情報処理学会論文誌 Vol.55 No.8 1856–1865 (Aug. 2014)

適応型スケジューリングによる平均応答時間の短縮法 — 実行時間見積り方法の影響

田中 清史[1,a)]

受付日 2013年11月12日, 採録日 2014年5月17日

概要：タスクの実行時間を考慮に入れるリアルタイムスケジューリング方式では，実行時間として最悪実行時間が費やされることが仮定される．しかし実際のシステム上で動作するタスクは，ほとんどの場合で最悪実行時間よりも短い時間で実行を完了する．著者は過去に非周期リクエストに対して予測実行時間を適用することにより，平均応答時間を短縮することを目的とした適応型リアルタイムスケジューリング法を提案した．この方法は，従来の Total Bandwidth Server における最悪実行時間を使用するデッドライン計算に対し，最悪実行時間の代わりに予測実行時間を使用して短いデッドラインを得ることで応答時間の短縮を実現する．予測実行時間を経過した際は最悪実行時間に基づくデッドラインに更新することでスケジューラビリティを保つ．本論文では，適応型リアルタイムスケジューリング法における実行時間の見積り方法を改善する手法を提案する．この改善手法では，適応型 TBS における 2 段階のデッドライン計算を多段階での実行時間見積りおよびデッドライン更新に拡張することにより，予測精度の粗さによる応答時間への影響を回避可能である．評価では，提案改善手法により平均応答時間が短縮されることと，提案手法のタスク実行のプリエンプション回数への影響，およびデッドラインの再計算回数への影響を確認する．

キーワード：リアルタイムスケジューリング，Total Bandwidth Server（TBS），最悪実行時間（WCET），予測実行時間（PET）

A Method of Shortening Average Response Times by Adaptive Scheduling — Effects of Estimating Execution Times

Kiyofumi Tanaka[1,a)]

Received: November 12, 2013, Accepted: May 17, 2014

Abstract: In real-time scheduling methods that take tasks' execution times into account, worst-case execution times (WCETs) are assumed to be spent for tasks' execution. However, in actual systems, tasks often finish in shorter execution times than their WCETs. In the previous research, the author proposed an adaptive real-time scheduling to shorten response times of aperiodic requests by applying predictive execution times. This technique achieves short response times by urgent deadlines obtained using predicted execution times instead of worst-case execution times which is supposed to be used in the deadline calculation of Total Bandwidth Server. When the predicted execution time expire, the deadline is changed to a conservative one corresponding to the worst-case execution time, which preserves schedulability. In this paper, methods of improving the estimation of tasks' execution times are proposed. In these methods, while deadlines are calculated at most twice in the previous proposal, predictive execution times and the corresponding deadlines are updated in a multistep fashion, which solves the problem of inaccurate prediction of execution times. In the evaluation, it is confirmed that the improved methods reduce average response times and how the methods influence the number of times preemption occurs and the number of times deadlines are re-calculated.

Keywords: real-time scheduling, total bandwidth server (TBS), worst case execution time (WCET), predictive execution time (PET)

1. はじめに

近年の組込みシステムの複雑化と多様化にともない，異

[1] 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923–1292, Japan
[a)] kiyofumi@jaist.ac.jp

なる性質，あるいは重要度の異なるタスクが混在するシステムが主流になりつつある [1], [2]．たとえば，完全なリアルタイム性が要求される対象機器の制御タスクと，ある程度の応答性能は要求されるが完全なリアルタイム処理は要求されないユーザインタフェースなどのタスクの混在があげられる．前者はハードタスク，後者はソフトタスクあるいは非リアルタイムタスクと呼ばれる [3]．このようなシステムで要求されるリアルタイム処理を実現するためには，ハードタスクとソフト（または非リアルタイム）タスクの両方を対象とし，ハードタスク群のスケジューラビリティを保証し，ソフト（または非リアルタイム）タスクの応答時間を短縮できるリアルタイムスケジューリングアルゴリズムを使用する必要がある．

ハードタスクはデッドラインを守る必要があるため，周期的な起動を前提とし，最悪実行時間（WCET）の実行を想定してスケジューラビリティを事前に確認することが重要である．一方，ソフトタスクに対するリアルタイム性の要求は厳しくないため，非周期的な起動が許される．このようなタスクセットを対象とし，スケジューラビリティと短い応答時間を提供するスケジューリングアルゴリズムとして，Total Bandwidth Server（TBS）がある [4]．TBS は Earliest Deadline First（EDF）スケジューリング [5] を基本としているため，スケジューラビリティを維持しつつプロセッサ使用率を 100%まで上げることが可能であるという特長がある．

近年のプロセッサやプログラムは複雑化の一途をたどり，WCET の正確な見積りは困難になっている．たとえば命令の高度なパイプライン実行は実行時間の見積りを困難にし，システムに多数のタスクが存在すると，キャッシュヒット/ミスの予測は困難を極める [6]．また，プログラム内には数多くの分岐やループ構造が含まれるため，すべての入力パターンに対して実行が最も長くなるパスを見つけることは事実上不可能である [7]．結果的に，安全のため WCET は悲観的に見積もらざるをえず，実際の実行時間とのギャップが大きい．

著者は過去の研究で，TBS において周期タスクのスケジューラビリティを確保しつつ，非周期タスクのデッドライン計算の中で最悪実行時間の代わりに予測実行時間を使用することによって，非周期タスクの応答時間を短縮する適応型 TBS を提案した [8], [10], [11]．この方式は，従来の TBS における WCET を使用するデッドライン計算に対し，WCET の代わりに予測実行時間を使用して短いデッドラインを得ることで応答時間の短縮を実現する．予測実行時間を経過した際は WCET に基づくデッドラインに更新することでスケジューラビリティを保つ．この方式はタスクが繰り返し実行され，実行のたびに実行時間が変動する場合に特に有力であるが，性能が実行時間の予測精度に依存する性質があった．本論文では，適応型 TBS における 2 段階のデッドライン計算を多段階での実行時間見積りおよびデッドライン更新に拡張することにより，予測精度の粗さによる応答時間への影響を回避する方法を提案する．

本論文ではまず，2 章で関連研究を述べ，3 章で適応型 TBS の概要を述べる．続いて 4 章において，考察を通して応答時間をさらに短縮するために実行時間の見積り方法を見直し，新たな実行時間見積法を導入した適応型 TBS の改良版を提案し，評価する．最後に 5 章で本論文をまとめる．

