Vol.2013-SLDM-160 No.25
Vol.2013-EMB-28 No.25
2013/3/14

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

田中 清史[1,a)]

概要：リアルタイムスケジューリング理論では，タスクの実行は最悪実行時間を費やすものと仮定するのが一般的である．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，実際の実行時間を予測し，予測した時間を利用するスケジューリングにより，応答時間を短縮する 2 つの方法を提案する．評価では，提案手法により重要度の高い周期タスクの平均応答時間が最大 13.4%，非周期リクエストの平均応答時間が最大 32.0%短縮されることが確認された．

Real-Time Scheduling by Exploiting Fluctuation in Execution Time

Kiyofumi Tanaka[1,a)]

Abstract: In real-time scheduling theory, it is supposed that task execution spends its worst-case execution time. However, in actual systems, tasks finish, in most cases, in shorter execution time than their worst-case execution time. In this paper, two methods of reducing response times by predicting actual execution times and exploiting the predicted execution times are proposed. In the evaluation, the proposed methods reduced average response times of an important periodic task and aperiodic requests by 13.4% and 32.0%, respectively.

1. はじめに

組込みシステムの複雑化と多様化に伴い，リアルタイムスケジューリングの重要性が増している．完全なリアルタイム性が要求される制御タスク（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: Worst-Case Execution Time）の実行を想定してプロセッサ使用率を算出することにより，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．

周期タスクセットを対象とする代表的なスケジューリングアルゴリズムとして，Rate Monotonic（RM）と Earliest Deadline First（EDF）がある [1]．RM は固定優先度アルゴリズムの一つであり，周期の短いタスクに高い優先度を与え，優先的に実行する．RM では優先度が高いタスクのジッタや応答時間が小さくなる特長があるが，スケジューラビリティを確保しつつプロセッサを 100%使用することが不可能である[*1]．EDF は動的優先度アルゴリズムの一つであり，デッドライン時刻の近いタスクに高い優先度を与え，優先的に実行する．100%の使用率の下でスケジューラビリティが保証される特長があるが，タスクに静的な優先度を与えることができないため，重要度の高いタスクのジッタや平均応答時間を小さく保つことが困難である．

本研究では EDF をベースとしたアルゴリズムを探求し，システムにとって重要な特定タスク，および非周期タスクの応答時間を短縮することを目的とする．

2. 予測実行時間

一般にリアルタイムスケジューリングおよびスケジューラビリティ解析では，タスクの実行時間として最悪実行時間（WCET）が想定される [2]．タスクの実行時間は未知であるため，特にハードリアルタイムシステムでは WCET の仮定が強いられる．

現在はプロセッサやプログラムの複雑化により，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

[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

[*1] 理論的には $\ln 2 = 69\%$ が上限である [1].

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，キャッシュのヒット／ミスの予測は困難を極める [3]．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 [4]．結果的に，安全のため WCET は悲観的に見積もらざるをえず，ほとんどの場合で実際の実行時間とのギャップが大きい．

その影響の例として，仮に SJF（Shortest Job First）アルゴリズムで，最悪実行時間に基づいてスケジューリングした場合，実際の実行時間でスケジューリングするもの（非現実的であるが）と比較すると，平均ターンアラウンドタイムが最適とならないことは容易に想像できる．**図 1** では，最悪実行時間の情報を SJF に適用すると，図中の（1）のように，タスク A（WCET=2），タスク B（WCET=3），タスク C（WCET=4）の順で実行されることになる．実際の実行時間がタスク A が 2，タスク B が 1，タスク C が 2 であるとすると，WCET 情報による実行順を適用した場合は図中の（2）のように，平均ターンアラウンドタイムは 3.3 となる．一方，実際の実行時間が既知であるとすると，それにしたがって SJF を適用すると図中の（3）のような実行順となり，平均ターンアラウンドタイムは 3 となる．このように，最悪実行時間に基づく判断は，実際の実行では最適とならない場合がある．

