set $R = 0.8$ so that it is schedulable[1, 2]. In this case, the theoretical value Q^f of a fair QoS level is equal to 0.26 by solving Eqs. (3) and (4), numerically.

B: We use *Rate Monotonic*(RM) algorithm. We set $R = 0.6$ so that schedulability for any periodic task set is guaranteed[1, 2]. In this case, we have $Q^f = 0.17$.

The results of simulations **A** and **B** are as shown in Figs. 5 and 6, respectively. These figures show that fair resource allocations are achieved after some times of activation of the controller. Moreover, in both simulations, every $Q_i(k)$ converges to the theoretical value Q^f of the fair QoS level and no deadline miss occurs. Thus, these simulations show that the proposed QoS adaptation control method achieves a fair resource allocation without an overload condition.

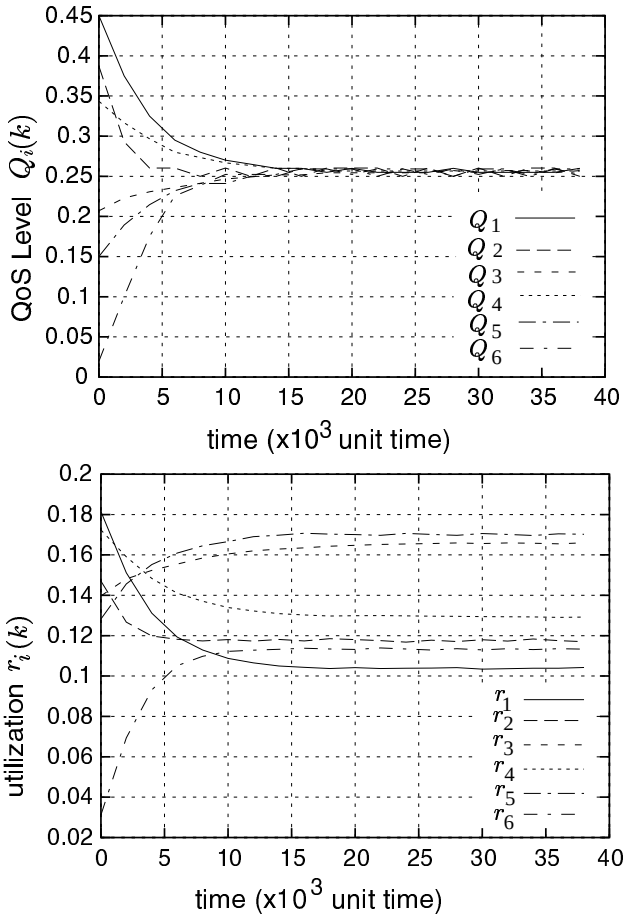

Figure 5. Behavior of QoS level and CPU utilization with EDF scheduling and $R = 0.8$.

6. Conclusions

We proposed a novel control method for a fair resource allocation based on QoS levels. The proposed adaptation controller allocates a CPU utilization factor to each task with on-line search for the fair QoS level, and the proposed control rule is simple since allocations is based on errors between the current QoS levels and their average. So, its computation time is small so that it does not yield a heavy overhead.

As future work, we are relaxing the fair resource allocation condition given by Theorem 1, which is a sufficient condition, and it is shown in simulation that the fair resource allocation can be achieved even if the gain parameter is larger than Eq. (8). Moreover, we focused on the stability and the feasibility, but transient behavior of controlled QoS levels is also an important problem.

It is also future work to generalize the proposed method to a multiple resource case and a multiple QoS dimensions case.

References

[1] G. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Kluwer Academic Publishers, Boston, 1997.

[2] J. Liu, *Real-Time Systems*, Prentice Hall, Upper Saddle River, NJ, 2000.

[3] T. Abdelzaher, E. Atkins, and K. Shin, "QoS Negotiation in Real-Time Systems and its Application to Automated Flight Control", *Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium*, Montreal, June 1997, pages 228–238.

