論　文

QoS レベル公平化に基づくリアルタイムシステムの QoS 適応制御

原田　史子[†]　　潮　　俊光[†]　　中本　幸一[††]

QoS Adaptation Control of Real-Time Systems Based on QoS Level Equalization

Fumiko HARADA[†], Toshimitsu USHIO[†], and Yukikazu NAKAMOTO[††]

あらまし　リアルタイムシステムにおける過負荷状態時には，各タスクの CPU 利用率を下げることでデッドラインミスを回避する必要がある．一般に各タスクの QoS レベルはそれぞれ異なった CPU 利用率の非線形関数で表されるため，同じ割合で CPU 利用率を変化させたとき， QoS レベルの変化はタスクによって異なる．本論文では，過負荷状態を避けながら QoS レベルの偏りをなくすために，指定された総 CPU 利用率のもとですべてのタスクの QoS レベルを公平化するような CPU 利用率の制御を行う QoS 適応制御法を提案する．QoS レベルを公平化するような CPU 利用率を求め，それに基づいて制御するのは困難であるが，本手法では，現在の平均 QoS レベルに基づいて制御するような QoS 適応制御器を導入することで， QoS 公平化を達成する CPU 利用率が未知であっても QoS の公平化が可能である．この目的を達成するために，制御理論を用いて，QoS 適応制御器で配分される CPU 利用率がすべての QoS レベルを等しくする値に収束する条件について述べる．またシミュレーションによって本手法の有効性を示す．

キーワード　リアルタイムシステム，QoS 適応制御，CPU 利用率

1. まえがき

リアルタイムシステムにおいて，過負荷状態はシステム応答の予測性，システム性能の著しい低下を引き起こす [1]．従来，過負荷状態が発生しないように，タスクの起動要求があったとき，その実行を許可するか否かというアドミッション制御を行ってきた．最近では，タスクに割り当てられたサービス品質（Quality of Service, QoS）に応じて，タスクは CPU 利用率を下げてジョブをリリースすることで，過負荷状態を避けるという研究が行われている．

一方，処理時に割り当てられる資源が増加するにしたがって，提供するサービス品質 QoS が向上する，Flexible Application と呼ばれるソフトウェアが存在する [2, pp.394–419]．ここで QoS レベルは，リリースされたジョブの実行結果に対する利用者の満足度のレベルを表す．例えば，MPEG データの再生処理ではイメージ全体を有する I フレームと I フレームからの差分情報を有する B フレーム，P フレームを処理する必要がある．この場合，I フレームのみならず，P フレーム，B フレームの処理を行うタスクに CPU 時間とメモリなどの資源を割り当てることにより，より精確な MPEG 画像を再生できる．しかし，各ソフトウェアが提供する QoS レベルを同時に保つ，あるいは向上させようとすると，システムが過負荷状態になる可能性がある．このため，競合する QoS レベルを調停する必要が出てくる．

QoS レベルを調停する研究はこれまでいくつかなされてきた．システム過負荷時のタスクに対して受容可能な QoS レベルを予め記述し，システム過負荷時には，この記述をもとに各タスクとネゴシエーションを行うことにより，各タスクはリリースするジョブの QoS レベルを低下させ，過負荷状態に対処する研究がされている [3]．Buttazzo らは，リアルタイムシステムにおいて，ジョブの処理がバネのように伸縮するものと見なすエラスティックモデルを導入し，過負荷時には各ジョブの実行時間を短縮して過負荷状況に適応するアルゴリズムを提案している [4]．この場合，プログラム設計者はアプリケーションソフトウェアをエラスティックモデルに基づき設計しなければならない．

[†] 大阪大学大学院基礎工学研究科，豊中市
Graduate School of Engineering Science, Osaka University, 1–3 Machikaneyama-cho, Toyonaka-shi, Osaka, 560–8531 Japan
[††] NEC ネットワーク開発研究本部，横浜市
NEC Network Development Laboratories, 4035, Ikebe-cho, Tsuzuki-ku, Yokohama, 224–3555, Japan

これは容易なことでないと考えられる．Rajkumar らは，複数の資源に同時にアクセスする場合に QoS に基づく資源確保モデルを提案している [5]．またこのモデルに基づき，各アプリケーションソフトウェアの持つ最低限の QoS レベルの要求を満たすという制限下で，QoS レベルを最大化するアルゴリズムを提案している．追川らはリアルタイムソフトウェアの時間上の QoS レベルを保証するためにソフトウェアアーキテクチャと資源予約を行う API を提案し，ポータブルなカーネルモジュールを開発している [6]．本カーネルモジュールではアドミッション制御機能を提供している．