EDF におけるスケジューラビリティ解析では実行時間が考慮される．タスクによるプロセッサ使用率の総和が 100%以下ならば，スケジュール可能であることが保証される．この解析では，タスクの実行時間として最悪実行時間が仮定されるが，実際のシステム稼働時には最悪実行時間を費やすことは稀であるため，プロセッサ使用率は想定したものよりも低くなる．本研究では，タスクの実行時間として WCET ではなく，予測された時間を仮定し，予測時間をスケジューリングに反映させる方法を試みる．

実行時間の予測方法については，様々な選択肢がある．
(1) ランダムな実行時間を仮定する
(2) 事前に実行時間を計測する
(3) 実行履歴から推測する
上記（1）は，実際の実行時間よりも短い時間を仮定する可能性が高く，（後述する）予測実行時間に依存してデッドラインを決定するアルゴリズムではデッドラインミスを多発させる可能性がある．（2）は，システム稼働中に同一タスクが何度も実行されることを考慮すると，実行時間の変動に追随できない欠点がある．（3）は，同一タスクが過去に実行された履歴から実行時間を予想するものであり，実行時間の変動に追随できる可能性を持つ．本稿では（3）に相当する以下の方法を採る．

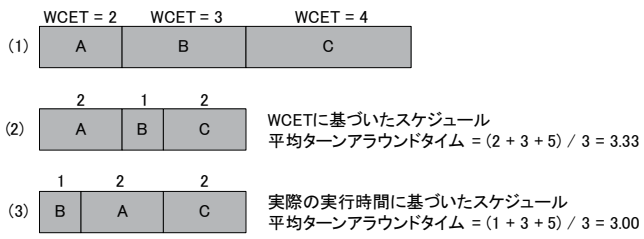

図 1 Shortest Job First（SJF）スケジューリング
Fig. 1 Shortest Job First (SJF) scheduling.

$$C_{i_{PET_k}} = \alpha \times C_{i_{PET_{k-1}}} + (1-\alpha) \times C_{i_{ET_{k-1}}}, \qquad (1)$$
$$C_{i_{PET_0}} = C_{i_{WCET}}$$

ここで，$C_{i_{PET_k}}$ は周期タスク τ_i の k 回目 $(k = 0, 1, \ldots)$ の実行のための予測実行時間を意味する．$C_{i_{ET_{k-1}}}$ 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 $C_{i_{PET_0}}$ 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，前回までの $C_{i_{PET_{k-1}}}$ と直前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している．

3. 予測実行時間を利用するスケジューリング

本節では，予測実行時間を利用する手法として，適応型 Earliest Deadline First（Adaptive EDF）と適応型 Total Bandwidth Server（Adaptive TBS）を提案する．

3.1 適応型 Earliest Deadline First

RM では重要度の高いタスクの周期を短く設定することにより，重要なタスクの応答時間を短く保つことが可能であるが，アプリケーションからみた重要度を周期と独立に設定することは不可能である．一方 EDF では，デッドライン時刻にしたがって優先度が決定されるため，周期の短いタスクが高い優先度を得られる傾向があるものの，やはり周期と独立した重要度を導入することはできない．提案する適応型 EDF では，周期と独立した重要度を導入する．

適応型 EDF では，重要度の高い周期タスクの実行インスタンスは 2 つに分解される．すなわち，周期タスク τ_i の重要度が高いと仮定すると，その k 番目（$k = 0, 1, ...$）のインスタンス J_{i_k} の実行を 2 つの部分（$J_{i_{PET_k}}$ と $J_{i_{REST_k}}$）に分割する．$J_{i_{PET_k}}$ は J_{i_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．$J_{i_{REST_k}}$ は予測された終了時刻から後の実行に相当する．τ_i の最悪実行時間を $C_{i_{WCET}}$，$J_{i_{PET_k}}$ の予測実行時間を $C_{i_{PET_k}}$，$J_{i_{REST_k}}$ の実行時間を $C_{i_{REST_k}} = C_{i_{WCET}} - C_{i_{PET_k}}$ とする．