[4] G. Buttazzo and L. Abeni,"Adaptive Workload Management through Elastic Scheduling," *Real-Time Systems*, July 2002, pages 23(1–2):7–24.

[5] G. Buttazzo, G. Lipari, M. Caccamo, and L. Abeni, "Elastic Scheduling for Flexible Workload Management," *IEEE Transactions on Computers*, March 2002, pages 51(3):289–302.

[6] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A Resource Allocation Model for QoS Management," *Proceedings of the 18th IEEE Real-Time Systems Symposium*, San Francisco, December 1997, pages 298-307.

[7] S. Oikawa and R. Rajkumar, "Portable RK: a Portable Resource Kernel for Guaranteed and Enforced Timing Behavior," *Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium*, Vancouver, June 1999, pages 111–120.

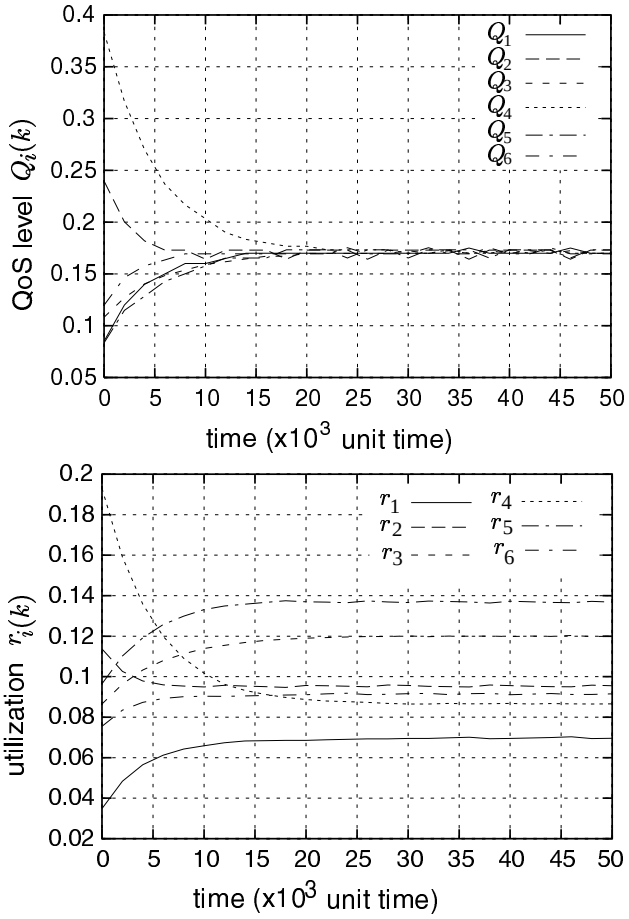

Figure 6. Behavior of QoS level and CPU utilization with RM scheduling and $R = 0.6$.

[8] D. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-Driven Proportional Allocator for Real-Rate Scheduling," *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, January 1999.

[9] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, "Analysis of a Reservation-based Feedback Scheduler," *Proceedings of the 23rd Real-Time Systems Symposium*, Austin, December 2002, pages 71–80.

[10] C. Lu, J. Stankovic, S. Son, and G. Tao, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Real-Time Systems*, July/September 2002, pages 23(1):85–126.

[11] T. Abdelzaher, K. Shin, and N. Bhatti, "Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach," *IEEE Transactions on Parallel and Distributed Systems*, January 2002, pages 13(1):80–96.

[12] L. Palopoli, L. Abeni, and G. Lipari, "On the Application of Hybrid Control to CPU Reservation," *Hybrid Systems: Computation and Control*, LNCS2623, Prague, April 2003, pages 389–404.

[13] W. Wonham, *Linear Multivariable Control: A Geometric Approach*, Third edition, Springer-Verlag, NY, 1985.

[14] C. Desoer and M. Vidyasagar, *Feedback Systems: Input-Output Properties*, Academic Press, NY, 1975.