一方，リアルタイムシステムに制御理論を適用する研究が近年注目されている．[7] では，リアルタイムシステムにおけるタスクの CPU 使用率を管理する手法に，制御理論が利用されている．この研究では，CPU の使用時間を制御するために PID（Propotional-integral-derivative）制御器が応用されている．しかしながら，望ましい状態に制御できるための PID 制御器の条件についての理論的考察はなされていない．

また，タスクの CPU 利用率やデッドラインミス率を指定された値に維持するためにフィードバック制御理論を用いる研究がなされている．例えば [8] では，相対的デッドラインの目標値を与え，実際の相対的デッドラインとその目標値の差を基にして，CPU 時間の予約を PI 制御器によって行うことで，デッドラインミス率をある範囲内に押さえている．[9] では，CPU 利用率またはデッドラインミス率に対して目標値を与え，ゲインフィードバックによる制御法を提案している．いずれの場合にも目標値を前もって与える必要がある．しかしながら，目標値は処理されるべきタスクの集合に依存しており，時間とともにタスクの集合が変動するようなリアルタイムシステムでは，タスクの集合が変動すると目標値の再計算が必要となる．この計算には，非線形方程式の数値計算が含まれており，n をタスクの個数としたとき，$O(n^2)$ の計算量になる．

過負荷状態を回避するために，あるいは競合する QoS レベルを調停するために，一般に同じ割合でタスクの CPU 利用率を変化させたとき，QoS レベルの変化はタスクによって異なる．本論文では，QoS レベルの偏りをなくすために，指定された総 CPU 利用率のもとですべてのタスクの QoS レベルを公平化するような QoS 適応制御手法を提案する．提案手法では制御理論を用いて，各タスクの QoS レベルをフィードバックすることで各タスクに CPU 利用率を割り当てる．各タスクはその割り当てられた CPU 利用率で処理できるように処理内容を変更してジョブをリリースする．さらに，本論文で提案する手法では前もって目標値を計算する必要はなく，制御器での計算時間をほとんど増やすことなく，目標値を探索しながらそれに近づいていくという特徴をもっている．

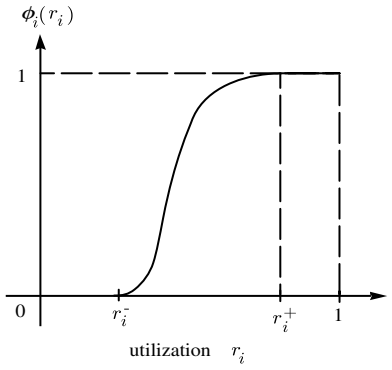


図 1　関数 ϕ_i の概形の例
Fig. 1　An example of illustration of function ϕ_i.

本論文の構成は以下のとおりである．**2.** で，本論文で取り扱うリアルタイムシステムと QoS 適応制御系の構成と制御目的について述べる．**3.** で QoS レベル公平化を実現するような QoS 適応制御器を提案する．**4.** でシミュレーション実験によって提案手法を評価する．最後に，**5.** で結論を述べる．

2. 問題設定

2.1 リアルタイムシステムと QoS レベル

本論文では，独立周期タスク集合 $\{T_1, T_2, \ldots, T_N\}$，スケジューラ，及び一個のプロセッサからなるリアルタイムシステムを考える．$\{T_1, \ldots, T_N\}$ とスケジューラで用いられるスケジューリング則は既知であり，総 CPU 利用率が R 以下ならばそのスケジューリング則でタスク集合がスケジュール可能であるとする．以下，CPU 利用率を単に利用率と書く．

各タスク T_i は周期的にジョブをリリースする．各ジョブはリリースされるときに，T_i に割り当てられた利用率 r_i $(0 < r_i \leqq 1)$ に応じてプロセッサの利用時間が与えられる．

ジョブは実行が終了すると，その処理結果を QoS（Quality of Service）として評価する．タスクがリリースするジョブが利用率 r_i で実行されたときの QoS レ