$J_{i_{PET_k}}$ と $J_{i_{REST_k}}$ には，以下のように絶対デッドラインが設定される．（式の中で，U_i は τ_i による（WCET に基づいた）プロセッサ使用率である．）

$$d_{i_{PET_k}} = k \times T_i + \frac{C_{i_{PET_k}}}{U_i} \qquad (2)$$

$$d_{i_{REST_k}} = d_{PET_k} + \frac{C_{i_{REST_k}}}{U_i} = (k+1) \times T_i \qquad (3)$$

デッドラインが与えられると，2 つのインスタンスは EDF アルゴリズムにしたがってスケジュールされる．式 (2) と式 (3) の関係から，$J_{i_{PET_k}}$ は $J_{i_{REST_k}}$ に先行して実行される．$J_{i_{PET_k}}$ が予測終了時刻かその前に終了した場合は，$J_{i_{REST_k}}$ は存在しないことになる．

図 2 において，2 つのタスク $\tau 1$，$\tau 2$ はそれぞれ周期が 4，6，WCET が 2，2 であり，プロセッサ使用率は τ_1 が $U_1 = 2/4 = 0.5$，τ_2 が $U_2 = 2/6 = 0.33$ である．$\tau 1$ の実際の実行時間は 2，$\tau 2$ の実際の実行時間が 1 である．（この例では実際の実行時間は変動しないものとする．）また，$\tau 2$ が重要度が高いタスクであると仮定し，適応型 EDF の制御対象とする．PET の計算における α は 0.5 とする．

EDF では $\tau 2$ の 3 回の実行において平均応答時間は (3 + 1 + 3) / 3 = 2.33 となる．適応型 EDF では $\tau 2$ の 1 回目の実行は $C_{2_{PET_0}} = C_{2_{WCET}} = 2$ であり，$d_{2_{PET_0}} = T_2 = 6$ となる．2 回目の実行は $C_{2_{PET_1}} = 0.5 \times 2 + 0.5 \times 1 = 1.5$ となり，$d_{2_{PET_1}} = 1 \times T_2 + 1.5/U_2 = 10.5$ となる．3 回目の実行は $C_{2_{PET_2}} = 0.5 \times 1.5 + 0.5 \times 1 = 1.25$ となり，$d_{2_{PET_2}} = 2 \times T_2 + 1.25/U_2 = 15.75$ となる．したがって，EDF では 3 回目の実行は 14 ティック目で開始し，15 ティック目で終了するが，適応型 EDF ではデッドラインが $\tau 1$ の 4 回目の実行のデッドラインよりも早くなり，12 ティック目で開始し，13 ティック目で終了する．平均応答時間は (3 + 1 + 1) / 3 = 1.67 であり，EDF よりも短くなる．

3.2 適応型 Total Bandwidth Server

複雑な組込みアプリケーションでは，周期タスクに加えて非周期タスクが存在することがある．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（ハードタスク）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（ソフトタスク）の混在が挙げられる．このような混在タスクセットを対象とするスケジューリングアルゴリズムの一つに Total Bandwidth Server（TBS）がある [5]．TBS は EDF スケジューリング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り，かつ非周期タスクに対して短い応答時間を提供するという特長がある．

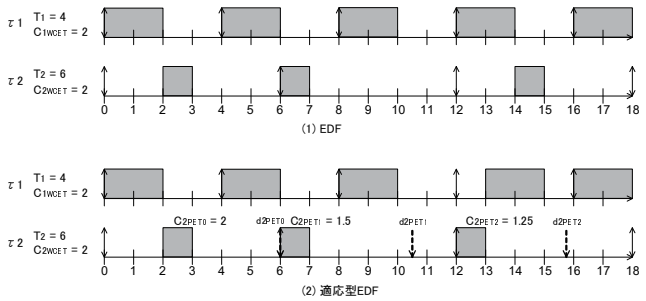

図 2 EDF vs. 適応型 EDF の例
Fig. 2 Example of the original vs. adaptive EDF.

