Vol.2013-SE-180 No.9
Vol.2013-EMB-29 No.9
2013/5/28

# 適応型リアルタイムスケジューリングのための実行時間の見積法

田中 清史[1,a)]

**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．本稿では，応答時間の短縮を目的として著者が過去に提案した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方法を改善する手法を提案する．これにより，実行時間の見積りの精度に依存せずに応答時間短縮の効果が得られる．評価では，提案改善手法により非周期リクエストの平均応答時間が最大 48.6%短縮されることが確認された．

# A Method of Estimating Tasks' Execution Times for Adaptive Real-Time Scheduling

Kiyofumi Tanaka[1,a)]

***Abstract:*** In real-time scheduling methods that take tasks' execution times into account, worst-case execution times are assumed to be spent for tasks' execution. However, in actual systems, tasks finish, in most cases, in shorter execution times than their worst-case execution times. In this paper, for the adaptive real-time scheduling that the author proposed in the previous research to shorten response times, a method of improving the estimation of tasks' execution times is proposed. By this method, benefit of the adaptive scheduling can be obtained independent of the prediction accuracy of execution times. In the evaluation, the improved method reduced average response times of aperiodic requests by 48.6%, at a maximum.

## 1. はじめに

近年の組込みシステムの複雑化と多様化に伴い，異なる性質，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]．例えば，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等のタスクの混在が挙げられる．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て，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．

[1] 北陸先端科学技術大学院大学
JAIST, Asahidai 1-1, Nomi, Ishikawa 923–1292, Japan
[a)] kiyofumi@jaist.ac.jp

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 の正確な見積もりは困難になっている．例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，全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，実際のタスク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]．この方式はタスクが繰り返し実行され，実行の度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．本稿ではまず，2 節で関連研究を述べ，3 節において適応型 TBS の概要を述べる．続いて 4 節で考察を通して，応答時間を更に短縮するために実行時間の見積方法を見直し，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 節で本稿をまとめる．

## 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 [10]，Priority Exchange [10]，Sporadic Server [11]，Slack Stealing [12] 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 [9] などがある．これらのアルゴリズムは全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一つであり，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], [13]．前提として，ハードタスクは周期的に起動され，周期と一致する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．

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

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（一つ前）のインスタンスのデッドライン，$C_k$ 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毎に，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仮のデッドラインを与えられた後，全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 が過大見積も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 [14] 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．

リソース回収を伴う 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．

$$d'_k = \overline{r}_k + \frac{C_k}{U_s} \quad (2)$$

$\overline{r}_k$ は以下で計算される値である．

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

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える．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．

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

Vol.2013-SE-180 No.9
Vol.2013-EMB-29 No.9
2013/5/28

## 3. 適応型 TBS

本節では，実行時間の変動を考慮したアルゴリズムとして，過去に提案された適応型 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] では以下の方法を採っている．

$$C_{i_k}^{PET} = \alpha \times C_{i_{k-1}}^{PET} + (1-\alpha) \times C_{i_{k-1}}^{ET} \quad (5)$$
$$C_{i_0}^{PET} = C_i^{WCET}$$

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 $\alpha$ 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．

### 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二つに分解される．それらを異なるタスクとみなすことによって，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 $J_k$ の実行を二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，二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．

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

$$d_k^{REST} = d_k^{PET} + \frac{C_k^{REST}}{U_s} \quad (7)$$

（オリジナルの）TBS のデッドライン計算は以下の通りであった．

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

$C_k^{REST} = C_k^{WCET} - C_k^{PET}$ と式（6），（7），（8）から，

$$\begin{aligned} d_k^{REST} &= max(r_k, d_{k-1}) + \frac{C_k^{PET}}{U_s} + \frac{C_k^{WCET} - C_k^{PET}}{U_s} \\ &= max(r_k, d_{k-1}) + \frac{C_k^{WCET}}{U_s} \quad = \quad d_k \end{aligned}$$

したがって，非周期タスクが到着した際に，式（6）と式（8）により二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毎に定数となり，システム稼働前に一度計算するのみで良いため，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．