2. 関連研究

周期タスクと非周期タスクが混在するタスクセットに対する様々なスケジューリングアルゴリズムがある．それらは固定優先度サーバと動的優先度サーバに分類できる．固定優先度サーバは Rate Monotonic（RM）[5] をベースとするものであり，高優先度のタスクのジッタが小さいという特長がある．固定優先度サーバの代表的な例として，Deferrable Server [13]，Priority Exchange [13]，Sporadic Server [14]，Slack Stealing [15] などが提案されている．一方，動的優先度サーバは EDF をベースとするものであり，スケジューラビリティを保証しつつ，プロセッサ使用率を 100%まで上げることが可能であることが特長である．代表例として，Dynamic Priority Exchange [4]，Dynamic Sporadic Server [4], Earliest Deadline Late Server [4], Constant Bandwidth Server [12] などがある．これらのアルゴリズムはすべて非周期リクエストの応答時間を短縮することを目的としているが，その効果は実装の複雑さとのトレードオフである．

Total Bandwidth Server（TBS）はハードタスクと非リアルタイムタスクの混在セットに対する動的優先度サーバの 1 つであり，非リアルタイムタスクに対して短い応答時間を提供し，かつ軽い実装を可能としている [4], [16]．前提として，ハードタスクは周期的に起動され，周期の長さと一致する相対デッドラインを持つ．一方，非リアルタイムタスクは非周期的に起動され，デッドラインは持たないが，起動（要求）されたときに，以下の式から求まる仮のデッドライン d_k が与えられる．

$$d_k = \max(r_k, d_{k-1}) + \frac{C_k}{U_s} \tag{1}$$

ここで，k は k 番目の非周期タスクインスタンスであることを意味する．r_k は k 番目のインスタンスの要求時刻，d_{k-1} は $k-1$ 番目（1 つ前）のインスタンスのデッドライン時刻，C_k は k 番目のインスタンスの最悪実行時間，および U_s は非周期タスクの実行を担当するサーバのための，プロセッサ使用率として表現されるバンド幅である．この式により，非周期タスクの要求発生ごとに，そのインスタンスの実行に U_s のバンド幅を与えることになる．$\max(r_k, d_{k-1})$ の項により，連続するインスタンス間でバンド幅が重なら

ないように調整している．非周期タスクのインスタンスが本式により仮のデッドラインを与えられた後，すべての周期タスクと非周期タスクがEDFアルゴリズムによってスケジュールされる．U_p をハード（周期）タスクのプロセッサ使用率とすると，$U_p + U_s \leq 1$ でタスクセットがスケジューラブルであることが証明されている [4]．

TBSにおいて，非周期タスクのWCETが過大見積りされている状況下では，式 (1) によってタスクのデッドラインが過大に計算されることになる．したがって，当該タスクの実行が遅延され，応答時間が長くなる可能性がある．文献 [17] では，タスク実行が終了したときに，実際に費やした実行時間によってそのタスクのデッドラインを再計算し，後続する非周期タスクのデッドライン計算に反映させるリソース回収（resource reclaiming）を試みている．これにより，後続するインスタンスは短いデッドラインの恩恵を受けて応答時間が改善することが期待できる．

リソース回収をともなうTBSでは，非周期タスクは以下の式によって仮のデッドラインが与えられる．

$$d'_k = \overline{r}_k + \frac{C_k}{U_s} \tag{2}$$

$\overline{r}_k$ は以下で計算される値である．

$$\overline{r}_k = \max(r_k, \overline{d}_{k-1}, f_{k-1}) \tag{3}$$

すなわち，要求時刻と，前のインスタンスの再計算されたデッドライン（$\overline{d}_{k-1}$，後述），および前のインスタンスの終了時刻（f_{k-1}）のうち，最大のものがリリース時刻として選ばれる（$\overline{d}_{k-1}$ が f_{k-1} よりも早くなることがありうる．これは，あくまでも $k-1$ 番目のインスタンスはリソース回収前の古いデッドラインで実行されたためである）．$k-1$ 番目の非周期タスクが終了したとき，実際に費やした実行時間（$\overline{C}_{k-1}$）を使用する以下の式によってデッドラインの再計算を行い，後続タスクのリリース時刻選択の式 (3)，続いて後続タスクのデッドライン計算の式 (2) へ反映させる．

$$\overline{d}_{k-1} = \overline{r}_{k-1} + \frac{\overline{C}_{k-1}}{U_s} \tag{4}$$

3. 適応型TBS

本章では，実行時間の変動を考慮するアルゴリズムとして，著者が過去に提案した適応型TBSの概要を説明する．

TBSの対象タスクセットにおいて，非周期タスクはデッドラインを持たないため，WCETを仮定したスケジューラビリティ保証を行う必要はない．また，TBSは非周期タスクに仮のデッドラインを与えるが，このデッドラインをミスすることは深刻ではない．したがって，TBSのデッドライン計算にWCETを使用することは必須ではなく，より短い実行時間を仮定することが可能である．仮定した実行時間から計算されるデッドラインを使用して，タスクセット全体のスケジューラビリティを確保し，仮定した実行時間が経過した際には，再度長い実行時間であるWCETを使用してデッドラインを再計算すればよい．この方法により，非周期タスクが仮定した実行時間で終了した場合は，その短いデッドラインとEDFアルゴリズムにより応答時間の短縮が期待できる．

3.1 予測実行時間（Predictive Execution Time：PET）

適応型TBSでは実行時間を予測する必要がある．文献 [8] では以下の方法をとった．

$$\begin{aligned} C_{i_k}^{PET} &= \alpha \times C_{i_{k-1}}^{PET} + (1-\alpha) \times C_{i_{k-1}}^{ET} \\ C_{i_0}^{PET} &= C_i^{WCET} \end{aligned} \tag{5}$$

ここで，$C_{i_k}^{PET}$ は非周期タスク J_i の k 回目の実行のための予測実行時間を意味する．$C_{i_{k-1}}^{ET}$ は同一タスクの前回の実行時の実際に費やした実行時間を意味する．初期値 $C_{i_0}^{PET}$ はそのタスクのWCET（C_i^{WCET}）とする．この式は，加重係数を α として，前回の予測実行時間と前回の実際の実行時間との加重平均を計算して，実行時間の予測値とするものである．この予測方法により，同一タスクの実行時間の変動に追随することを狙っている．タスクの実行時間の見積りに関して，類似のフィードバック方法をとるものとして文献 [9] などがある．

