周期タスクのジッタを軽減するリアルタイムスケジューリング

田中 清史[1,a)]

概要：周期タスクとして実現される制御アプリケーションにおいて，実行タイミングあるいは実行時間が変動することにより周期性が乱れ，システム性能の低下や不安定性を引き起こす．本研究では周期タスクのジッタと平均応答時間を抑制するリアルタイムスケジューリング方式を提案する．提案する方式では，タスクの実行時間の変動に対して適応的にデッドラインを設定する技法と，リリース時刻を仮想的に遡及させることにより短いデッドラインを与える方法を使用する．評価において，前者の技法は対象タスクの平均応答時間を最大 20.5%削減し，後者はジッタを最大 35.4% 軽減することが示された．

キーワード：リアルタイムスケジューリング，ジッタ，応答時間，周期タスク

Real-Time Scheduling for Reducing Jitters of Periodic Tasks

Kiyofumi Tanaka[1,a)]

Abstract: For control applications implemented with periodic tasks, fluctuation in start timing or execution times may disturb the periodicity and cause performance degradation or instability. This study proposes real-time scheduling techniques that reduce jitters and average response times of periodic tasks. In the proposed scheduling, two techniques are employed: one adaptively extending deadlines according to varying execution times, and the other further obtaining short deadlines by virtually advancing release times. The evaluation shows that the former shortens average response times of the target tasks by up to 20.5% and the latter mitigates jitters of them by up to 35.4%.

Keywords: Real-time scheduling, jitter, response time, periodic tasks

1. はじめに

電子計算機の利用を前提とした制御理論において，多くの場合，サンプリングやアクチュエーションは時間同期で（周期的に）行われることが仮定される [1]．特に閉ループ制御（closed-loop control）系システム等では，厳密なタイミングでのタスク実行が要求されるため，応答時間のずれ，すなわちジッタの発生により周期性が乱れるとシステム性能の低下や不安定性を引き起こす可能性がある [2], [3]．単一対象の制御アプリケーションではジッタは小さいことが期待できるが，システム開発において，各制御構造の設計とシステムの実装を分離して行った場合，マルチタスク化やクリティカリティ混在（Mixed-Criticality [4], [5]）化による影響から，前者の設計工程では予測不可能であった遅延やジッタが発生し，周期的な振る舞いから逸脱する可能性がある [6]．

ジッタの問題の他，アプリケーションによってはタスクの応答時間が重要になる．例えば資源を多数使用するタスクの応答時間が長くなる場合，資源競合が頻繁に発生し，他のタスクが実行の中断を余儀なくされ，システムの性能を低下させる．このような資源競合を緩和するために，資源利用に着目して特定のタスクの応答時間を短縮する技術が望まれる．

ハードリアルタイムシステムでは，ジッタおよび応答時間の問題の他，各タスク実行がデッドライン以内に完了することが最重要である．周期タスクを対象とし，デッドライン制約が満たされること（スケジューラビリティ）を保証することが可能な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 [7]．RM は固定優先度アルゴリズムの一つであり，周期の短いタスクが高い優先度を持つとみなされ，優先的に実行される．RM では優先度が高い（周期の短い）タスクのジッタや応答時間が小さいという特長

[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

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時間を 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，絶対デッドライン時刻の近いタスクを優先的に実行する．100%のプロセッサ使用率の下でスケジューラビリティが保証されるという特長があるが，タスクに静的な優先度を与えることができないため，特定の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 100%のプロセッサ使用率を達成する EDF に基づき，スケジューラビリティを確保しつつ，システムにとって（周期の長さとは無関係に）重要な周期タスクの平均応答時間およびジッタを抑制することを目的とする．提案する手法では，実際のシステムにおいてタスクの実行時間が変動することに着目し，非周期タスクの応答時間を短縮することを目的として著者が過去に提案した適応型 TBS [8], [9] を周期タスクに対して適用し，かつ，新たにリリース時刻を仮想的に前進させる方法を導入し，応答時間とジッタの短縮を実現する．

本稿は以下の構成をとる．2 節で周期タスクのジッタ軽減に関する関連研究を述べ，3 節で適応型 TBS の周期タスクへの適用方法および仮想リリース遡及法を提案する．続いて 4 節において提案手法の評価と考察を行い，5 節でまとめる．

2. 関連研究

2.1 周期タスクのジッタの軽減

周期タスクのジッタを軽減する方法に関して様々な研究が存在し，それらはその方策の違いから 3 種類に分類される [10]．第一の方策はタスクコードを複数のフェーズに分割し，それぞれをサブタスクとして実行することで小さいジッタを得るものである．第二は，対象タスクのデッドラインを短くすることで応答時間を短縮し，ジッタを小さく抑える方法である．第三はプリエンプションを禁止することにより実行時間を一定に保つことでジッタを小さくするものである．