ベルを $QoS_i(r_i)$ と書くことにする．本論文では各 T_i の QoS レベル $QoS_i(r_i)$ がスカラー値で，下限 QoS_i^- から上限 QoS_i^+ までの連続値を持つ場合を考える．QoS_i^-，QoS_i^+ はそれぞれ利用率の最少要求量 r_i^- と最大要求量 r_i^+ によって得られる QoS レベルに対応している．ここで関数 $\phi_i(r_i)$ を

$$\phi_i(r_i) = \frac{QoS_i(r_i) - QoS_i^-}{QoS_i^+ - QoS_i^-} \tag{1}$$

と定義すると，$0 \leqq \phi_i(r_i) \leqq 1$ であり，$\phi_i(r_i) = 1$ のときはタスク T_i に最大 QoS レベルとなる利用率が，$\phi_i(r_i) = 0$ のときは最小の QoS レベルとなる利用率が，それぞれ割り当てられていることを意味する．各タスク T_i の QoS レベルが QoS_i^- から QoS_i^+ までの値をとるので，$\phi_i(r_i)$ は QoS レベルの達成度を意味していると考えることができ，全てのタスクで $\phi_i(r_i)$ が等しくなること，即ち

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

となるように各タスクの利用率 $r_1, \ldots, r_N$ が定められているとき，利用率が公平に割り当てられていると考える [10]．

本論文では ϕ_i が以下の条件を満たすと仮定する．

- ϕ_i は $r_i \in (r_i^-, r_i^+)$ で微分可能な単調増加関数であり，

$$\phi_i(0) = 0 \tag{3}$$

$$0 < \frac{d\phi_i}{dr_i} \leqq D_i \tag{4}$$

を満たす．ここで D_i は ϕ_i の最大変化率であり，$D_i < \infty$ とする．

- 任意の $r_i \leqq r_i^-$ に対して $\phi_i(r_i) = 0$
- 任意の $r_i \geqq r_i^+$ に対して $\phi_i(r_i) = 1$

例えば ϕ_i は図 1 のような形を持つ．以降，単に QoS レベルとは $\phi_i(r_i)$ で表される値を指すものとする．本論文では，過負荷時において各タスクの利用率を抑えることで各タスクの QoS は下がるが，デッドラインミスを避けるような利用率の制御器を提案する．そのため，ジョブをリリースするときに，制御器によって指定された利用率で処理できるようにジョブの処理内容を変更できる機能を各タスクがもっていると仮定する．次節では，この制御器を設計するときの制御目的について説明する．

2.2 制御目的

時刻 t において，各タスク T_i が利用率 r_i^+ でジョブをリリースしたとき，総利用率 $\sum_{i=1}^N r_i^+$ が R 以下であれば，リリースされたジョブはすべてデッドラインまでに処理を終了することができるので，特に制御する必要はない．また，$\sum_{i=1}^N r_i^-$ が R より大きい場合は各タスクに最小限必要な利用率を割り当てられない．よって本論文では，$\sum_{i=1}^N r_i^+ > R > \sum_{i=1}^N r_i^-$ である場合について検討する．このとき，r_i を小さくすると ϕ_i も小さくなるが，一般にその変化は各タスクによって異なるので，同じ割合で r_i を減らすと QoS レベル ϕ_i の値が式 (2) を満たさなくなり，公平性が損なわれる．一方，明らかに $\sum_{i=1}^N r_i = R$ とおけば，デッドラインミスを発生することなく QoS レベルをより大きく取れる．またシステムの動作を保証するために，各タスク T_i に R より大きい利用率を配分したり，負の量の利用率を配分することを避けなければならない．ただし $\sum_{i=1}^N r_i(t) = R$ のもとでは，すべての $r_i(t)$ が 0 または正値であれば自動的に $r_i(t) \in [0, R]$ が保証される．よって，以下各時刻 $t \geqq 0$ における T_i の利用率を $r_i(t)$ とおくと，

$$r_i(t) \geqq 0 \tag{5}$$

$$\sum_{i=1}^N r_i(t) = R \tag{6}$$

を満たすように動的に $r_i(t)$ を変更しつつ，公平になるように，すなわち，式 (2) を満たす r_i の値に $r_i(t)$ が収束させることが制御目的となる．本論文では，この制御目的を実現するための制御器を QoS 適応制御器と呼ぶことにする．

2.3 QoS 適応制御システム

本論文では，図 2 に示すような QoS 適応制御システムを提案する．本システムでは実行終了したジョブの QoS レベルを測定するモニタと QoS 適応制御器からフィードバック機構が構成される．以下，表記の簡略化のために，$\phi_i(r_i(t))$ を $Q_i(t)$ と書くと，本システムの動作は以下のようになる．

QoS 適応制御器は指定された時刻になると起動される．以下，QoS 適応制御器の l 回目の起動時刻を t_l とおく．時刻 t_l で QoS 適応制御器によって配分された各タスクの利用率は，次に QoS 適応制御器が起動する時刻 t_{l+1} まで保持される．即ち，$t_l \leqq t < t_{l+1}$ では $r_i(t) = r_i(t_l)$ となる．