3.2 適応型TBSの定義

適応型TBSでは非周期タスクのインスタンスは2つに分解される．それらを異なるタスクと見なすことによって，TBSと同一の方式として帰着可能である．

以下の説明では，非周期タスクを区別せず（式 (5) 内の i を無視し），起動要求順の通し番号 k を使用する．k 番目の非周期タスクのインスタンスである J_k の実行を2つのサブインスタンス J_k^{PET} と J_k^{REST} に分割する．J_k^{PET} は J_k の最初から予測された終了時刻までの実行に相当する．J_k^{REST} は予測された終了時刻から後の実行に相当する．J_k が予測終了時刻かその前に終了した場合は，J_k^{REST} は存在しないことになる．J_k の最悪実行時間を C_k^{WCET}，J_k の予測実行時間を C_k^{PET}，J_k^{REST} の実行時間を $C_k^{REST} = C_k^{WCET} - C_k^{PET}$ とする[*1]．k 番目の非周期リクエスト（J_k）が時刻 $t = r_k$ に到着したとき，2つのサブインスタンスには以下の仮のデッドラインが与えられる．

$$d_k^{PET} = \max(r_k, d_{k-1}) + \frac{C_k^{PET}}{U_s} \tag{6}$$

$$d_k^{REST} = d_k^{PET} + \frac{C_k^{REST}}{U_s} \tag{7}$$

（オリジナルの）TBSのデッドライン計算は以下のとおりであった．

[*1] 実際には $0 \leq C_k^{REST} \leq C_k^{WCET} - C_k^{PET}$ である．適応型TBSでは最悪の場合を想定し，$C_k^{REST} = C_k^{WCET} - C_k^{PET}$ とする．

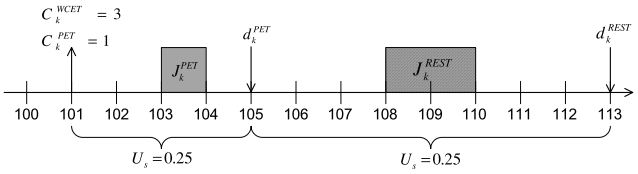

図 1 適応型 TBS におけるデッドライン割当て
Fig. 1 Deadline assignment in the adaptive TBS.

$$d_k^{WCET} = \max(r_k, d_{k-1}) + \frac{C_k^{WCET}}{U_s} \tag{8}$$

$C_k^{REST} = C_k^{WCET} - C_k^{PET}$ と式 (6)，(7)，(8) から，$d_k^{REST} = d_k^{WCET}$ となる．したがって，非周期タスクが到着した際に，式 (6) と式 (8) により 2 つのデッドラインが計算できる．式 (7) ではなく式 (8) を使用することにより，右辺第二項がタスクごとに定数となり，システム稼働前に 1 度計算するのみでよいため，デッドライン計算のオーバヘッドを抑えることが可能である．

図 1 はティック 101 で到着した非周期タスクリクエストに対して，デッドラインを設定する例である．当該非周期タスクは k 番目の非周期インスタンスであり，最悪実行時間が $C_k^{WCET} = 3$，予測実行時間が $C_k^{PET} = 1$ と想定されている．また，非周期サーバのバンド幅は 0.25 とする．予測実行時間に相当する実行である J_k^{PET} に対して，デッドラインは $d_k^{PET} = 101 + 1/0.25 = 105$ となり，残りの実行（図中の J_k^{REST}）に対するデッドラインは $d_k^{REST} = d_k^{WCET} = 101 + 3/0.25 = 113$ となる．この例では予測実行時間内で実行が終了した場合は応答時間は $104 - 101 = 3$ となり，残り部分が実行された場合は応答時間は $110 - 101 = 9$ となる．

3.3 適応型 TBS のスケジューラビリティ

非周期リクエストを 2 つのサブインスタンスに分割した後は，適応型 TBS はオリジナル TBS と同様に振る舞う．すなわち，2 つの（異なる）サブインスタンスは同時に発生したものと考える．式 (6) と式 (7) から，明らかに，2 つのサブインスタンスによる，$\max(r_k, d_{k-1})$ と $d_k^{REST} (= d_k)$ の間での使用率は，以下のようにオリジナル TBS における使用率と同じく U_s となる．

$$U_{J_k^{PET}} = \frac{C_k^{PET}}{d_k^{PET} - \max(r_k, d_{k-1})} = U_s$$

$$U_{J_k^{REST}} = \frac{C_k^{REST}}{d_k^{REST} - d_k^{PET}} = U_s$$

したがって，適応型 TBS のスケジューラビリティは文献 [16] におけるオリジナル TBS のものと同一となり，$U_p + U_s \leq 1$ のときタスクセットはスケジュール可能となる．

3.4 実装の複雑さ

オリジナル TBS と比較し，適応型 TBS はデッドライン計算に PET を使用する点および PET を更新する点と，PET の経過時にデッドラインを再設定する点が異なる．概念的には非周期タスクの実行インスタンスは 2 つのサブインスタンスに分解されるが，オペレーティングシステムはタスクを 1 つの情報セット（タスク制御ブロック）で管理するべきである．これを実現するためには，PET が経過したときにタスクが未完了だった場合に，デッドラインを再設定し，タスクをレディキューに再挿入すればよい．その際，3.2 節で述べたように，デッドラインの再計算のオーバヘッドを軽減するために，式 (8) の右辺第二項を静的に計算しておき，必要なときに使用するべきである．その他，PET に到達したことを検出するために，スケジューラを全ティックタイミングで実行するべきである．このことは，タイマー/ティック割り込みの発生時にスケジューラを呼び出すことで実現されるが，これは（実は）オペレーティングシステムが通常とる自然な手続きである．

3.5 リソース回収法の適用

2 章で述べたリソース回収法 [17] が適応型 TBS に容易に適用できる．非周期リクエストの実行が終了したときは，それが分割後の第一，第二のサブインスタンスのどちらであるかにかかわらず，デッドラインが式 (4) によって再計算され，更新されたデッドラインを式 (3) に適用し，結果的に，後続非周期タスクは式 (2) によってより早いデッドラインを与えられることになる．

4. 実行時間見積り法とアルゴリズムの改良

本章では，適応型 TBS の問題点を議論し，それを解決する改良版の適応型 TBS を提案し，評価する．

4.1 適応型 TBS の特徴と実行時間予測方法の改良

文献 [8] における評価では，3.1 節の実行時間の予測方法（$\alpha = 0.5$）を使用しており，結果としては実行時間が既知の場合（すなわち理想的に完全に予測可能な場合）と比較し，応答時間に大きな差があった．このことは，実行時間の予測方法を改善することで，大きな改善が期待できることを意味する．

適応型 TBS の性質を考慮すると，予測が過大見積りとなった場合は十分に短いデッドラインが得られず，短い応答時間を期待できない．逆に過小見積りの場合は，予測時間が経過しても実行が終了せず，残りの実行は WCET に基づくデッドラインに従ってスケジュールされることになり，やはり短い応答時間は期待できない．