第一の方策に属する研究として，Balbastre らはタスク実行を入力（Data acquisition）フェーズ，計算（Computation of the control action）フェーズ，および出力（Output the control action）フェーズに分割するタスクモデルを提案した [11]．入力フェーズを担当するサブタスクと出力フェーズに相当するサブタスクにそれぞれ短いデッドラインを与えて優先度を上げることにより，ジッタを小さく抑えることが可能となる．しかしこの方法を実現するためは，アプリケーションタスクの設計時に 3 つのタスクコードへ分割することが強いられる．また，各サブタスクに対して本来の（周期と等しい）デッドラインよりも短いデッドラインを設定するため，スケジューラビリティ解析において，プロセッサ総使用率に基づく容易な方法ではなく，プロセッサ要求（Processor demand）／応答時間に基づく，計算量が大きく複雑な解析が必要となる [12]．

同様にタスクを 3 つに分割する手法として，Buttazzo らは入力，出力フェーズを低レベルハンドラとして実装し，周期タイミングと同時に優先的に実行することによりジッタを小さくする方法を提案した [10]．具体的には，各周期タイミングで前回のジョブの出力処理と次ジョブの入力処理を行うことにより，ジッタをほぼ零とすることが可能となる．この方法は前述の方法と同様にタスクの分割設計が必要であり，かつサブタスクを OS ハンドラとして実装することが必要である．また，ハンドラ部分は固定優先度ルーチンとなるため，純粋な EDF が実現不可能であり，スケジューラビリティを確保するためにプロセッサ使用率の上限を低下させることになる．加えて，応答時間に関するジッタは最小化されるが，各ジョブの入力フェーズと出力フェーズの時間間隔が基本的に周期の長さとなるため，周期の長いタスクでは入力（サンプリング）に対して時宜を得た出力（アクチュエーション）とならない可能性がある．さらに，複数タスクで周期タイミングが重なったときに，ハンドラ実行が複数行われ，オーバヘッドが大きくなる．以上の問題点から，本研究ではタスクを静的に分割する方法は対象外とする．

第二のデッドラインを短縮する方法として，Baruah らは EDF スケジューリングを基本とし，タスクセット全体の最大ジッタを最小に保つためのデッドライン計算法を提案した [13]．この方法では，本来は周期と等しいデッドラインを持つ各タスク（τ_i）に対し，タスクのプロセッサ使用率 C_i/T_i（C_i，T_i はそれぞれ τ_i の最悪実行時間と周期）よりも大きなバンド幅 θ_i（$\theta_i \geq C_i/T_i$，ただし総プロセッサ使用率が 100%を超えない範囲）を与えることにより，より短い相対デッドライン，すなわち $D_i = C_i/\theta_i$ を設定することでジッタを短く抑える．この方法は，（文献 [13] では明記されていないが）対象タスクに Total Bandwidth Server (TBS) [14] を適用する方法と見なすことができる．しかしこの方法は，タスクセットのプロセッサ使用率が 100%に近く，余剰バンド幅が小さい場合は十分大きな θ_i を確保できず，効果が得られない．また，タスクの実行時間が一定である（すなわち，同一タスクの全ジョブは実行時間が等しい）ことを仮定しているため，各タスクに設定される相対デッドラインは不変であり，実行時間が変動するアプリケーションではジョブ毎に最適なデッドラインを得ているとはいえない．さらに，周期とは異なるデッドラインを使用するため，スケジューラビリティ解析はプロセッサ要求に基づく方法を採る必要がある．この研究はタスクセット全体のジッタを抑えることを目的としており，特定タス

[*1] 任意のタスクセットに対してスケジューラビリティを保証するための十分条件はプロセッサ使用率が 69%以下となることである [12].

クのみのジッタを軽減する本研究とは目的が異なるが，基本方式として TBS を利用する点は同様である．

第三の方策であるプリエンプション禁止方法は，ジョブ開始から終了まで中断が無いためジッタを小さく保つことが可能であるが，スケジューラビリティの保証が不可能であることが欠点である [10]．本研究はスケジューラビリティの確保を最重要視するため，プリエンプションの禁止は対象としない．

本研究では，上記の問題点の解決を図る．すなわち，第一の方策のような特別な（フェーズ分割を強いる）タスク設計を必要とせず，EDF におけるプロセッサ使用率の上限（100%）を保ち，プロセッサ使用率に基づく容易なスケジューラビリティ解析を可能とし，かつ実行時間が変動するタスクに対してジョブ毎に適切なデッドラインを設定することを可能とする方法を提案する．

2.2 Total Bandwidth Server の適用

Total Bandwidth Server (TBS) [14] 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（あるいはソフトデッドラインを持つ）非周期タスクの混在セットにおいて，非周期タスク実行を担当するサーバアルゴリズムであり，実装コスト／オーバヘッドが小さく，周期タスクのスケジューラビリティを保障し，非周期タスク実行の応答時間を短く保つ特徴がある．

TBS では，時刻 r_k で到着する（リリースされる）k 番目の非周期ジョブに対し，以下のような仮の絶対デッドライン d_k が割り当てられる．

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

d_{k-1} は前（$k-1$ 番目）ジョブの絶対デッドライン，C_k^{WCET} は当該ジョブの最悪実行時間，U_s はサーバに割り当てられたバンド幅（プロセッサ使用率）である．max の項により，連続するジョブ同士でバンド幅領域がオーバーラップしないように調整している．