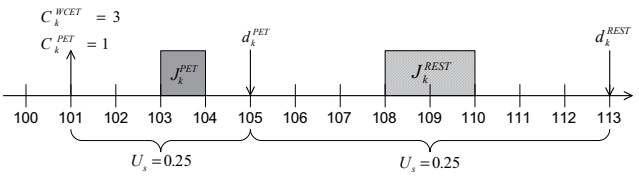


**図 1** 適応型 TBS におけるデッドライン割当て．
**Fig. 1** Deadline assignment in the adaptive TBS.

[*1] 実際には $0 \le C_k^{REST} \le C_k^{WCET} - C_k^{PET}$ 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．

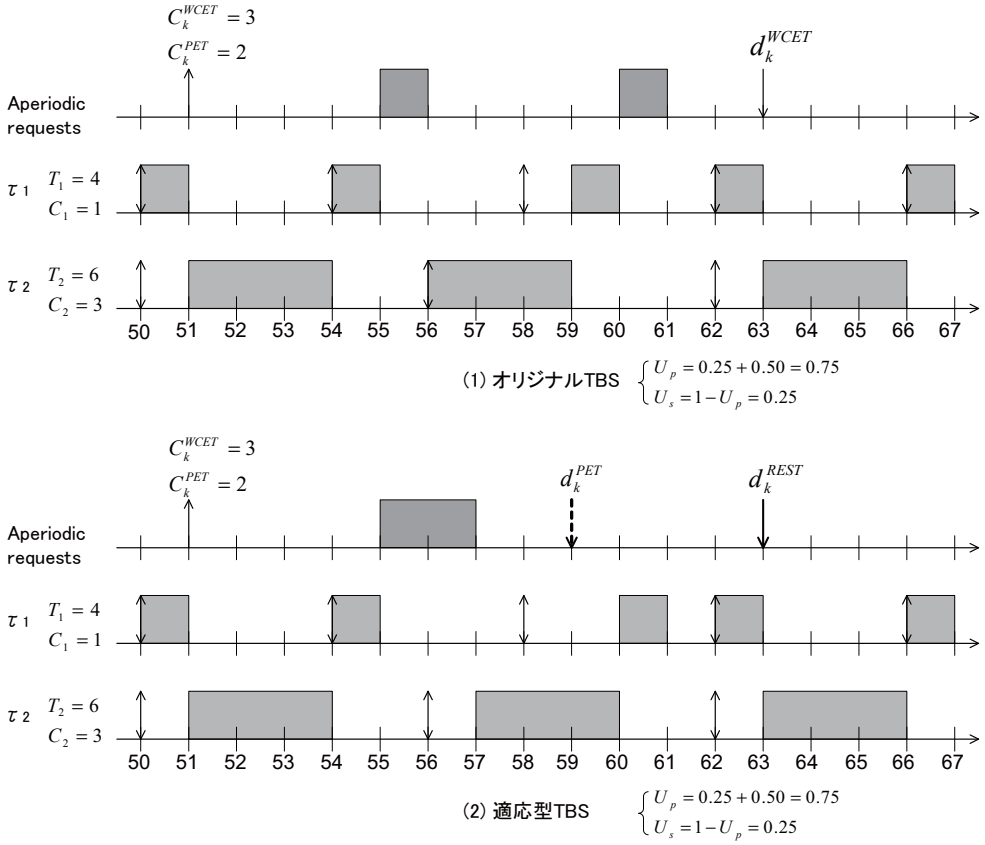


**図 2** オリジナル TBS と適応型 TBS によるスケジュール例.
**Fig. 2** Example of schedules by the original TBS and adaptive TBS.

### 3.3 適応型 TBS の例

適応型 TBS のスケジュール例を示す．**図 2** において，(1) と (2) はそれぞれオリジナル TBS，適応型 TBS によるスケジューリング結果を示している．二つの周期タスク $\tau_1$，$\tau_2$ と，ティック 51 で到着する非周期リクエストがある．$\tau_1$ の周期は $T_1 = 4$ であり，実行時間は $C_1 = 1$ である．$\tau_2$ の周期は $T_2 = 6$ であり，実行時間は $C_2 = 3$ である[*2]．したがって，二つの周期タスクによるプロセッサ使用率は $U_p = 0.25 + 0.5 = 0.75$ であり，非周期サーバのバンド幅は $U_s = 1 - U_p = 0.25$ となる．