適応型 TBS のこれらの弱点は，次のように解決可能である．予測実行時間として，実行要求後の最初は最小のもの，すなわち 1 ティックの長さを使用する．これにより，実際に実行時間が 1 ティック以内の場合以外は予測実行時間内で終了しない．予測実行時間の経過時に，従来の適応型 TBS では残りの実行に対して WCET に基づいたデッド

ラインを割り当てていたが，これを改良し，残りの実行時間が 1 ティックと仮定した場合のデッドラインを与える．残り実行の実際の所要時間が 1 ティック以内の場合以外は，最初の実行と同様，予測実行時間経過時に再度 1 ティック分の残り時間を仮定してデッドラインを更新する．以下，繰り返す．この方法により，実行時間の過大見積りに起因するデッドラインの延長の問題，および過小見積り時に残り実行時間として WCET を仮定することによるデッドライン延長の問題を解決可能である．

本改良方式の欠点としては，タスクの実行が終了するまで毎ティック，デッドラインの再設定が必要となることがあげられる．これを緩和するために，予測実行時間の初期値として 1 ティックではなく，より大きな長さを使用することが有効である．その長さとしては，タスクの最短実行時間（Best-Case Execution Time：BCET）を基準とした長さが候補となる．なぜなら，BCET より短い時間で実行が終了することはないためである．ただし WCET と同様，BCET の取得方法が課題となる．本論文の評価では，各時点において，同一タスクのそれまでの全実行で最も短い実行時間を BCET とする方法をとり，予測実行時間の初期値として，BCET，BCET の 2 倍，BCET の 4 倍，および BCET の 8 倍を使用し，比較する．

4.2 改良版適応型 TBS の定義

改良版適応型 TBS では，非周期タスクは 1 つ以上のサブインスタンスに分解される．適応型 TBS と同様，各サブインスタンスを異なるタスクと見なすことにより，TBS に帰着可能である．

非周期タスクの k 番目のインスタンス J_k の実行をサブインスタンス $J_k^1, J_k^2, J_k^3, \ldots$ に分割する．J_k^1 は J_k の最初から，1 ティック実行後までの実行に相当する．J_k^i は $i-1$ ティック実行後から i ティック実行後までの実行に相当する．J_k が i ティックで終了した場合は，J_k^{i+1} 以降のサブインスタンスは存在しないことになる．また，予測実行時間の初期値として 1 ティックより大きな値 j を使用する場合は，J_k^1 から J_k^j までをあわせたものが最初のサブインスタンスとなり，これを J_k^j とする．

k 番目の非周期リクエスト（J_k）が時刻 $t = r_k$ で到着したとき，最初のサブインスタンス J_k^j と後続のサブインスタンス J_k^i にはそれぞれ以下のデッドライン d_k^j と d_k^i が与えられる．

$$d_k^j = \max(r_k, d_{k-1}) + \frac{j}{U_s} \tag{9}$$

$$d_k^i = d_k^{i-1} + \frac{1}{U_s} \qquad (i > j) \tag{10}$$

上記式内で，d_{k-1} は J_{k-1} のデッドラインである．この部分に関しては，J_{k-1} の最後のサブインスタンス実行終了時に，リソース回収を適用することが可能である．

非周期タスクが発生すると，最初のデッドラインを式 (9) によって計算し，当該タスクの制御ブロックをレディキューに挿入する．このタスクは，予測実行時間の初期値の長さを実行した後と，その後 1 ティック実行されるたびに，式 (10) によってデッドラインが再計算され，レディキューに再挿入される．これをタスク実行が終了するまで繰り返す．この方法では，十分短い初期値を使用する限り，適応型 TBS における実行時間予測の過大見積り，過小見積りに起因する問題が発生しない．

4.3 改良版適応型 TBS のスケジューラビリティ

改良版適応型 TBS のスケジューラビリティは適応型 TBS とオリジナル TBS の場合と等しい．すなわち，タスクセットがスケジュール可能である必要十分条件は $U_p + U_s \leq 1$ となる．なぜなら，適応型 TBS との唯一の違いは非周期タスク分割後のサブインスタンスの数であるが，すべてのサブインスタンスを異なるタスク実行と見なせば，やはり TBS と同一のスケジューラビリティの性質が保たれるためである．

4.4 改良版適応型 TBS のスケジュール例

図 2 において，上段の (1) は PET が過大見積りされた場合の適応型 TBS，中段の (2) は PET が過小見積りされた場合の適応型 TBS，下段の (3) が改良版適応型 TBS のスケジュール例（予測実行時間の初期値が 1 ティックの場合）を示している．すべての方式において，周期 $T_1 = 4$，実行時間 $C_1 = 2$ の周期タスク τ_1，周期 $T_2 = 3$，実行時間 $C_2 = 1$ の周期タスク τ_2 が存在し，$U_p = 0.5 + 0.33 = 0.83$ となる．また，$U_s = 1 - U_p = 0.17$ である．ティック 51 で発生した非周期タスク J_k は，$C_k^{WCET} = 4$ であり，実際の実行時間は $C_k^{ET} = 3$ であると仮定する．

図 2 (1) では，PET を 4 ティックと過大見積りした結果，デッドラインは $d_k^{PET} = d_k^{WCET} = 75$ となる．このデッドラインに従ってスケジュールした結果，非周期タスクはティック 57 で実行を開始し，中断，再開を繰り返し，ティック 70 で終了する．その応答時間は $70 - 51 = 19$ となる．

一方，図 2 (2) では，PET を 1 ティックと過小見積りした結果，最初のサブインスタンスは十分に早いデッドラインを与えられるため早くスケジュールされるが，残り実行は WCET を使用して再計算した $d_k^{WCET} = 75$ に基づいてスケジュールされるため，図 2 (1) と同様の中断，再開を繰り返し，やはり応答時間は 19 となる．

最後に図 2 (3) では，改良版適応型 TBS は非周期タスクの 3 つのサブインスタンスにそれぞれ，デッドライン $d_k^1 = 57$，$d_k^2 = 63$，$d_k^3 = 69$ を与えている．これらのデッドラインに従ってそれぞれのサブインスタンスがスケジュールされ，最後のサブインスタンスはティック 66 で開始し，