TBS はジョブの最悪実行時間を使用してデッドラインを決定しているため，ジョブが早期終了した場合は結果的に過度な長さのデッドラインを割り当てていたことになる．この過大なデッドラインが式 1 の max 項によって後続リクエストに影響することを回避するために，文献 [15] においてリソース回収法が提案された．

リソース回収法では，k 番目の非周期ジョブに以下のデッドライン d'_k が与えられる．

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

$\overline{r}_k$ は以下の式により与えられる．

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

すなわち，当該ジョブのリリース時刻（r_k），前（$k-1$ 番目）ジョブの再計算されたデッドライン（$\overline{d}_{k-1}$，後述），および前ジョブの終了時刻（f_{k-1}）のうち，最大のものでリリース時刻を置き換える．$k-1$ 番目の非周期ジョブが終了したとき，実際に費やした実行時間 $\overline{C}_{k-1}$ を使用して，次の式によってデッドラインを再計算する．

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

この $\overline{d}_{k-1}$ を式 3 に反映し，続いて式 2 によって後続ジョブのデッドラインが求められる．この方法は早期終了で得られる余剰（スラック）時間を後続ジョブのデッドライン計算に反映させるものである．（一方，3.2 節で提案する方法はジョブの早期終了を先取り（予測）して自身のデッドラインに反映させる方法である．）なお，本稿で提案する仮想リリース遡及法は TBS のリソース回収法の使用を前提としている．

周期タスクは非周期タスクの部分集合（特別な場合）であるため，TBS を周期タスクに適用することが可能である．2.1 節で述べたように，文献 [13] では TBS の適用により周期よりも短いデッドラインを提供し，ジッタを短縮する方法を採っている．

TBS を，周期 T_i，最悪実行時間 C_i^{WCET} を持つ周期タスク τ_i に適用した場合，k 番目のジョブは以下の絶対デッドラインが割当てられる．

$$d_{i,k} = \phi_i + k \times T_i + \frac{C_i^{WCET}}{\theta_i} \qquad (k \geq 0) \tag{5}$$

ϕ_i は τ_i のフェーズ，すなわち最初のジョブのリリース時刻である．θ_i はタスクに割当てられるバンド幅であり，これが τ_i による使用率（$U_i = C_i^{WCET}/T_i$）と等しい場合，$d_{i,k} = \phi_i + (k+1) \times T_i$，すなわち絶対デッドラインが周期のタイミングと一致する．タスクセットのプロセッサ使用率 U が 100%未満である場合，余剰バンド幅を使用して $\theta_i > U_i$ とすることにより，τ_i は周期タイミングよりも早い絶対デッドラインが割り当てられ，EDF によって早期にスケジュールされる可能性が高まり，応答時間およびジッタの短縮が期待できる．なお，この手法を適用する場合，n 個のタスクからなるタスクセットがスケジューラブルとなる必要条件は $\sum_{i=1}^{n} \theta_i \leq 1$ である．（十分条件はプロセッサ要求に基づく解析によって与えられる [12].）

3. 提案手法

3.1 対象タスクセットとジッタ

本研究で対象とする周期タスクセットにおいて各タスク τ_i $(i = 1, 2, ...)$ は周期 T_i，最悪実行時間 C_i^{WCET} を持ち，相対デッドラインは周期と等しい（$D_i = T_i$）ものとする．このタスクモデルでは絶対デッドラインは周期タイミングとなる．

周期タスクのジッタとして，相対ジッタと絶対ジッタを

対象とする．タスク τ_i の相対ジッタ RJ_i および絶対ジッタ AJ_i はそれぞれの以下の定義に従う [12]．

$$RJ_i = \max_k |(f_{i,k} - r_{i,k}) - (f_{i,k-1} - r_{i,k-1})| \quad (6)$$

$$AJ_i = \max_k (f_{i,k} - r_{i,k}) - \min_k (f_{i,k} - r_{i,k}) \quad (7)$$

定義において，$r_{i,k}$，$f_{i,k}$ はそれぞれ τ_i の k 番目のジョブのリリース時刻，終了時刻である．相対ジッタは同一タスクの連続するジョブの応答時間の差であり，絶対ジッタは同一タスクの全ジョブの最長応答時間と最短応答時間の差である．