非周期リクエストの WCET は 3, 予測実行時間と実際の実行時間は 2 であると仮定する．オリジナル TBS では，この非周期タスクのデッドラインは $d_k^{WCET} = 51 + 3/0.25 = 63$ となる．EDF アルゴリズムに基づいて，この非周期タスクはティック 55 で実行を開始し，ティック 56 で中断し，$\tau_2$ と $\tau_1$ の実行を挟んでティック 60 で再開し，そしてティック 61 で終了する．結果的に，応答時間は $61 - 51 = 10$ となる．一方，適応型 TBS では，二つのデッドラインとして $d_k^{PET} = 51 + 2/0.25 = 59$ と $d_k^{REST} = d_k^{WCET} = 63$ が与えられる．EDF にしたがって，非周期タスクはティック 55 で実行を開始し，ティック 57 で終了する．応答時間は $57 - 51 = 6$ となる．この例では，適応型 TBS はオリジナル TBS と比較し，応答時間を 4 ティック短縮したことになる．

非周期タスクの実際の実行時間が 3 ティックであるとし，その他は同じ条件でタスクセットをスケジュールする場合を考える．オリジナル TBS では，当該非周期実行はティック 62 で終了する．一方，適応型 TBS ではティック 57 で非周期実行を中断し，ティック 61 で再開し，ティック 62 で終了する．このように，たとえ実行時間が間違って（この例では過小評価に）予測された場合でも，応答時間はオリジナル TBS のものと同じか短くなる．

[*2] この例では，$\tau_1$ と $\tau_2$ の実行時間は不変であると仮定する．

### 3.4 適応型 TBS のスケジューラビリティ

非周期リクエストを二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，二つの（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，二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 [13] 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.

### 3.5 実装の複雑さ

提案スケジューリングアルゴリズムでは，非周期タスクの実行インスタンスは二つのサブインスタンスに分解される．しかし，オペレーティングシステムはタスクを一つの情報セット（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．この点が実装上オリジナル TBS と唯一異なる部分である．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採る自然な手続きである．加えて，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，必要な時に使用するべきである．

### 3.6 リソース回収法の適用

TBS では，$k$ 番目の非周期リクエストのデッドラインが計算されるときに，$d_{k-1}$ が必要である．適応型 TBS では，一つ前（$k-1$ 番目）の非周期実行は二つのサブインスタンスに分割されるため，その二つ目のサブインスタンスのデッドライン（$d_{k-1}^{REST}$）が計算に使用される．$k-1$ 番目のリクエストの実行が PET（$C_{k-1}^{PET}$）以内に終了したときは，二つ目のサブインスタンスは実行されない．この場合，$k$ 番目のタスクのデッドライン計算に，$d_{k-1}^{REST}$ の代わりに $d_{k-1}^{PET}$ が使用可能である．これは適応型 TBS のための第一のリソース回収法である．

2 節で述べたより貪欲な方法 [14] 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に関わらず，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．これは適応型 TBS にとって第二のリソース回収法となりうる．この方法は自然に前述の第一のリソース回収法を含んでいる．文献 [8] ではリソース回収法については言及されず，評価でも使用されていなかった．本稿の評価では，第二のリソース回収法を採り入れる．

## 4. 実行時間見積法とアルゴリズムの改良

本節では，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（あるいは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，デッドラインミスが発生し，残りの実行は WCET に基づくデッドラインにしたが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デッドラインミスを発生させる．デッドラインミス発生時，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，デッドラインミスを発生させる．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．

### 4.2 改良版適応型 TBS の定義

改良版適応型 TBS では，非周期タスクは一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みなすことにより，TBS と同様な方法となる．

$k$ 番目の非周期タスクのインスタンス $J_k$ の実行をサブインスタンス $J_k^0, J_k^1, J_k^2, \ldots$ に分割する．$J_k^0$ 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 $i$ ティック実行後から $i+1$ 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，$J_k^i$ 以降のサブインスタンスは存在しないことになる．