TBS では，非周期タスクはあらかじめデッドラインは持たないが，起動されたときに以下の式から求まる絶対デッドライン時刻 d_k が与えられる．

$$d_k = max(r_k, d_{k-1}) + \frac{C_k}{U_s} \tag{4}$$

ここで，k は k 番目の非周期タスク実行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）のインスタンスの絶対デッドライン時刻，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プロセッサ使用率である．サーバは U_s の使用率を持ち，非周期タスクの発生毎に，そのインスタンスの実行に U_s のバンド幅を与える．$max(r_k, d_{k-1})$ 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絶対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 によってスケジュールされる．U_p を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 [5]．

TBS に対しては，以下のような考察が可能である．非周期タスクはあらかじめ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を保証する必要はない．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．仮定した実行時間に基づいてデッドラインを設定し，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以内で終了した場合は，その短いデッドラインと EDF アルゴリズムの性質により応答時間の短縮が期待できる．以上が提案する適応型 TBS の概要である．

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．それらを別々の非周期タスクの実行と考えれば，TBS と同じものと見なせる．k 番目のインスタンス J_k の実行を 2 つの部分（J_{PET_k} と J_{REST_k}）に分割する．J_{PET_k} 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_{REST_k} 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_{REST_k} は存在しないことになる．J_k の最悪実行時間を C_{WCET_k}，J_{PET_k} の予測実行時間を C_{PET_k}，J_{REST_k} の実行時間を $C_{REST_k} = C_{WCET_k} - C_{PET_k}$ 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，2 つのインスタンスには以下の絶対デッドライン時刻が設定される．

$$d_{PET_k} = max(r_k, d_{k-1}) + \frac{C_{PET_k}}{U_s} \quad (5)$$

$$d_{REST_k} = d_{PET_k} + \frac{C_{REST_k}}{U_s} \quad (6)$$

一方，TBS のデッドライン設定は以下のものである．

$$d_k = max(r_k, d_{k-1}) + \frac{C_{WCET_k}}{U_s} \quad (7)$$

$C_{REST_k} = C_{WCET_k} - C_{PET_k}$ と式（5），（6），（7）から，

$$\begin{aligned} d_{REST_k} &= max(r_k, d_{k-1}) + \frac{C_{PET_k}}{U_s} + \frac{C_{WCET_k} - C_{PET_k}}{U_s} \\ &= max(r_k, d_{k-1}) + \frac{C_{WCET_k}}{U_s} \quad = \quad d_k \end{aligned}$$

したがって，到着した際に式（5）と（7）により，二つの締切時刻が計算できる．式（6）ではなく（7）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，タスク毎に事前に一度計算するのみで良い．

図 3 において，上段（1）が TBS，下段（2）が適応型 TBS のスケジューリングを示している．図中には 2 つの周期タスク，$\tau 1$ と $\tau 2$，および 1 つの非周期タスクのリクエストがある．$\tau 1$ は周期 $T_1 = 4$, 実行時間 $C_1 = 1$ であり，$\tau 2$ は周期 $T_2 = 6$，実行時間 $C_2 = 3$ であり，周期タスクによるプロセッサ使用率は $U_p = 0.25 + 0.5 = 0.75$，非周期サーバの使用率は $U_s = 1 - U_p = 0.25$ である．

非周期リクエストはティック 3 で発生し，その最悪実行時間は 3 と想定されているが，予測実行時間および実際の実行時間は 2 となる予定である．（1）では，TBS によって非周期タスクのデッドラインは $d_{WCET} = 3 + 3/0.25 = 15$ となる．EDF アルゴリズムにしたがって，非周期タスクはティック 5 で実行を開始し，ティック 6 で $\tau 2$ のために中断している．そして $\tau 1$ の実行を挟んで，ティック 10 で実行を再開し，ティック 11 で実行を終了する．結果的に応答時間は $11 - 3 = 8$ となる．