3.2 適応型 TBS の適用

本節において，著者が過去に提案した適応型 TBS [8], [9] を周期タスクに適用する方法を提案する．

応答時間およびジッタを抑制する対象となるタスクを τ_i とし，このタスクを適応型 TBS によって実行する．適応型 TBS が使用可能なバンド幅を θ_i とする．θ_i は 2.2 節で述べたものと同様であり，例えば対象タスクが一つのみである場合は，$\theta_i = 1 - (U_p - U_i)$ となる．（U_i は τ_i のプロセッサ使用率（C_i^{WCET}/T_i），U_p は全周期タスクのプロセッサ使用率の合計である．）このバンド幅を使用し，τ_i の k 番目のジョブは以下のように段階的なデッドラインが割り当てられる．

$$d_{i,k}^0 = \phi_i + k \times T_i + \frac{C_i^0}{\theta_i} \quad (8)$$

$$d_{i,k}^j = d_{i,k}^{j-1} + \frac{C_i^j}{\theta_i} \qquad (j > 0) \quad (9)$$

C_i^j は予測実行時間である．τ_i のジョブが実行されるときは，最初に時間 C_i^0 で実行が終了するものと予測する．C_i^0 で終了しない場合は，残りの実行が時間 C_i^1 で終了すると仮定する．これをジョブが最終的に終了するまで繰り返す．この方法は早期終了を予測することでジョブ自身のデッドラインを早める効果がある．予測実行時間は適応型 TBS の定義とは独立して設定可能である．（本稿における評価では，全ての j に対して $C_i^j = 1$（ティック）とした．）スケジューラビリティおよび実装の複雑さについては文献 [9] を参照されたい．

3.3 仮想リリース遡及法

本節において，適応型 TBS の適用に加えて更にデッドラインを早める方法として仮想リリース遡及法（Virtual Release Advancing）を提案する．TBS では，基本的にジョブのリリース時刻（到着時刻）を起点として絶対デッドライン時刻が計算される．これは，仮にリリース時刻が早まれば，より早いデッドライン時刻を得ることを意味する．周期タスクのジョブのリリース時刻は周期タイミングで固定されるため，実際にリリース時刻が早まることは無いが，仮想リリース遡及法では，仮想的にリリースがより早い時刻になされたものと仮定し，早いデッドラインを算出することを試みる．

仮想リリース遡及法の基本方針は，過去のスケジュールに影響を及ぼさない範囲でリリース時刻を遡らせることである．すなわち，たとえリリース時刻 $r_{i,k}$ を仮想的に過去の時刻 $vr_{i,k}$ に移動させ，それに従いデッドラインを前倒しにしたと仮定しても，$vr_{i,k}$ と $r_{i,k}$ の間の（過去の）時間帯に，より早いデッドラインを持つ他のタスクが実行されていた場合は，過去のスケジュール（タスクの実行順序）は変わらない．このような場合にリリース時刻を遡らせることが可能となる．

本方式は TBS を前提としているため，式 1 の max 項が示す通り，サーバの使用可能バンド幅の観点からリリース時刻を前ジョブのデッドラインを越えて前倒しすることはできない．すなわち，本研究では $D_i = T_i$ である周期タスクを仮定しているため，リリース時刻は前ジョブのデッドラインと一致し，このままではリリース時刻を前倒しすることは不可能である．そこで，2.2 節のリソース回収法，あるいは 3.2 節適応型 TBS の適用により，前ジョブのデッドライン時刻 $d_{i,k-1}$ が当該ジョブのリリース時刻 $r_{i,k}$ よりも早くなった場合，すなわち $d_{i,k-1} < r_{i,k}$（リソース回収法の場合は $\overline{d}_{i,k-1} < r_{i,k}$）の場合に，仮想リリース遡及法が適用可能となる．

3.3.1 仮想リリース遡及法の例

図 1 に仮想リリース遡及法の例を示す．図では，3 つの周期タスク τ_1，τ_2，τ_3 がそれぞれ周期として $T_1 = 10$，$T_2 = 9$，$T_3 = 6$，最悪実行時間として $C_1 = 2$，$C_2 = 2$，$C_3 = 3$，およびフェーズ（最初のジョブのリリース時刻）として $\phi_1 = 0$，$\phi_2 = 1$，$\phi_3 = 1$ を持つ．この例では τ_1 を対象タスク（重要度の高いタスク）とし，そのジッタに着目する．図中の上段がオリジナル EDF でスケジュールした場合であり，下段が τ_1 に TBS および仮想リリース遡及法を適用した場合である．なお，τ_1 の最悪実行時間は 2 であるが，最初のジョブの実際の実行時間は 1 であり，2 番目のジョブの実行時間は 2 であるとする．オリジナル EDF では，最初のジョブは応答時間が 1 となり，2 つ目のジョブの応答時間は 7 であるため，2 つのジョブ間の相対ジッタは $7 - 1 = 6$ となる．

一方，TBS と仮想リリース遡及法の例では，τ_1 の最初のジョブの実行終了時に，TBS のリソース回収により，デッドラインが図中の下向き破線矢印のように時刻 5 に再設定される．これにより，2 つ目のジョブのリリース時刻の仮想的な前倒しが可能となる．図では，上向き破線矢印のように仮想リリース時刻を時刻 7 に設定している．これに応じてデッドラインは時刻 17 となる．時刻 7 から 10 の間は τ_3 が実行されていたが，このデッドラインは 13 であるため，この仮想リリース遡及によって過去の実行順番が変化することは無い．結果的に，2 つ目のジョブはデッドライ