$k$ 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，サブインスタンス $J_k^i$ には以下のデッドラインが与えられる．

$$d_k^i = max(r_k, d_{k-1}) + \frac{1}{U_s} \quad (i = 0) \tag{9}$$

$$d_k^i = d_k^{i-1} + \frac{1}{U_s} \quad (i \geq 1) \tag{10}$$

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

分に関してはリソース回収の適用が可能である．

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 1 ティック実行される度に，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．明らかに，この方法では適応型 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．何故なら，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，全てのサブインスタンスを異なるタスク実行とみなせば，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 3** において，上段の（1）が，PET が過大見積りされた場合の適応型 TBS，中段の（2）が，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 のスケジュール例を示している．全ての方式において，周期 $T_1 = 4$，実行時間 $C_1 = 2$ の周期タスク $\tau_1$，周期 $T_2 = 3$，実行時間 $C_2 = 1$ の周期タスク $\tau_2$ 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．

（1）では，PET を 4 ティックと過大見積もりした結果，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したが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（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，（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（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 $d_k^0 = 57$，$d_k^1 = 63$，$d_k^2 = 69$ を与えている．これらのデッドラインにしたが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，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．

この例からわかるように，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，評価方式は 2 節で述べたリソース回収法を適用した TBS（Original TBS），3.6 節で述べたリソース回収法を適用した適応型 TBS（Adaptive TBS），および同じくリソース回収法を適用した改良版適応型 TBS（Improved ATBS）とする．

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 $U_p$ 毎に 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 $U_p$ 毎に $10 \times 10 = 100$ 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毎に平均 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．

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

**図 4** 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，方式間でほぼ同一の応答時間となっている．70%以上では徐々に応答時間の差が表れている．明らかに Improved ATBS は，全ての場合