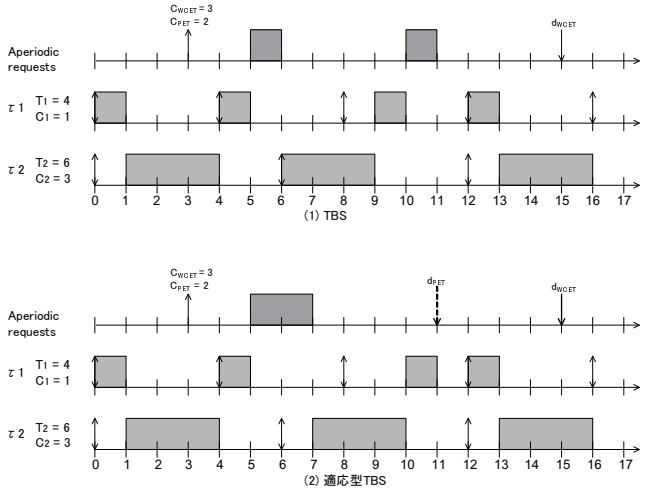

図 3 TBS vs. 適応型 TBS の例
Fig. 3 Example of the original vs. adaptive TBS.

一方，適応型 TBS において，絶対デッドラインは $d_{PET} = 3 + 2/0.25 = 11$ と $d_{REST} = 11 + (3 - 2)/0.25 = 15 = d_{WCET}$ が設定される．EDF アルゴリズムにしたがって，ティック 5 で開始した非周期タスクは，ティック 7 で実行を終了し，その応答時間は $7 - 3 = 4$ となる．この例では，TBS に対して，適応型 TBS は応答時間を 4 ティック分削減したことになる．

3.3 スケジューラビリティ

適応型 EDF のスケジューラビリティは以下のように論じることが可能である．重要度の高い対象周期タスク τ_i による，最悪実行時間に基づいたプロセッサ使用率を U_i とすると，適応型 EDF は，U_i を使用率とする Total Bandwidth Server（TBS）によって τ_i から分割された 2 つのインスタンスが実行される状況と同じものと見なすことができる．すなわち，2 つのインスタンスの実行時間は $C_{i_{PET_k}}$ と $C_{i_{REST_k}}$ であり，TBS により与えられるそれぞれのデッドラインは式（2）と式（3）に等しい．したがって，文献 [5] における TBS のスケジューラビリティの性質が保たれるため，全タスクによるトータルプロセッサ使用率 U に対して $U < 1$ であることがタスクセットがスケジューラブルとなる必要十分条件となる．

適応型 TBS においても同様である．タスク実行を 2 つに分解した後，明らかに適応型 TBS は TBS と同じ振る舞いとなる．適応型 TBS では分解した 2 つのタスクが同時に到着したと考えればよい．$max(r_k, d_{k-1})$ と d_k の間の時間では，式（5），（6）から，2 つの非周期タスクによるプロセッサ使用率は以下のように TBS におけるものと同じ値となる．

$$\begin{aligned} U_{J_{PET_k}} &= \frac{C_{PET_k}}{d_{PET_k} - max(r_k, d_{k-1})} = U_s \\ U_{J_{REST_k}} &= \frac{C_{REST_k}}{d_{REST_k} - d_{PET_k}} \qquad = U_s \end{aligned}$$

したがって，適応型 TBS のスケジューラビリティは TBS の場合と同じ証明により導かれる．

3.4 実装の複雑さ

2 つの提案アルゴリズムではタスク実行を 2 分割するが，OS 管理においてはタスク情報は一つのタスク構造体として存在するべきである．PET 時間の経過時にタスクの実行が終了していない場合は，デッドラインを再設定し，レディキューに挿入し直すことで，タスク毎に一つのタスク構造体のみ用意することで実現可能である．したがって EDF や TBS との違いは，PET 経過の出検とデッドラインの再設定のみである．PET 経過の検出を可能とするために，ティック毎にスケジューラの実行が必要となるが，これは通常のタイマー／ティック割込みハンドラがスケジューラを呼び出すことで可能である．また，3.2 節で述

べたように，適応型 TBS において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（7）による値を静的に計算しておき，必要な時に使用することが可能である．適応型 EDF の場合のデッドラインの再計算は式（3）からわかるように，実際はシフトおよび加算で行うことが可能である．