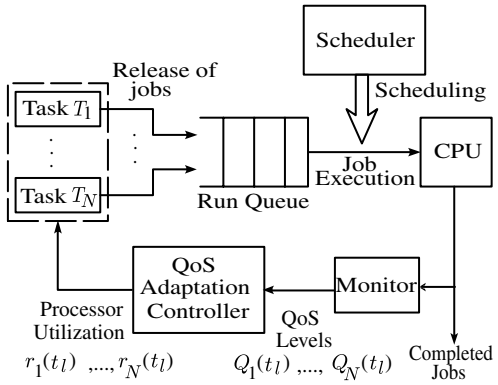


図 2　QoS 適応制御システム
Fig. 2　QoS adaptation control system.

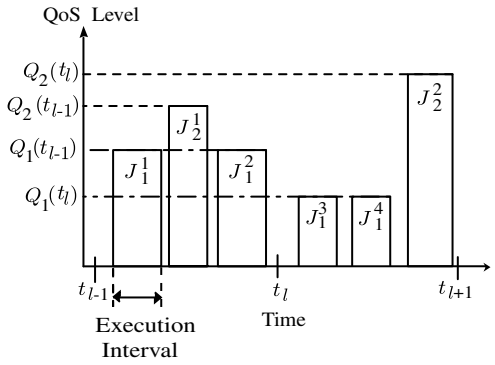


図 3　QoS レベルの経時変化
（J_i^k: タスク T_i の k 番目のジョブ，$i = 1, 2$）
Fig. 3　Variation of QoS Levels.
(J_i^k: the k-th job of task T_i, $i = 1, 2$)

制御の流れは次のようになる．時刻 $t \in [t_l, t_{l+1})$ の間にタスク T_i が k 番目のジョブ J_i^k をリリースすると，J_i^k は利用率が $r_i(t_l)$ になるような処理を行う．J_i^k の実行終了後，利用率 $r_i(t_l)$ によって決まるタスク T_i の QoS レベル $Q_i(t_l)$ がモニタで測定され，QoS 適応制御器に送られる．それに基づいて QoS 適応制御器が時刻 t_{l+1} までに $r_i(t_{l+1})$ を計算する．時刻 $t \in [t_{l+1}, t_{l+2})$ では，同様にリリースされたジョブには利用率 $r_i(t_{l+1})$ が与えられる．簡単のため，時刻 $t \in [t_l, t_{l+1})$ の間で，すべてのタスクが少なくとも一つのジョブをリリースし，それが実行終了となるように QoS 適応制御器の起動時間間隔をとるようにする．このことにより QoS 適応制御器はすべての更新された $Q_i(r_i)$ の値を知ることができる．また，スケジューリング則によるジョブのリリース順，実行順に依存しない制御を行うことができる．例えば，タスク T_1，T_2 の QoS レベルを制御する場合，各タスクの QoS レベルの経時変化は図 3 のようになる．時刻 $t \in [t_{l-1}, t_l)$ では，この範囲の時間にリリースされたジョブの利用率によってタスクの QoS レベルが決まり，時刻 t_{l+1} に QoS 適応制御器が起動するまで QoS レベルは不変である．

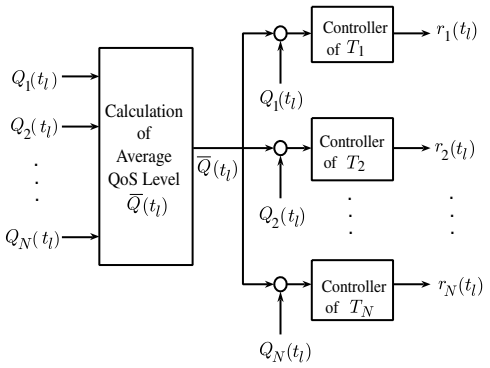


図 4　QoS 適応制御器
Fig. 4　QoS adaptation controller.

ϕ_i が連続関数であることから，

$$\sum_{i=1}^{N} r_i^e = R \tag{7}$$

$$\phi_1(r_1^e) = \phi_2(r_2^e) = \cdots = \phi_N(r_N^e) = Q^e \tag{8}$$

となる定数 $r_i^e > 0$，$Q^e > 0$ が必ず存在する．もし r_i^e を求めることができるならば，QoS 適応制御器は r_i^e を目標値として文献 [7], [9], [11] と同様に PID 制御などの制御器を構成できる．しかしながら，そのためには ϕ_i の形状を QoS 適応制御器に知らせる必要がある．さらに，たとえ ϕ_i がわかったとしても，ϕ_i は非線形関数であるため式 (7)，(8) を満たす r_i^e，Q^e を効率よく計算することは難しく，リアルタイムで計算することは望ましくない．そこで次章では，r_i^e，Q^e を計算することなく，r_i，Q_i が r_i^e，Q^e に収束するような QoS 適応制御器の設計法を提案する．