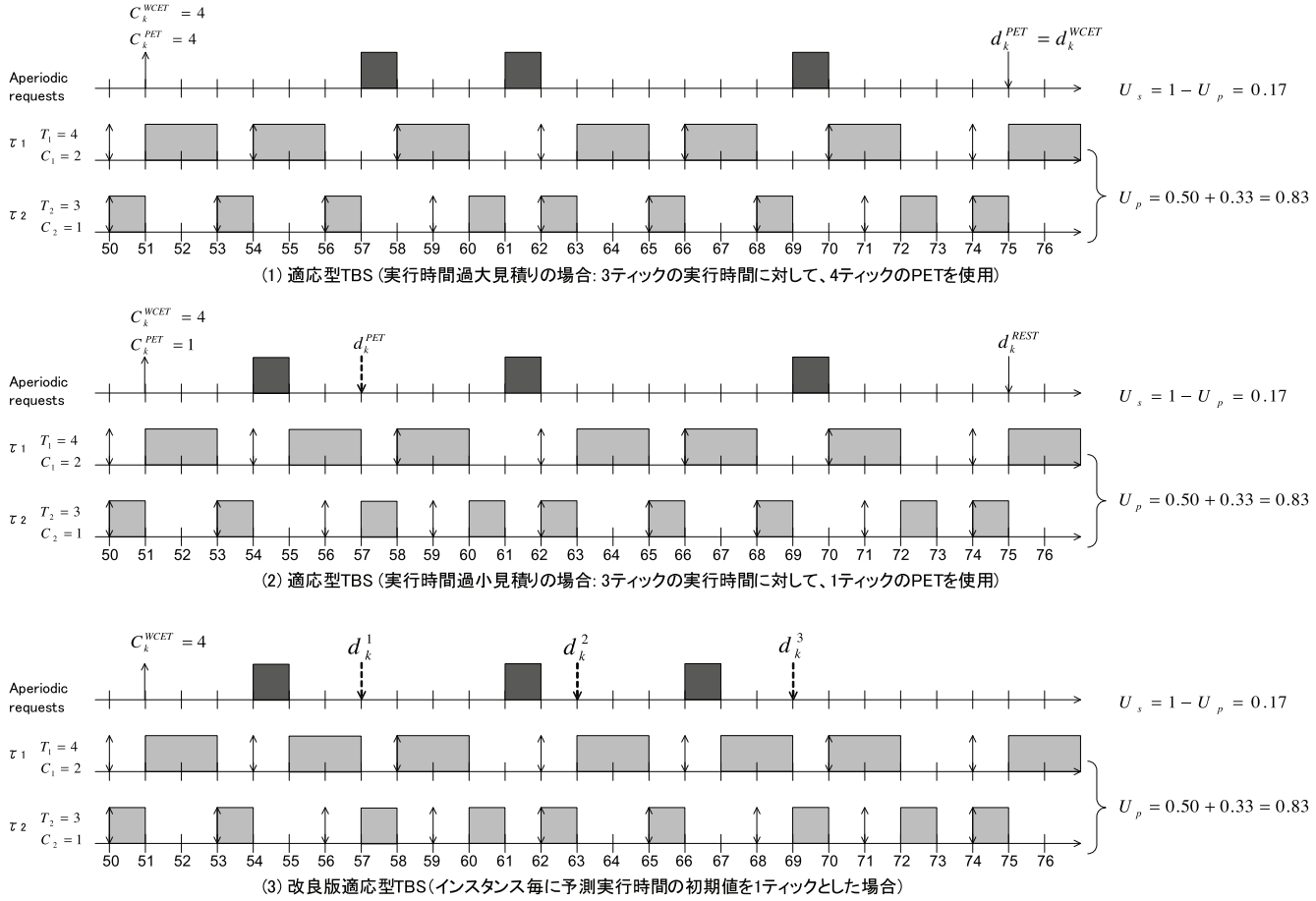

図 2 適応型 TBS と改良版適応型 TBS によるスケジュール例
Fig. 2 Example of schedules by the adaptive TBS and the improved adaptive TBS.

ティック 67 に終了する．したがって，当該タスクの応答時間は 67 − 51 = 16 となり，適応型 TBS よりも 3 ティック削減している．

この例から分かるように，予測実行時間の初期値を 1 ティックとした場合の改良版適応型 TBS は，適応型 TBS のように PET の過大見積り，あるいは過小見積りの影響を受けることがなく，最短の応答時間を与えることができる．

4.5 改良版適応型 TBS の評価

本節においてシミュレーションによる性能比較の結果を示す．性能として非周期タスクの平均応答時間を対象とし，評価対象方式はオリジナルの TBS（Original TBS），3 章で述べた既定案の（3.1 節の予測実行時間を使用する）適応型 TBS（Adaptive TBS（PET）），予測実行時間の初期値を 1 ティックとする改良版適応型 TBS（Adaptive TBS (1)），予測実行時間の初期値として BCET，BCET の 2 倍，4 倍，8 倍を使用する改良版適応型 TBS（Adaptive TBS (BCET*1〜8)）とする．BCET の取得方法として，同一非周期タスクが実行されるごとに，それまでの最も短かった実行時間を BCET とする方法をとった．なお，全方式にリソース回収法を適用した．

プロセッサ使用率（U_p）が 60%から 90%までの 5%おきとなる周期タスクセットを，各 U_p ごとに 10 セット用意した．また，観測時間（100,000 ティック）に対してトータルでのプロセッサ使用率が 2%程度となる非周期タスクセットを 10 セット用意した．したがって，周期タスクセットと非周期タスクセットを混在させることにより，各 U_p ごとに $10 \times 10 = 100$ 個のタスクセットが存在することになる．これらに対してシミュレーションを行い，その平均値を結果とする．

非周期サーバのプロセッサ使用率（バンド幅）は $U_s = 1 - U_p$ とする．周期タスクに関して，周期は平均 100 ティックの指数分布で決定し，最悪実行時間と実際の実行時間は同一とし，平均 10 ティックの指数分布で決定した．非周期タスクセットに関して，セット内で 4 種類のタスク（各タスクは複数回実行される）を用意し，各タスクの到着は平均で 1,000 ティックあたり 1.25 回となるポワソン到着で決定し，最悪実行時間は全タスク間で平均 8 となる指数分布で決定した．実際の実行時間はタスクごとに平均 4 となる指数分布で決定した．ここで，実際の実行時間が最悪実行時間よりも大きくならないように，実際の実行時間の上限を最悪実行時間とした．なお，最悪実行時間に対する実際の実行時間の比は全タスクセットの平均で約 0.33 となった．

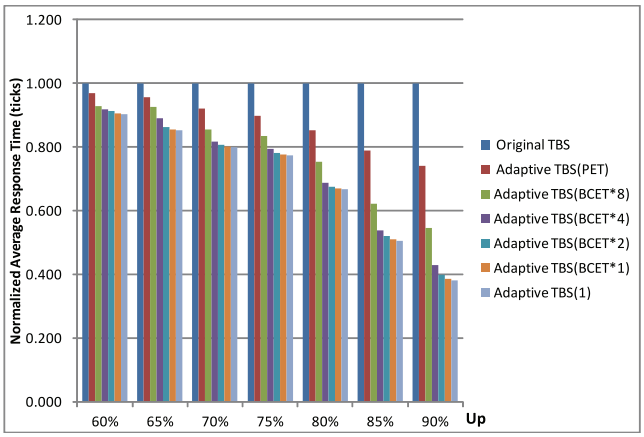

図 3 平均応答時間（オリジナル TBS，適応型 TBS，および改良版適応型 TBS）

Fig. 3 Average response times (Original TBS, Adaptive TBS, and Improved Adaptive TBS).