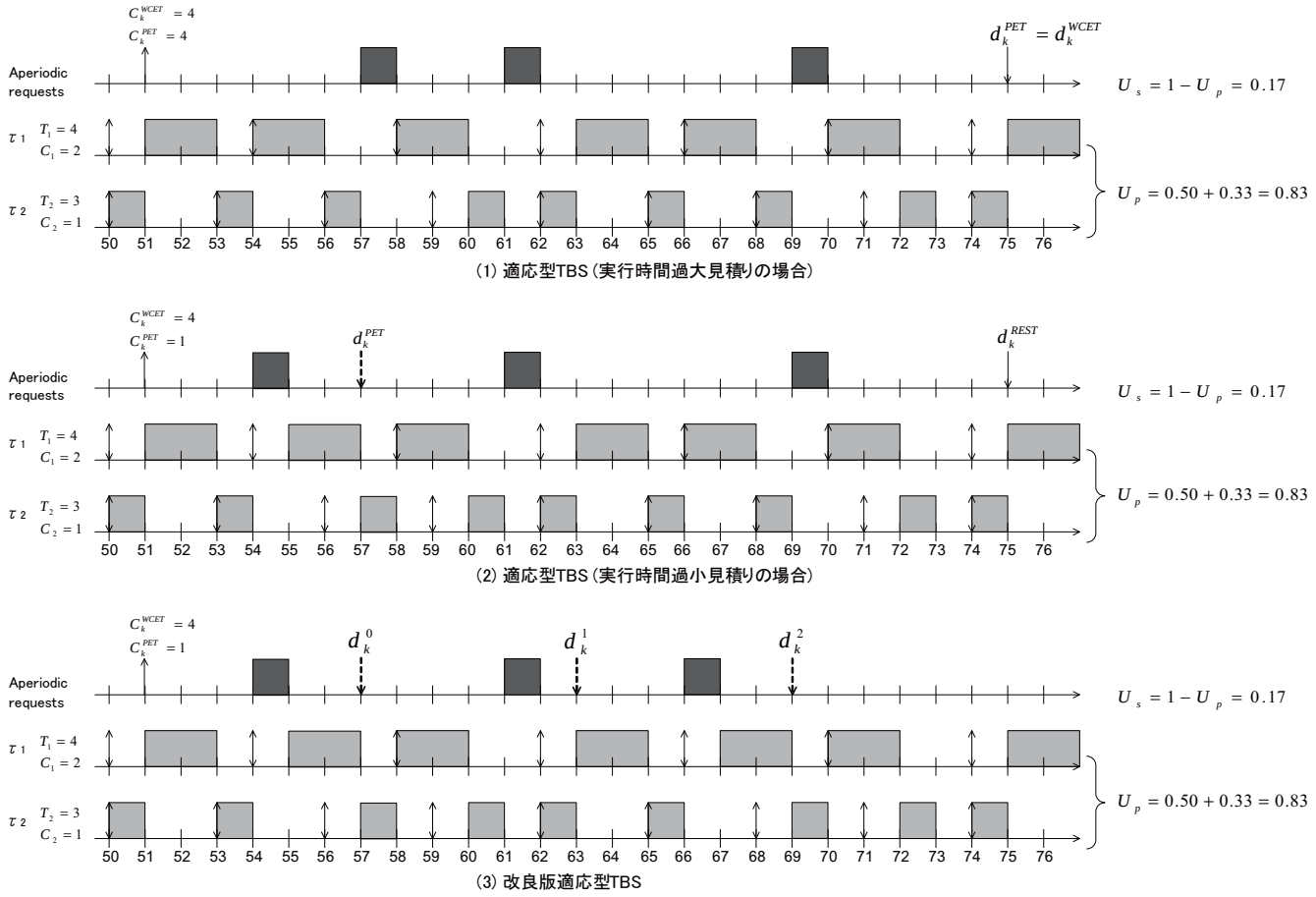


**図 3** 適応型 TBS と改良版適応型 TBS によるスケジュール例.
**Fig. 3** Example of schedules by the adaptive TBS and the improved adaptive TBS.

で他の方式よりも良い結果となっているのがわかる．最大では，$U_p = 90\%$のときに適応型 TBS と比較し 48.6%，オリジナル TBS と比較し 62.0%の改善を達成している．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．

## 5. おわりに

本稿では，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 ティックの実行毎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，実行時間が変動した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，非周期タスクが存在する限り，ティック毎にデッドラインの再計算のオーバヘッドが存在する．しかし，式（10）からわかるように，$1/U_s$ の項はあらかじめ計算しておき，定数として扱うことができるため，大きなオーバヘッドとはならないことが予想される．これに対し，適応型 TBS では式（6）の除算のオーバヘッドは避けられなかった．したがって，改良版方式のほうがオーバヘッドが小さくなる可能性がある．

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一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], [15]．これに対して，本稿で提案した実行時間見積方法を適用し，評価する予定である．