3. QoS 適応制御器の設計法

3.1 制　御　則

前章で述べたように Q^e が未知のため，Q^e と $Q_i(t)$ の偏差に基づいて制御することは困難である．そこで $Q_i(t)$ を未知の Q^e に近づけるために，各時刻での QoS レベルの平均値 $\overline{Q}(t) := \frac{1}{N}\sum_{j=1}^{N} Q_j(t)$ を仮想的な目標値として，$\sum_{i=1}^{n} r_i(t_l) = R$ のもとで $\overline{Q}(t)$

と $Q_i(t)$ の偏差に基づく QoS 適応制御器を提案する．図 4 に提案する QoS 適応制御器のブロック線図を示す．図 2 のモニタによって測定された各タスク T_i の QoS レベル $Q_i(t_l)$ を用いて，すべてのタスク間での QoS レベルの平均値 $\overline{Q}(t_l)$ を計算し，各タスク T_i の制御器では，$\overline{Q}(t_l)$ と $Q_i(t_l)$ の誤差をなくすように利用率 $r_i(t_{l+1})$ を設定する．各タスクに対する制御器では他のタスクに関する情報をまったく用いないので全体としてはシンプルな構成となる．T_i の制御器の構成には様々な方法が考えられるが，本論文では情報量が少ない次式を用いることにする．

$$r_i(t_l) = r_i(t_{l-1}) + \alpha(\overline{Q}(t_{l-1}) - Q_i(t_{l-1})) \quad (9)$$

但し α はフィードバックゲインで，全ての制御器で同じ値を用いる．式 (9) の直感的な解釈は以下のとおりである．$\overline{Q}(t_l) - Q_i(t_l)$ を 0 に近づけることで $Q_1(t_l)$, $Q_2(t_l)$, $\cdots$, $Q_N(t_l)$ の間のばらつきが小さくなっていき，最終的に $Q_1(t) = Q_2(t) = \cdots = Q_N(t)$ が達成されるように制御することを示している．

$t_0 = 0$ とし，QoS 適応制御を開始した時刻であるとする．$r_i(t_0) = r_i(0)$ は QoS 適応制御器の初期値であり，式 (5) を満たすのであれば任意の値に設定できる．

QoS 適応制御器のパラメータ α を設定するときに以下の二点が重要な問題である．

(i)QoS 適応制御器で生成される $r_i(t)$ が，実行可能スケジュールを生成できることを保証する．

(ii) 各 $r_i(t)$ が r_i^e に収束する，即ちシステムが r_i^e, Q^e で漸近安定となる．これはある回数以上 QoS 適応制御器を起動すれば，Q_i が必ず公平化された値 Q^e に近づき，QoS レベル公平化が達成されることを意味する．

問題 (i) と (ii) についてはそれぞれ 3.2 節，3.3 節で検討する．

3.2 実行可能な利用率

スケジュール可能性を保証するために式 (5)，(6) を満たさなければならない．これに関して補題 1 が成立する．

［補題 1］

$$\sum_{i=1}^{N} r_i(0) = R \quad (10)$$

$$0 \leqq r_i(0), \quad i = 1, 2, \ldots, N \quad (11)$$

$$0 < \alpha \leqq \frac{1}{\max_i D_i} \quad (12)$$

が成り立つならば任意の時刻 $t_l \geqq 0$ で式 (5)，(6) が成り立つ．

(証明)

$Q_i(t_l) \in [0,1],\ i = 1, 2, \ldots, N$ より $-1 \leqq \overline{Q}(k) - Q_i(k) \leqq 1,\ i = 1, 2, \ldots, N$ が成り立つ．また

$$F_i(r_i) = D_i r_i - \phi_i(r_i)$$

とおくと，

$$F_i(0) = 0$$
$$\frac{dF_i}{dr_i} = D_i - \frac{d\phi_i}{dr_i} \geqq 0$$

となることから $\phi_i(r_i) \leqq D_i r_i$ が成り立つ．

明らかに $t = t_l,\ l = 1, 2, \ldots$ において式 (5)，(6) が成り立つことを示せば十分である．ここでは帰納法を用いて証明する．時刻 t_l で式 (5)，(6) が成り立っているとき，

$$\begin{aligned}
\sum_{i=1}^{N} r_i(t_{l+1}) &= \sum_{i=1}^{N} \left[r_i(t_l) + \alpha(\overline{Q}(t_l) - Q_i(t_l)) \right] \\
&= \sum_{i=1}^{N} r_i(t_l) = R \\
r_i(t_{l+1}) &= r_i(t_l) + \alpha(\overline{Q}(t_l) - Q_i(t_l)) \\
&\geqq \left(\frac{1}{D_i} - \frac{1}{\max_j D_j} \right) Q_i(t_l) \\
&\quad + \alpha \overline{Q}(t_l) \\
&\geqq 0
\end{aligned}$$