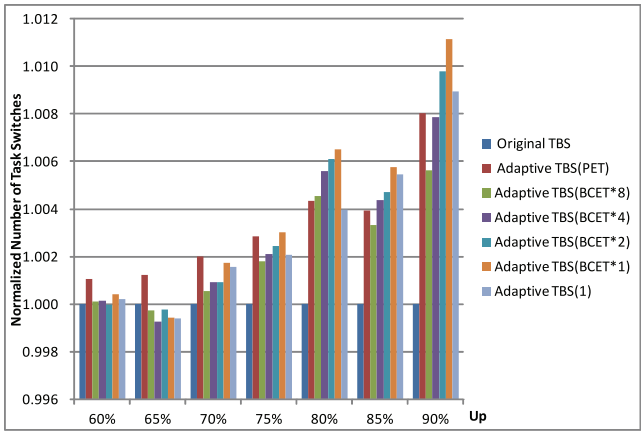

図 4 タスク切替え回数（オリジナル TBS，適応型 TBS，および改良版適応型 TBS）

Fig. 4 Number of task switches (Original TBS, Adaptive TBS, and Improved Adaptive TBS).

適応型 TBS 方式では，予測実行時間の算出における加重平均の重み係数（3.1 節における α）として 0.5 を使用した．

図 3 に結果を示す．図中の横軸は周期タスクのプロセッサ使用率（U_p），縦軸は非周期タスク実行の平均応答時間を Original TBS の結果で正規化した値で示している．U_p が 60%のときは，サーバのバンド幅（$U_s = 1 - U_p$）が十分に大きいために，方式間で応答時間の差が小さいが，U_p が大きくなるにつれて応答時間の差が顕著になっている．明らかに改良版の Adaptive TBS は，すべての場合で既存方式よりも良い結果となっているのが分かる．Adaptive TBS（1）は，最大では $U_p = 90\%$のときに Adaptive TBS（PET）と比較し 48.6%，Original TBS と比較し 62.0%の改善を達成している．Adaptive TBS（BCET*1）～Adaptive TBS（BCET*8）については，初期値が小さくなるほど短い応答時間を与えていることが分かる．これは，小さい初期値を使用することにより，過大見積りによる応答時間の悪化を防ぐ効果があることを示している．なお，本評価において，全シミュレーションの全タスク実行に関してデッドラインミスは発生しなかったことが確認されている．

以上の評価では，予測実行時間の初期値として，最短として 1 ティックと，BCET の 1～8 倍の値を使用した結果，値が小さいほど平均応答時間が短くなることが確認されたが，実際のシステムとして実現した場合には，タスク切替え頻度や，デッドライン計算のオーバヘッドが性能に影響する可能性がある．以下では，これらについて議論する．**図 4** に，観測時間におけるタスク切替え回数（Original TBS の値で正規化したもの）を示す（図では拡大して差を表すために縦軸を 0.996～1.012 の範囲としている）．この結果からは，方式（予測実行時間の初期値の与え方）とタスク切替え回数との間に特に関係はないことが分かる．また，最大でも 1.1%程度の差となっており，方式の違いによって性能に大きな影響を及ぼすことはないといえる．

表 1 デッドライン計算の回数（全タスクセットの平均）

Table 1 Number of deadline calculations (Average of all task sets).

Method	# of deadline calculations
Original TBS	500.9
Adaptive TBS（PET）	717.4
Adaptive TBS（BCET*8）	625.5
Adaptive TBS（BCET*4）	1,024.1
Adaptive TBS（BCET*2）	1,506.7
Adaptive TBS（BCET*1）	1,854.2
Adaptive TBS（1）	1,883.8

続いて，**表 1** に観測時間における非周期タスクのデッドライン計算の回数（全タスクセットの平均）を示す．非周期タスクの実行モデルは全 U_p で共通のもの（10 種類）を使用しているため，U_p による違いはない．表から，提案方式では予測実行時間の初期値が短くなるほど，デッドライン計算回数が増加することが分かる．これは，初期値を小さくすることで非周期タスクを分割したときのサブインスタンス数が増えるためである．

提案手法では，予測実行時間の初期値を 1 ティックとした場合，非周期タスクの実行中は毎ティックでデッドラインの再計算が必要となる．再計算は（式 (10) において）定数加算であるため，1 命令～数命令で計算可能である[*2]．仮に再計算に 10 命令の実行が必要であるとする．システムティックを 100 μ 秒，（スカラパイプライン）プロセッサの動作周波数を 100 MHz と仮定すると，キャッシュミスなどの要因を考慮しない限り，非周期タスクの実行中に $10/(100{,}000\,\text{ns}/10\,\text{ns}) = 0.1\%$の実行オーバヘッドが発生し，応答時間が 0.1%長くなる．提案方式による改善率と比較すると，再計算の影響は非常に小さいといえる．

[*2] 実際の命令数は命令セットやコンパイラに依存する．

続いて，デッドラインの再計算後のレディキュー挿入処理のオーバヘッドについて以下のように考察する．4.2 節の定義では，デッドラインの再計算の後，対象非周期タスクの制御ブロックがレディキューに挿入される．この挿入操作によるオーバヘッドが存在するが，周期タスクと非周期タスクで異なるレディキューを用意することで[*3]，非周期タスク用のレディキューの先頭に挿入することになるため[*4]，この挿入操作は $O(1)$ で可能である．これは，TBS 内で非周期タスク間の FCFS（First Come First Served）の関係が $d_k < d_{k+1}$ により保存されるためである．このことから，提案手法による付加的なレディキュー挿入操作によって，大きなオーバヘッドは発生しないといえる．