4. 評価

4.1 評価方法

評価のために，周期タスクと非周期タスクから構成されるタスクセットを作成した．各周期タスクの周期は 1 ティックから 100 ティックの間の一様乱数によって決定し，WCET は周期の長さの 1/10 から 1/3 の間の一様乱数によって決定した．実際の実行時間は WCET の 1/3 から 1/1 の間の一様乱数によって決定した．（同一タスクの異なる実行インスタンスは異なる実行時間を持つ．）複数の周期タスクを混ぜて，周期タスクによるプロセッサ使用率 U_p が 70%から 95%の間の 5% おきとなるように，周期タスクセットを用意した．

一方，非周期タスクについて，リクエストの到着間隔は 1000 ティックあたり平均到着数 1.25 のポワソン分布，WCET は平均 8 ティックの指数分布，および実際の実行時間は WCET を上限とする平均 4 ティックの指数分布に従った．全ての非周期タスクインスタンスについて，WCET に対する実際の実行時間の割合の平均は 0.33 であり，観測時間における非周期タスクによるプロセッサ使用率は 0.5%から 2%程度であった．

以下の 6 つの方式を評価対象とした．

(1) RM+BGS
周期タスクを RM，非周期タスクをバックグラウンドサーバ（BGS）でスケジューリングする方式である．BGS は，実行可能な周期タスクが存在しない場合にのみ，非周期タスクを到着准に実行するものである．

(2) EDF+BGS
周期タスクを EDF, 非周期タスクを BGS でスケジューリングする方式である．

(3) AEDF+BGS
周期タスクに対しては，最重要タスク 1 つを対象とする適応型 EDF（AEDF）によってスケジューリングし，非周期タスクを BGS でスケジューリングする方式である．

(4) AEDF+TBS
周期タスクを AEDF，非周期タスクを TBS でスケジューリングする方式である．

(5) AEDF+ATBS
周期タスクを AEDF，非周期タスクを適応型 TBS でスケジューリングする方式である．

(6) Oracle EDF+Oracle TBS
周期タスクのうち最重要タスク 1 つを各インスタンスの実行時間が既知である場合の AEDF（Oracle EDF）の制御対象とし，非周期タスクを各インスタンスの実行時間が既知である場合の ATBS（Oracle TBS）でスケジューリングする方式である．

各 U_p に対して，周期タスクセットを 10 種類，非周期タスクセットを 10 種類用意し，それらを組み合わせて 100 通りのシミュレーションを行い，その平均値を当該 U_p における結果とする．本評価では周期の最も長いタスクを最重要タスクとみなした．適応型 EDF および適応型 TBS において，PET を求める式（1）内の α として 0.5 を使用した．なお，観測時間は 100,000 ティックである．

4.2 結果

図 4 に最重要周期タスクの平均応答時間を示す．横軸は U_p，縦軸は平均応答時間（ティック数）である．周期の最も長いタスクを最重要タスクとしたため，RM によるスケジューリングが最も応答時間が長くなった．AEDF を使用する 3 つの方式はほぼ同じ結果となり（図では完全に重なっている），これらは EDF よりも短い応答時間となり，U_p が 95%のときに最大 13.4%の平均応答時間の短縮となった．実行時間が既知の AEDF（Oracle EDF）が最も短い応答時間となった．これは実行時間を完全に予測した場合の適応型 EDF であり，適応型 EDF の限界であると見なせる．

図 5 に非周期タスクの平均応答時間を示す．全体的にみると U_p が大きいほど応答時間が長くなることが確認できる．BGS を使用する 3 つの方式は完全に一致しており，TBS を使用する方式が応答時間を大幅に短縮していることがわかる．ATBS を使用する方式は，さらに応答時間を短縮しており，U_p が 85%のときに TBS に対して最大 32.0%の平均応答時間の短縮となった．なお，実行時間が既知の ATBS（Oracle TBS）が最も短い応答時間となった．これは実行時間を完全に予測した場合の適応型 TBS であり，適応型 TBS の限界であると見なせる．

5. おわりに

本稿ではタスクの実行時間が変動することを考慮し，予測実行時間を導入してデッドライン計算に利用することにより，応答時間を短縮する 2 つの方式を提案した．一つ目は適応型 EDF であり，重要度の高いタスクの実行を 2 つに分割し，その一つ目の実行が予測実行時間内に完了した場合は短いデッドラインの効果により応答時間の短縮が期待できる．もう一つは適応型 TBS であり，非周期リクエストの実行を 2 つに分割し，適応型 EDF と同様の応答時間短縮を狙うものである．