より時刻 t_{l+1} で式 (5)，(6) が成立する． □

3.3 制御系の安定性

QoS レベルの公平化が達成できるための条件を求める．以下，

$$\left. \frac{d\phi_i(r_i)}{dr_i} \right|_{r_i = r_i^e} = d_i,\ i = 1, 2, \ldots, N \quad (13)$$

とおき，一般性を失うことなく

$$d_N \leqq d_i,\ i = 1, 2, \ldots, N-1 \quad (14)$$

と仮定する．

$\Delta r(t) = [r_1(t) - r_1^e, \cdots, r_{N-1}(t) - r_{N-1}^e]^T$ とおく

と，不動点 $[r_1^e,\ldots,r_N^e]^T$ の近傍での式 (1)，(9) の線形化システムは次式のように書ける．

$$\Delta r(t_{l+1}) = A\Delta r(t_l) \tag{15}$$

ただし $A=(a_{ij})\in\mathbb{R}^{N-1\times N-1}$ で

$$a_{ij}=\begin{cases} 1-\dfrac{\alpha}{N}\{(N-1)d_i+d_N\} & i=j \\ \dfrac{\alpha}{N}(d_j-d_N) & i\neq j \end{cases} \tag{16}$$

である．このとき，式 (15) の原点が漸近安定 [12]，即ち $\Delta r(t)$ が原点に収束すればよい．これはある回数以上 QoS 適応制御器を起動すれば，必ず $r_i(t_l)$ が r_i^e に近づき，その結果 $Q_i(t_l)$ が Q^e に収束して QoS レベルの公平化が達成されることを意味する．このことに関して補題 2 が成立する．

［補題 2］ 式 (14) が成立するとき，

$$0<\alpha\leqq\frac{N}{(N-1)\max_i d_i+d_N} \tag{17}$$

ならば 式 (15) の原点が漸近安定となる．

（証明）

式 (14) より

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

が成り立つ．したがって

$$\lim_{l\to\infty}\|\Delta r(t_l)\|_1=0$$

となり式 (15) の原点は漸近安定である． □

補題 2 の証明では L_1 ノルムを用いて漸近安定となる十分条件を示したが，この十分条件を緩めることは今後の課題である．

表 1 シミュレーションにおけるタスクパラメータ
Table 1 Task Parameters for Simulation.

タスク	T_1	T_2	T_3	T_4	T_5	T_6
周期	500	400	700	600	300	600
位相	0	40	30	50	10	0
D_i	2.50	6.28	1.90	2.00	3.14	2.36
r_i^-	0	0	0.029	0	0	0
r_i^+	0.40	0.250	0.857	0.50	0.50	0.333

注 1：周期，位相は単位時間で表記
注 2：各タスクの周期と相対デッドラインは等しい

3.4 ゲインパラメータの設定

補題 1，2 より，QoS レベルの公平化を達成するために，QoS 適応制御器のゲインパラメータ α を式 (12)，(17) を満たすように設定する必要がある．提案手法では不動点を求めずに制御をしているので，式 (17) の d_j は未知である．しかし，$d_j\leqq D_i$ であることを利用すると式 (12) が成立すれば式 (17) も成立する．以上より定理 1 が導かれる．

［定理 1］（QoS レベル公平化可能条件） 式 (10) と式 (11) が満たされるとき，

$$\alpha\leqq\frac{1}{\max_i D_i} \tag{18}$$

ならば式 (1) で与えられる QoS レベルを持つ独立な周期タスク集合 $\{T_1,T_2,\ldots,T_N\}$ の QoS レベル公平化が可能である． □