さらにレディキューへの挿入操作の回数を削減するために，デッドラインの再計算後，レディキューの先頭の周期タスクとのデッドラインの比較を行い，優先順位が逆転する場合にのみレディキュー挿入を行うことが有効である[*5]．本節のシミュレーションにおいて，再計算の回数が最も多いのが $U_p = 90\%$のときの Adaptive TBS（1）の場合で $1{,}884 - 501 = 1{,}383$ 回であったが，再計算された非周期タスクと先頭の周期タスクの間での優先順位の逆転は平均で 157 回発生していた．これは再計算回数の 11%に相当し，挿入操作のオーバヘッドが 89%削減可能であることを意味する．

実際の詳細なオーバヘッドの影響については実際のコードで実現されるスケジューラとタスクを使用して評価することで初めて明らかになるため，これが今後の課題である．

4.6 実行時間変動のタイプと適用方式の選択

本節では，タスクの実行時間変動の種類に対して，いずれの方式（オリジナル TBS，適応型 TBS，および改良版適応型 TBS）を適用すべきかについての定性的考察を行う．

実行時間が毎回 WCET と等しくなるタスクに対しては，オリジナル TBS は使用可能バンド幅に関して最適なデッドライン与える．改良版適応型 TBS も同様，最適なデッドラインを段階的に与えることが可能であるが，デッドライン計算が 1 回のみ行われるオリジナル TBS がオーバヘッドの観点から優位であることは明らかである．

しかし，スケーラビリティの保証の観点から，1 章で述べたように WCET は実際の実行時間に対して過大に見積もられることが一般的である．実行時間が WCET よりも小さく不変であるタスクに対しては，オリジナル TBS は十分に小さいデッドラインを与えることが不可能である．これに対し，過去の実行履歴から実行時間を予測する適応型 TBS が有効である．最初の数回の実行で定常的な実行時間と等しい予測実行時間を得た後は，最適なデッドラインを与えることになる．定常状態では予測実行時間が過大に見積もられることはなく，デッドライン計算は 1 回のみとなるため，段階的にデッドライン計算を行う改良版 TBS よりも優位である．

実行時間が変動するタスクの場合は，適用すべき方式はその変動の頻度に依存する．実行時間が変化する頻度が低いタスクの場合は，適応型 TBS では変化後の数回の実行は適切なデッドラインが得られないが，その割合が全実行回数に対して小さい限りオーバヘッドの観点から改良版適応型 TBS に対して優位である．一方，変化の頻度が高いタスクの場合は，適応型 TBS では予測実行時間の精度が低くなり，適切なデッドラインとならない．この場合は段階的に最適なデッドラインを与える改良版適応型 TBS が優位である．

以上から，提案する改良版適応型 TBS は，実行時間が頻繁に変化するタスクに対して有効であるといえる．

4.7 Constant Bandwidth Server（CBS）との定性的比較

オリジナル TBS では，タスクの実行時間が WCET よりも短い場合に最適なデッドラインが得られず，短い応答時間を得ることができない．（改良版）適応型 TBS はこの問題点を解決する手法であるが，異なる方法で実行時間の変動に対処する方法として Constant Bandwidth Server（CBS）[12] がある．CBS は周期を持ち，周期あたりの実行可能時間をサーバ容量（budget）とし，サーバ管理下のタスクをサーバ容量の時間範囲内で実行する．タスクのデッドラインは基本的に周期の長さで設定される．タスク実行時間がサーバ容量を超える場合，サーバ容量の補充を行うと同時に，デッドラインを 1 周期の長さだけ延長する．

サーバ容量としては必ずしも最悪実行時間に相当する大きさを想定する必要はなく，平均的な実行時間を使用することが効果的である．タスクの早期終了によりサーバ容量が残る場合は，容量の有効期限内である限り，次のタスク実行に費やすことが可能である（以降，残余サーバ容量をスラックと呼ぶ）．これは 2 章で述べた TBS におけるリソース回収と同じ機能である．この点に着目すると，TBS と CBS では「先行タスク実行の早期終了によるスラックを後続タスク実行が使用する」という意味で本質的には同じものであるが，TBS が明示的なリソース回収の計算を要することに対し，CBS ではサーバ容量の範囲内で自然に（暗黙的に）行われる．このことが CBS の利点である．

ただし，ここでのリソース/スラック回収の適用先は後続タスク実行に限定される．CBS のサーバ容量には周期に基づく有効期限があるため，後続タスクの到着が期限切れ後の場合はスラック適用が不可能である．これに対し，（改

[*3] この場合，スケジューリングの際に 2 つのレディキューの先頭タスクどうしのデッドラインを比較することになる．
[*4] EDF アルゴリズムを実現する場合，キュー内でデッドライン時刻に関して昇順となるようにタスク制御ブロックが接続される．
[*5] 文献 [3] におけるリアルタイム OS カーネルでも，タイマハンドラが同様の方法をとっている．

良版）適応型 TBS は，後続タスクのスラック利用に関しては同様の限界があるが，タスク実行の早期終了（スラック発生）をあらかじめ予測し，これを事前に早期デッドラインの形で反映させることによる，「スラックを発生させるタスク実行自身によるスラックの利用」を実現している．この点は有効期限の問題がなく，CBS に対する優位性となっている．

CBS ではサーバ内のすべてのタスク実行に対して，周期の長さに相当するデッドラインが設定される．したがって，同一タスクが繰り返し実行された場合，個々の実行時間が異なったとしても，CBS では周期に基づいたデッドラインが設定される．平均実行時間に基づくサーバ容量を仮定すると，平均よりも短い実行にとっては過大なデッドラインとなる．平均よりも長い実行にとっては，実行時間が周期の倍数でない限りは最適なデッドラインとはならない．これに対し，改良版適応型 TBS では，実行ごとに独立して，実際の実行時間に基づく最適なデッドライン設定が可能であることが特長である．

以上のことから，発生間隔が CBS の周期よりも長く，実行時間が周期の倍数とならない非周期タスクに対して，定性的に CBS よりも改良版適応型 TBS のほうが応答時間に関して優位である．

5. おわりに

本論文では，ハードデッドラインを持つ周期タスクとデッドラインを持たない非周期タスクが混在するリアルタイムシステムのためのタスクスケジューリング方式である適応型 TBS において，非周期タスクの予測実行時間が過大あるいは過小見積りされた場合は最適なデッドラインが与えられず，十分な効果が得られないことに着目し，その弱点を補う方式として，改良版適応型 TBS を提案した．提案する方式では，非周期タスクは 1 ティックの実行ごとにデッドラインが再計算され，実行時間の変動に対して最適なデッドラインでスケジュールされることになる．シミュレーションによる評価では，特に高負荷のときに提案手法の効果が大きく，適応型 TBS に対して最大 48.6%，オリジナル TBS に対して最大 62.0%の平均応答時間の改善効果が確認された．