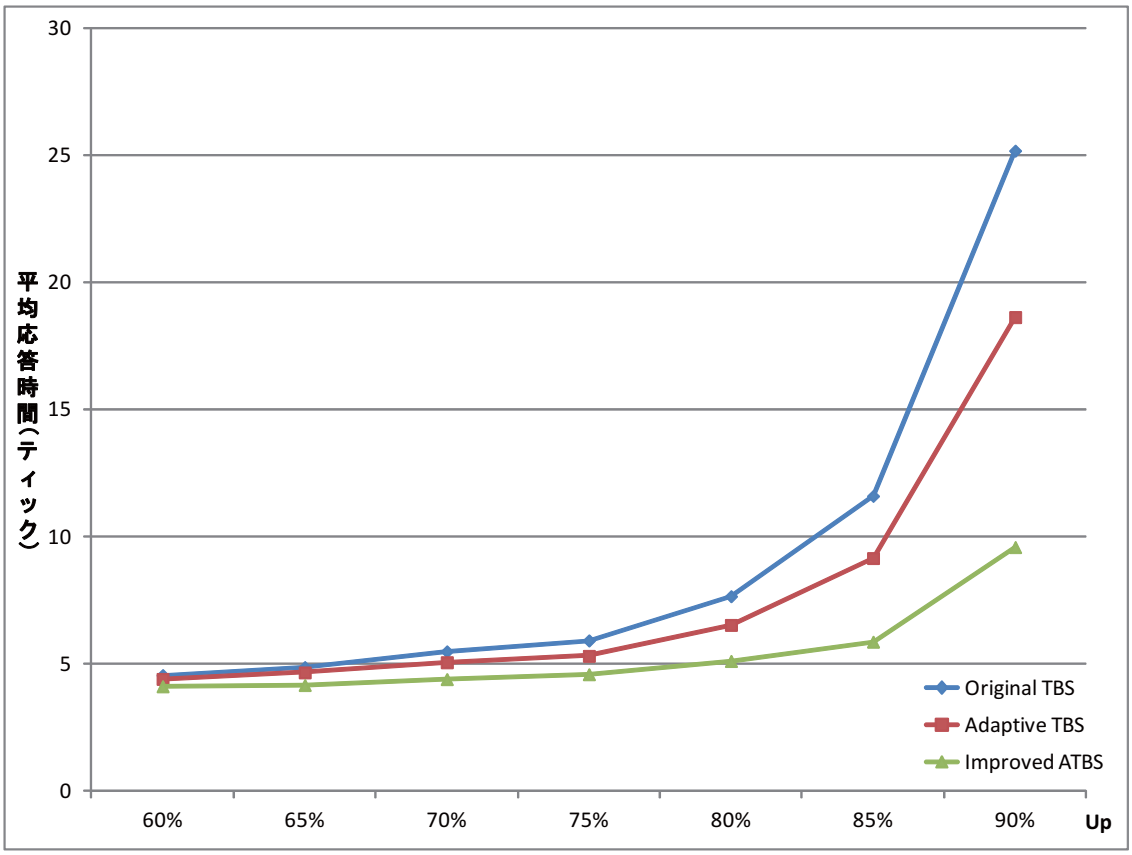


**図 4** 平均応答時間（オリジナル TBS，適応型 TBS，および改良版適応型 TBS）.
**Fig. 4** Average response times (Original TBS, Adaptive TBS, and Improved ATBS.

## 参考文献

[1] de Niz, D. Lakshmanan, K., Rajkumar, R.: *On the Scheduling of Mixed-Criticality Real-Time Task Sets*, Proc. of IEEE Real-Time Systems Symposium, pp.291–300, IEEE Computer Society, Washington, DC, (2009).
[2] Baruah, S., Li, H., Stougie, L.: *Towards the Design of Certifiable Mixed-Criticality Systems*, Proc. of IEEE Real-Time and Embedded Technology and Application Symposium, pp.13–22, IEEE Computer Society, Stockholm, (2010).
[3] Buttazzo, G.C.: *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Third Edition, Springer, (2011).
[4] Spuri, M., Buttazzo, G.C.: *Efficient Aperiodic Service under Earliest Deadline First Scheduling*, Proc. of IEEE Real-Time Systems Symposium, pp.2–11, IEEE Computer Society, San Juan (1994).
[5] Liu, C.L., 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., Stenström, P.: *Timing Anomalies in Dynamically Scheduled Microprocessors*, Proc. of IEEE Real-Time Systems Symposium, pp.12–21, IEEE Computer Society, Phoenix (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., Stenström, P.: *The Worst-Case Execution Time Problem – Overview of Methods and Survey of Tools*, ACM Trans. on 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, 対馬 (2013).
[9] Abeni, L., Buttazzo, G.: *Integrating Multimedia Applications in Hard Real-Time Systems*, Proc. of IEEE Real-Time Systems Symposium, pp.4–13, IEEE Computer Society, Madrid (1998).
[10] Lehoczky, J. P., Sha, L., Strosnider, J. K.: *Enhanced Aperiodic Responsiveness in Hard Real-Time Environments*, Proc. of IEEE Real-Time Systems Symposium, pp.261–270, IEEE Computer Society, San Jose (1987).
[11] Sprunt, B.m, Sha, L., Lehoczky, J.", *Aperiodic Task Scheduling for Hard-Real-Time Systems*, The Journal of Real-Time Systems, Vol.1, No.1, pp.27–60 (1989).
[12] Lehoczky, J. P., Ramos-Thue, S.: *An Optimal Algorithm for Scheduling Soft-Aperiodic Tasks in Fixed-Priority Preemptive Systems*, Proc. of IEEE Real-Time Systems Symposium, pp.110–123, Phoenix (1992).
[13] Spuri, M., Buttazzo, G.: *Scheduling Aperiodic Tasks in Dynamic Priority Systems*, The Journal of Real-Time Systems, Vol.10, No.2, pp.179–210 (1996).
[14] Spuri, M., Buttazzo, G., Sensini, F.: *Robust Aperiodic Scheduling under Dynamic Priority Systems*, Proc. of IEEE Real-Time Systems Symposium, pp."210–219, Pisa (1995).
[15] Tanaka, K.: *Adaptive EDF: Using Predictive Execution Time*, Proc. of 5th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES), pp.2–5, Philadelphia (2013).