$\|A\|_1$ が小さいほど $r_i(t_l)$ の r_i^e への収束は速くなるので，補題 2 の証明からわかるように，α をできるだけ大きくとるほうがよい．従って実用上は

$$\alpha=\frac{1}{\max_j D_j} \tag{19}$$

と選べばよい．

4. シミュレーション実験

前章までの結果をもとにシミュレーション実験を行った．対象となる独立なタスク集合 $\{T_1,T_2,\ldots,T_6\}$ は表 1 に示すようなパラメータを持ち，各タスクの相対デッドラインは周期に等しい．表中の周期と位相は単位時間で表されている．QoS 適応制御器が次に起動するまでに，全てのタスクのジョブが少なくとも一つリリースされ，実行終了されていることを保証するために，QoS 適応制御器は 2000 単位時間ごとに周期的に起動するように設定する（$t_l=2000\times l$）．使用するスケジューリング則として Earlist Deadline First(EDF) アルゴリズムと Rate Monotonic(RM) アルゴリズム

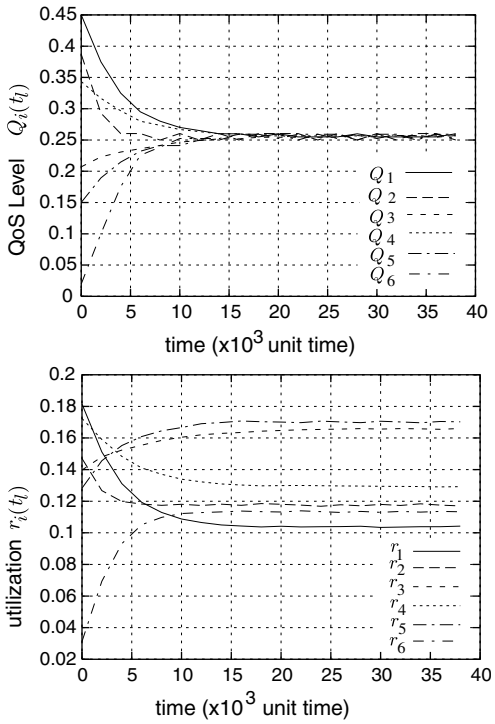


図 5　EDF スケジューリング，$R = 0.8$ のときの QoS レベル，利用率配分量の時間的変化
Fig. 5　Behavior of QoS Level and CPU Utilization with EDF Scheduling and $R = 0.8$.

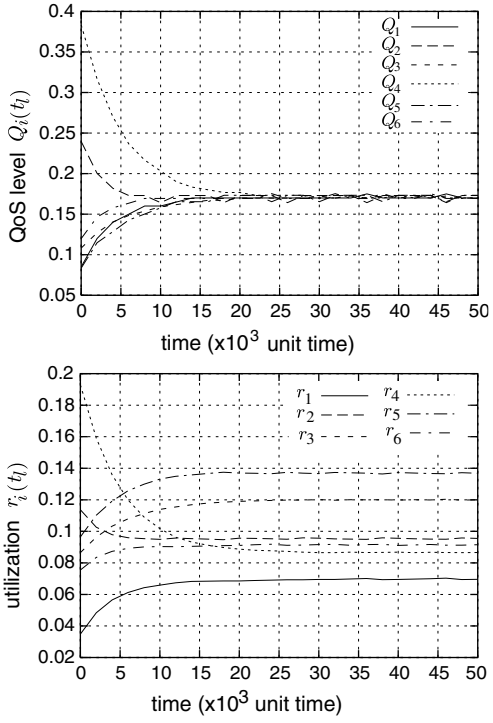


図 6　RM スケジューリング，$R = 0.6$ のときの QoS レベル，利用率配分量の時間的変化
Fig. 6　Behavior of QoS Level and CPU Utilization with RM Scheduling and $R = 0.6$.

の 2 種類を考える．前者の場合は $R \leqq 1$ で，後者の場合は $R \leqq 0.69$ であればスケジュール可能であることが知られている [1], [2], [13]．本論文では前者の場合に $R = 0.8$，後者の場合に $R = 0.6$ としてシミュレーションを行う．式 (7)，(8) を数値的に解くと Q^e の値は前者で約 0.26，後者で約 0.17 になる．式 (19) を満たすように $\alpha = 1/6.28 \approx 0.159$ に定めて提案手法でシミュレーションした結果について，EDF の場合の QoS レベルと利用率の時間的変化を図 5 に，RM の場合を図 6 に示す．いずれの場合も，$Q_i(t_l)$ は数値的に解いた Q^e に収束している．さらに，デッドラインミスもまったく発生しなかった．このようにどのようなスケジューリング則を用いても，スケジュール可能となるように R を設定する限り，本手法を用いれば，r_i^e を計算することなく，QoS レベルを公平化できることを確認した．

5. むすび

本論文では，すべてのタスクの QoS レベルが公平になるように，各タスクの CPU 利用率を動的に変更する QoS 適応制御法を提案した．本制御では現在得られているタスクの QoS レベルの平均値との偏差をもとに，ゲインフィードバックによって QoS レベルの公平化が達成できる．