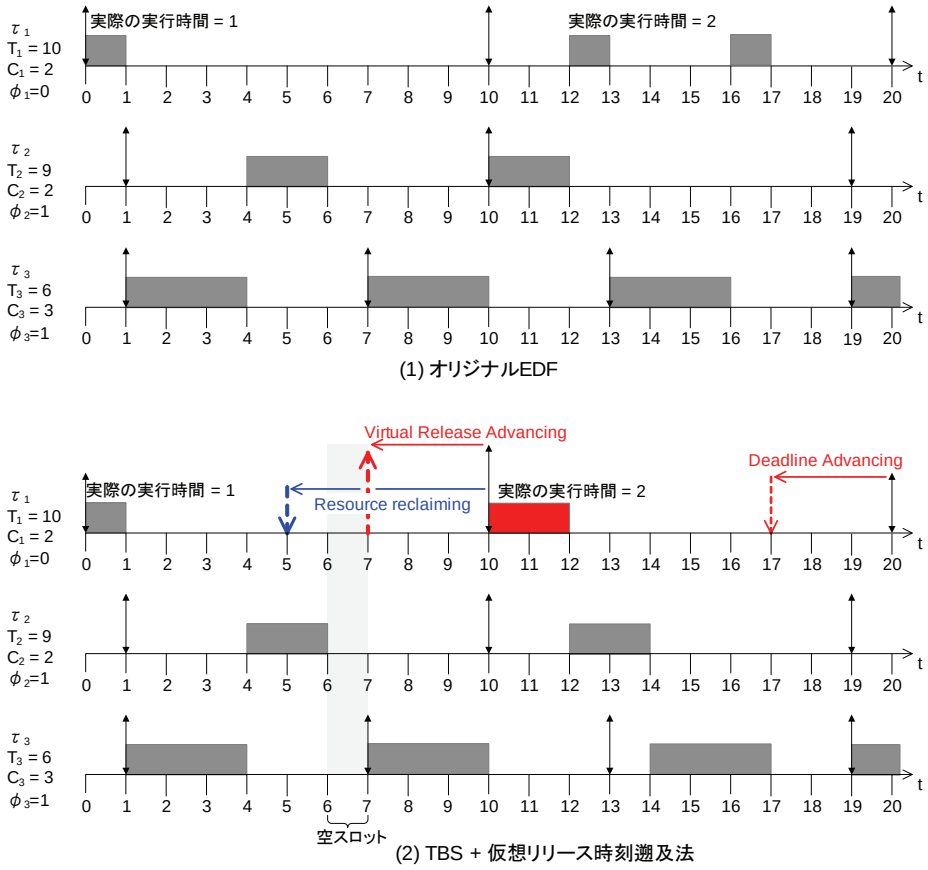

図 1 仮想リリース遡及法の例（空スロットによる遡及限界）.

ンが早くなることにより τ_2 の 2 番目のジョブよりも先に実行され，応答時間は 2 となり，相対ジッタは $2-1=1$ となる．

この例では更にリリース時刻の前倒しを行うことは不可能である．仮に，仮想リリース時刻を 6 に設定すると，時刻 6 と 7 の間は空スロットであるため，このスロットで当該ジョブが実行されていなければならないことになる．このような遡及は実際のリリース時刻（$t=10$）に対して過去のスケジュールを変化させることになるため不可能である．

3.3.2 仮想リリース遡及法の定義

仮想リリース時刻を過去のどの時点まで前倒し可能であるかを定義する．前倒しの限界点として 3 つの要因がある．それらは前ジョブのデッドライン，空スロット，および既使用最遅デッドラインである．

(1) 前ジョブのデッドライン

前ジョブのデッドラインを越えて仮想リリース時刻を前倒しすることは，TBS のスケジューラビリティの観点から不可能であるため，第一の限界要因となる．これは前述の通り，式 1 の max 項からも明らかである．この限界点検査を実現するためには前ジョブのデッドライン時刻を記憶する必要があるが，TBS の実現において既に記憶されているものを使用可能である．

(2) 空スロット

いかなるジョブも実行されなかった空スロットを越えて仮想リリース時刻を前倒しすることは，このスロットで当該ジョブが実行されなければならなかったことを意味し，過去のスケジュールに影響を及ぼす．したがって空スロットは第二の限界要因である．図 1 はこの空スロット限界要因が適用される例である．この検査を実現するために，最後の空スロット番号を記憶する必要がある．

(3) 既使用最遅デッドライン

過去のスロットのうち空スロット以外は，何れかのタスクのジョブが実行されている．各スロットはそこで実行されたジョブの絶対デッドラインが関連付けられる．これをスロットの既使用デッドラインとする．ここで，あるスロットに着目し，そのスロットを対象ジョブの仮想リリース時刻が前倒しによって越えることが可能かどうかを判断する状況を想定する．仮想リリース時刻がそのスロットを越えた場合，それに応じて前倒しされるデッドラインが当該スロットの既使用デッドラインよりも早くなる場合は，そのスロットは対象ジョブが実行されていなければならない．これは過去のスケジュールを変化させることになる．したがって，この条件は第三の限界点要因となる．より正