シミュレーションによる評価では，EDF に対して，適応型 EDF により重要度の高い周期タスクの平均応答時間が

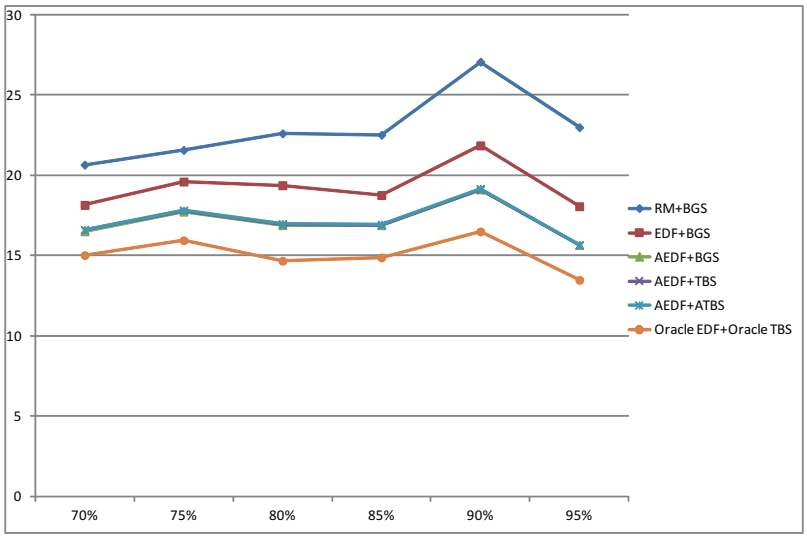

図 4 最重要周期タスクの平均応答時間

Fig. 4 Average response time of the most important periodic task.

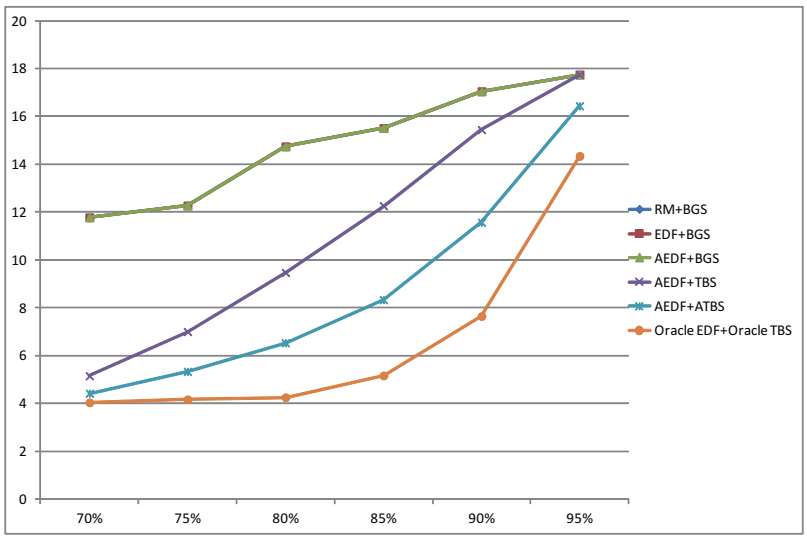

図 5 非周期タスクの平均応答時間

Fig. 5 Average response time of aperiodic tasks.

最大 13.4%，TBS に対して，適応型 TBS により非周期リクエストの平均応答時間が最大 32.0%短縮されることが確認された．

今回の評価では，PET の取得は直前の実際の実行時間と前回までの PET との単純な加重平均であった．仮に PET が完全に予測できたとすると，Oracle EDF や Oracle TBS が示すように，更なる平均応答時間の改善が見込まれる．したがってより良い PET の取得方法を議論する必要がある．また，今回の評価は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には，実プログラムを使用した評価が望まれる．今後は実プログラムを使用し，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．

参考文献

[1] 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).

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

[3] 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).

[4] Wilhelm, R., et al.: *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).

[5] 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).