提案手法における問題の一つとして，QoS 適応制御器を起動する時刻 t_l をどのように決定するかという問題がある．例えば QoS 適応制御器の起動間隔が短かすぎる場合，利用率 $r_i(t_l)$ で実行されたタスクの QoS レベル $Q_i(t_l)$ が得られる前に時刻 t_{l+1} になり，QoS 適応制御器が起動し，式 (9) による制御が実行できなくなる可能性がある．反対に起動間隔が長すぎる場合，$Q_i(t_l)$ の Q^e への収束が遅くなる．したがって QoS 適応制御器の起動時刻 t_l の設定は今後の重要な課題である．また提案手法を実装し，実際の環境で本手法の有効性を確認することも今後の課題である．

謝辞　貴重なコメントを頂いた編集委員ならびに査読委員の方々に感謝します。

文　献

[1] G.C. 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," Proc. IEEE Real-Time Technology and Application Symposium, 1997.

[4] G. Buttazzo and L. Abeni, "Adaptive workload management through elastic scheduling," Real-Time Systems, vol.23, No. 1, pp.7–24, 2002.

[5] R. Rajkumar, C. Lee, J. Leoczky, and D. Siewiorek, "A resource allocation model for QoS management," Proc. 18th IEEE Real-Time Systems Symposium, pp.298–307, December 1997.

[6] S. Oikawa and R. Rajkumar, "Portable RK: A portable Resource Kernel for guaranteed and enforced timing behavior, " Proc. IEEE Real-Time Technology and Applications Symposium, June 1999.

[7] D. Steere, A. Goel, J. Greunberg, D. McNamee, C. Pu and J. Walpole, "A feedback-driven proportion allocator for Real-Rate scheduling," Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, pp.145–158, 1999.

[8] L. Abeni, L. Palopoli, G. Lipari and J. Walpole, "Analysis of a Reservation-Based Feedback Scehduler," Processdings of the 23rd Real-Time Systems Symposium, pp.71–80, 2002.

[9] C. Lu, J. Stankovic, S. Son, and G. Tao, "Feedback control real-time scheduling: framework, modeling, and algorithms," Real-Time Systems, vol.23, No.1, pp.85–126, 2002.

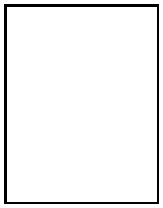
[10] 久保, 潮, 村田, 大崎, "伝播遅延時間を考慮した ATM の PID ふくそう制御, " 信学論, vol.J85-B, no.3, pp.371–380, 2002.

[11] B. Li and K. Nahrstedt, "A control-based middleware framework for Quality-of-Service adaptations," IEEE journal on selected areas in communications, vol.17, no.9, pp.1632–1650, September 1999.

[12] K.J. Åström and B. Wittenmark, Computer-Controlled Systems, Theory and Design, Third Edition, Prentice Hall, Upper Saddle River, NJ, 1997.

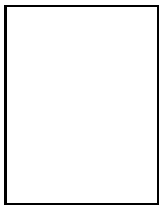
[13] 白川洋充, 竹垣盛一, リアルタイムシステムとその応用, システム制御情報学会 (編), 朝倉書店, 2001.

（平成 x 年 xx 月 xx 日受付）



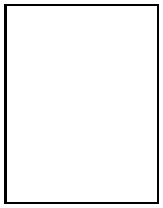
原田　史子　（学生員）

平 15 阪大・基礎工・システム科学卒．同年同大大学院基礎工学研究科博士前期課程入学，現在に至る．リアルタイムシステムの QoS 適応制御に関する研究に従事．



潮　俊光　（正員）

昭 55 神戸大・工・システム卒．昭 60 同大大学博士課程了．現在，大阪大学大学院基礎工学研究科教授．離散事象システムの制御，非線形現象の解析などの研究に従事．学術博士．システム制御情報学会，情報処理学会，IEEE など各会員．



中本　幸一　（正員）

昭 57 大阪大学大学院博士前期課程了．同年 NEC 入社．現在ネットワーク開発研究本部モバイルターミナル開発研究部長．リアルタイムシステム，分散システム，モバイルシステム，ソフトウェア開発環境の研究開発に従事．平 2〜3Cornell 大学計算機科学科客員研究員．平 9 大阪大学大学院博士課程入学．平 12 単位取得退学．博士（工学）．平 15 より電気通信大学客員教授．情報処理学会，日本ソフトウェア科学会，IEEE Computer Society 各会員．