確には，遡及プロセスの過程で計算されるデッドラインは，仮想リリースの前倒しで越えていく全スロットの既使用デッドラインよりも早くなることは許可されない．この検査を実現するためには，過去の各スロットの既使用デッドラインと，遡及プロセスにおいて越えていくスロット群の既使用デッドラインの最大値を記憶する必要がある．

図 2 に第三の既使用最遅デッドライン要因による前倒しの限界点の例を示す．ジッタ削減の対象タスクを τ_1 とする．τ_1 のジョブの実際のリリース時刻は $t = 5$ である．時刻 5 で当該ジョブがリリースされるときに，スケジューラは仮想リリース遡及プロセスを開始する．スロット 4（時刻 4 と 5 の間のスロット）は τ_3 の 2 番目のジョブが実行されており，この既使用デッドラインは 8 である．（過去のスロットの既使用デッドラインは配列 dl[] に記憶されている．）仮想リリース時刻が 1 スロット分の前倒しでこのスロットを越えると仮定すると，デッドラインは時刻 11 から 10 に変更される．このデッドラインは 8 よりも早くないため，この前倒しは許可される．同時に既使用最遅デッドライン（max_dl）として 8 を記憶する．

続いて，更なる前倒しでリリース時刻がスロット 3 を越えると仮定すると，デッドラインは時刻 9 となる．スロット 3 は τ_2 のジョブが実行されており，その既使用デッドラインは 6 であるが，これは max_dl の値よりも小さいため，max_dl は更新されず値は 8 のままとなる．前倒しした場合のデッドラインは max_dl よりも大きいため，この前倒しは許可される．次のスロット 2 も同様に越えることが可能であり，仮想リリース時刻は $t = 2$ となる．更にスロット 1 を越えることを仮定すると，新たなデッドラインは時刻 7 となり，max_dl よりも早くなるため，この前倒しは不可能であると判断される．最終的な仮想リリース時刻を時刻 2，デッドラインを時刻 8 として遡及プロセスが終了する．

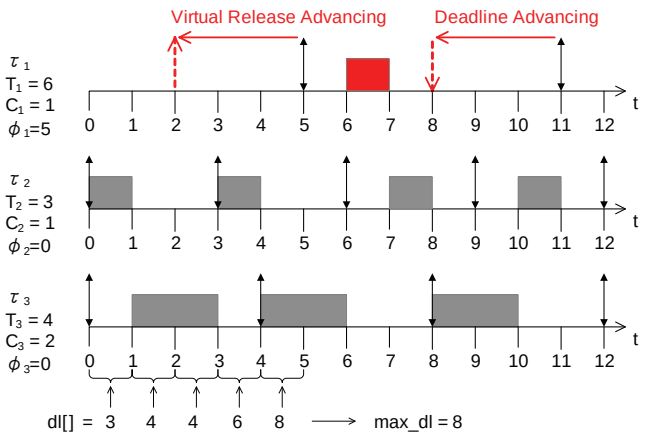

図 2 仮想リリース遡及法の例（既使用最遅デッドラインによる遡及限界）．

3.3.3 仮想リリース遡及法のアルゴリズム

Algorithm 1 に仮想リリース遡及法のアルゴリズムを示す．このアルゴリズムは一つの対象タスクに対するものであり，対象タスクのリリース時に実行される．アルゴリズム内でタスク番号は省略されており，r_k は k 番目のジョブの実際のリリース時刻，vr_k は仮想リリース時刻である．d_{k-1}，d_k はそれぞれ同一タスクの前ジョブ，当該ジョブの絶対デッドラインである．更に，C は当該タスク最悪実行時間，U_s は当該周期タスクのための TBS のサーババンド幅である．なお，適応型 TBS と組み合わせる場合は，C を 3.2 節で述べた C_i^0 に置き換える．

他の変数として，$last\_empty$ は最後の空スロットの番号を記憶するものであり，dl は各要素が対応するスロットの既使用デッドラインを記憶する配列である．更に，max_dl は遡及プロセスの過程で既使用最遅デッドラインを記憶する変数である．

アルゴリズムの 1 行目で変数 max_dl をゼロで初期化し，2 行目で vr_k を r_k の値で初期化する．続いて 3 行目から遡及プロセスのループ実行を開始する．ループ内では最初に 4 行目で現在の仮想リリース時刻を起点として（適応型）TBS のためのデッドライン計算を行う．続いて 6〜8 行目では，前ジョブのデッドライン要因による限界点検査を実行する．すなわち，仮想リリース時刻 vr_k が全ジョブのデッドライン d_{k-1} と一致した場合はループを抜けてアルゴリズムを終了する．

前ジョブのデッドライン要因検査を通過した場合は，10〜12 行目で第二の空スロット要因検査を行う．すなわち，仮想リリース時刻が空スロットの次スロット番号となった場