改良版適応型 TBS 方式は，実行時間が高頻度で変動する場合も最適なデッドラインを提供することが可能である．一方，非周期タスクが実行される限り，ティックごとにデッドラインの再計算のためのオーバヘッドが存在する．これを改善するため，予測実行時間の初期値として，BCET に基づく値を使用する方法をあわせて提案した．初期値をある程度大きくすることにより定性的にオーバヘッドが小さくなることが期待できる．

今回の評価は実行時間や到着時刻に関して確率的に生成したタスクセットに対するものであったが，実際の実行時間の変動を表現するためには，実プログラムを使用した評価が望まれる．加えて，デッドライン計算を含めたスケジューリングオーバヘッドを考慮した評価を行う予定である．また，実行時間を予測して使用するもう 1 つのリアルタイムスケジューリングアルゴリズムとして，著者は過去に適応型 EDF を提案している [8], [18]．これに対して，本論文で提案した実行時間見積り方法を適用し，評価する予定である．

また，本論文における提案方式は 1 つのサーバの性能向上を目指すものであったが，CBS を基本サーバとし，複数のサーバ間でスラック回収・適用を行う研究がある [19], [20], [21], [22]．これらはあるサーバのタスク実行（特にハードタスク）の早期終了によるスラックを，ソフト/非リアルタイムタスクのためのサーバが利用することを目的としている．EDF を前提とするサーバ方式であれば，デッドラインが関連付けられたスラックをサーバ間で授受することが可能であるため，これらのサーバ間スラック回収の仕組みを（改良版）適応型 TBS と組み合わせることにより，応答時間のさらなる短縮が見込める．これについては今後の課題である．

参考文献

[1] de Niz, D. Lakshmanan, K. and Rajkumar, R.: On the Scheduling of Mixed-Criticality Real-Time Task Sets, *Proc. Real-Time Systems Symposium*, pp.291–300, IEEE Computer Society (2009).

[2] Baruah, S., Li, H. and Stougie, L.: Towards the Design of Certifiable Mixed-Criticality Systems, *Proc. Real-Time and Embedded Technology and Application Symposium*, pp.13–22, IEEE Computer Society (2010).

[3] Buttazzo, G.C.: *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd Edition, Springer (2011).

[4] Spuri, M. and Buttazzo, G.C.: Efficient Aperiodic Service under Earliest Deadline First Scheduling, *Proc. Real-Time Systems Symposium*, pp.2–11, IEEE Computer Society (1994).

[5] Liu, C.L. and Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the Association for Computing Machinery*, Vol.20, No.1, pp.46–61 (1973).

[6] Lundqvist, T. and Stenström, P.: Timing Anomalies in Dynamically Scheduled Microprocessors, *Proc. Real-Time Systems Symposium*, pp.12–21, IEEE Computer Society (1999).

[7] Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P., Staschulat, J. and Stenström, P.: The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools, *ACM Trans. Embedded Computing Systems*, Vol.7, No.3, pp.1–53 (2008).

[8] 田中清史：実行時間の変動を利用するリアルタイムスケジューリング，情報処理学会研究報告，Vol.2013-EMB-28, No.25 (2013).

[9] Cervin, A. and Eker, J.: Feedback Scheduling of Control Tasks, *Proc. 39th IEEE Conference on Decision and*

Control, Vol.5, pp.4871–4876 (2000).
[10] Tanaka, K.: Adaptive Total Bandwidth Server: Using Predictive Execution Time, *Proc. 4th IFIP TC10 International Embedded Systems Symposium* (*IESS 2013*), pp.250–261, Springer (2013).
[11] Tanaka, K.: Adaptive Real-Time Scheduling for Soft Tasks with Varying Execution Times, *Journal of Information Processing*, Vol.22, No.2, pp.152–159 (2014).
[12] Abeni, L. and Buttazzo, G.: Integrating Multimedia Applications in Hard Real-Time Systems, *Proc. Real-Time Systems Symposium*, pp.4–13, IEEE Computer Society (1998).
[13] Lehoczky, J.P., Sha, L. and Strosnider, J.K.: Enhanced Aperiodic Responsiveness in Hard Real-Time Environments, *Proc. Real-Time Systems Symposium*, pp.261–270, IEEE Computer Society (1987).
[14] Sprunt, B., Sha, L. and Lehoczky, J.: Aperiodic Task Scheduling for Hard-Real-Time Systems, *The Journal of Real-Time Systems*, Vol.1, No.1, pp.27–60 (1989).
[15] Lehoczky, J.P. and Ramos-Thue, S.: An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems, *Proc. Real-Time Systems Symposium*, pp.110–123, IEEE Computer Society (1992).
[16] Spuri, M. and Buttazzo, G.: Scheduling Aperiodic Tasks in Dynamic Priority Systems, *The Journal of Real-Time Systems*, Vol.10, No.2, pp.179–210 (1996).
[17] Spuri, M., Buttazzo, G. and Sensini, F.: Robust Aperiodic Scheduling under Dynamic Priority Systems, *Proc. Real-Time Systems Symposium*, pp.210–219, IEEE Computer Society (1995).
[18] Tanaka, K.: Adaptive EDF: Using Predictive Execution Time, *Proc. 5th Workshop on Adaptive and Reconfigurable Embedded Systems* (*APRES*), pp.2–5 (2013).
[19] Caccamo, M., Buttazzo, G. and Thomas, D.: Efficient Reclaiming in Reservation-Based Real-Time Systems with Variable Execution Times, *IEEE Trans. Computers*, Vol.54, No.2, pp.198–213 (2005).
[20] Lin, C. and Brandt, S.: Improving Soft Real-Time Performance through Better Slack Reclaiming, *Proc. Real-Time Systems Symposium*, pp.410–421, IEEE Computer Society (2005).
[21] Marzario, L., Lipari, G., Balbastre, P. and Crespo, A.: IRIS: A New Reclaiming Algorithm for Server-Based Real-Time Systems, *Proc. Real-Time and Embedded Technology and Applications Symposium*, pp.211–218 (2004).
[22] Kato, S., Ishikawa, Y. and Rajkumar, R.: CPU Scheduling and Memory Management for Interactive Real-Time Applications, *Real-Time Systems*, Vol.47, No.5, pp.454–488 (2011).

田中 清史 （正会員）

1971 年生．2000 年東京大学大学院理学系研究科情報科学専攻博士課程修了，博士（理学）．2001 年より北陸先端科学技術大学院大学助教授（現准教授）．並列計算機アーキテクチャ，プロセッサアーキテクチャ，メモリシステム，リアルタイム組込みシステムの研究に従事．電子情報通信学会，IEEE，ACM 各会員．