Algorithm 1 仮想リリース遡及法

```
1:  max_dl ⇐ 0 /* 既使用最遅デッドライン変数の初期化 */
2:  vr_k ⇐ r_k    /* 仮想リリース時刻変数の初期化 */
3:  while TRUE do
4:    d_k ⇐ vr_k + C/U_s
5:    /* 前ジョブのデッドライン要因（d_{k-1}） */
6:    if vr_k = d_{k-1} then
7:      break
8:    end if
9:    /* 空スロット要因 */
10:   if vr_k = last_empty + 1 then
11:     break
12:   end if
13:   if max_dl < dl[vr_k − 1] then
14:     max_dl ⇐ dl[vr_k − 1]
15:   end if
16:   /* 既使用最遅デッドライン要因 */
17:   if d_k ≤ max_dl then
18:     break
19:   else
20:     /* 1 スロット分の仮想リリース遡及 */
21:     vr_k ⇐ vr_k − 1
22:   end if
23: end while
```

合は，それ以上の遡及が不可能であるため，アルゴリズムを終了する．

続いて 13〜18 行目において，第三の既使用最遅デッドライン要因の検査を行う．準備として 13〜15 行目で現在の仮想リリース時刻よりも 1 つ分過去のスロットの既使用デッドラインと max_dl を比較し，必要ならば既使用最遅デッドラインを更新する．17〜18 行目において当該ジョブの現在のデッドラインが max_dl と等しいか，あるいは早くなる場合は，これ以上の前倒しを行わずにアルゴリズムを終了する．

上記の全ての要因検査を通過した場合は，21 行目で 1 スロット分の仮想リリース遡及を行い，再度ループを繰り返す．

3.3.4 仮想リリース遡及法のスケジューラビリティ

仮想リリース遡及法では当該ジョブが前倒しされた仮想リリース時刻にリリースされたものと仮定するが，過去のスケジュールを変化させることはない．言い換えると，当該ジョブが実際に仮想リリース時刻でリリースされた場合でも，TBS によって結果として同一のスケジュールとなる．アルゴリズムにおいて特に第一の前ジョブのデッドライン要因検査を行うことにより，TBS のデッドライン計算方法に順守しているため，TBS のスケジューラビリティの性質が保存される．すなわち，全タスクのプロセッサ使用率の合計が 100%以下であれば，タスクセットはスケジューラブルである．

3.3.5 仮想リリース遡及法の実装の複雑さと実行オーバヘッド

Algorithm 1 において，ループの繰り返し毎に 4 行目で除算（C/U_s）が実行される．アルゴリズムの実行オーバヘッドを抑えるために，この除算は事前に計算しておき，ループ繰り返し時にはその結果値を使用することが有効である．

記憶オーバヘッドとして，$last\_empty$ と max_dl は単一データであるため問題とはならないが，配列 dl のサイズ（要素数）を考慮する必要がある．システム稼働時間を通して全てのスロットを配列要素として用意することは非現実的であるため，アルゴリズムは遡及スロット数，すなわちループ繰り返し回数に上限を儲け，dl サイズを最大遡及スロット数とする必要がある．最大遡及スロット数による応答時間／ジッタ削減効果への影響は 4 節において評価する．

4. 評価

4.1 評価対象方式とタスクセットモデル

提案方式による平均応答時間短縮とジッタ削減への効果をタスクセットモデルに対するシミュレーションにより評価する．評価において，本稿の提案方式を，RM，EDF，Dealine Monotonic（DM）[16], および 2 節で述べた文献 [13] における TBS の適用方法と比較する．ただし，文献 [13] では全タスクに渡ってジッタを最小化するために，余剰バンド幅の各タスクへの分配量を求めることが目的であるが，本評価では余剰バンド幅を（複数の）特定タスクにのみ分配する．

DM は固定優先度アルゴリズムの一つであり，RM に対し，“デッドラインが周期と一致する” という条件を緩和し，周期よりも短い相対デッドラインを設定可能とし，相対デッドラインの長さにしたがって優先度を割当てるものである．この条件緩和を利用し，文献 [13] や 3.2 節で提案した手法と同様に，システムの余剰バンド幅を利用し，対象タスクのデッドラインを式 5 で与えることができる．ただし，式 5 内の θ_i としては，EDF のための総プロセッサ使用率の上限（100%）ではなく，RM や DM のための総使用率の上限（$U_{lub} = n(2^{1/n} - 1)$（n はタスク数）[12]）を基に得られる値を使用する．

最悪実行時間に基づくプロセッサ使用率（U_p）が 70%から 90%までの 5%おきとなる周期タスクセットを各 U_p に対して 30 個用意し，それらに対するシミュレーション結果の平均値を示す．各周期タスクの周期は 1〜100 ティックの範囲の一様分布乱数で決定し，最悪実行時間は周期の 10 分の 1 以上，かつ 3 分の 1 以下の範囲の一様分布乱数で決定した．応答時間短縮およびジッタ削減の対象となるタスクの各ジョブの実際の実行時間は最悪実行時間の 3 分の 1 以上，最悪実行時間以下の範囲の一様分布にしたがった．（対象タスクの実行はジョブごとに実行時間が変動するモデルである．）観測時間は 100,000 ティックである．なお，プロセッサ使用率が 85%あるいは 90%のときに，RM と DM の実行においてデッドラインミスが観測されたが，このことは本評価では特に考慮しない．

4.2 結果 – 平均応答時間（単一対象タスク）

タスクセット内で最も短い周期を持つタスク，最も長い周期を持つタスクを対象とした場合の平均応答時間をそれぞれ**図 3**，**図 4** に示す．図において横軸はタスクセットのプロセッサ使用率（U_p），縦軸は RM の結果を 1 として正規化した平均応答時間である．凡例内の方式名に “+RA” が付くものは，仮想リリース遡及法を組み合わせた方式であり，括弧内の数字は 3.3.5 節で述べたアルゴリズムのループ繰り返し回数に上限を設けた場合の，上限値（最大遡及スロット数）である．（“Inf” は無限大，すなわち上限値が無いことを意味する．）なお，“ATBS” は 3.2 節で述べた適応型 TBS を適用した方式である．

図 3 では EDF を除く他の全ての方式の結果がほぼ重なっている．RM と DM は最短周期である対象タスクのジョブに常に最高優先度を与えるため，応答時間が最も短くなる方法であるが，この結果から提案する方式は全て最適な RM，DM と同等の応答時間を与えていることがわ

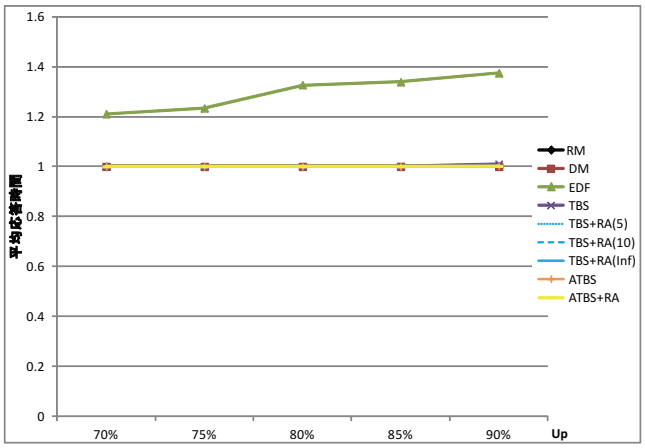

図 3 平均応答時間（最長周期タスクを対象とした場合）.

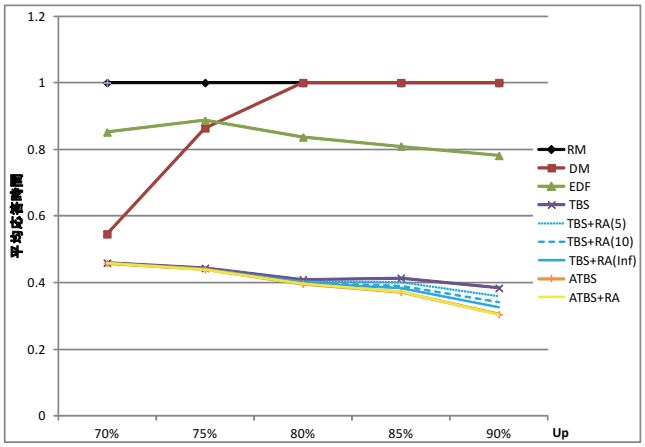

図 4 平均応答時間（最長周期タスクを対象とした場合）.

かる．

図 4 において，RM は最長周期の対象タスクに最低優先度を与えるため，応答時間は最も長くなっている．DM は U_p が低いときは余剰バンド幅の利用により RM よりも短い平均応答時間となるが，U_p が 80%以上のときは余剰バンド幅を確保できないため，RM と同じ結果となる．EDF はデッドライン時刻の比較により動的に優先度が決定するため，対象タスクの優先度は必ずしも最低ではなく，RM よりも良い結果となっている．

TBS の適用により平均応答時間は大幅に改善されることがわかる．これに仮想リリース遡及法を組み合わせることにより，アルゴリズムの繰り返し回数の上限値に応じて更に改善されている．比較対象方式の中では適応型 TBS が最も短い平均応答時間を示しており，U_p が 90%ときに，既存方法である TBS の適用に対し，最大 20.5%の改善を達成している．このことから，対象タスクに段階的なデッドラインを与えることが応答時間の改善に有効であるといえる．

なお，適応型 TBS では仮想リリース遡及法との組み合わせによる更なる改善は見られなかった．（図中で “ATBS” と “ATBS+RA” は完全に重なっている．）これは，ジョブが C_i^0 に対応する最初のデッドラインを与えられたとき，デッドラインが既に十分に短いため，多くの場合でアルゴリズムの既使用最遅デッドライン要因による検査を通過できず，遡及量がゼロになるためである．

4.3 結果 – ジッタ（単一対象タスク）

図 5，**図 6** にそれぞれ最長周期のタスクを対象とした場合の相対ジッタ，絶対ジッタを，RM の結果を 1 として正規化した値で示す．（最短周期のタスクを対象とした場合は 4.2 節の平均応答時間と同様，EDF 以外は有意の差が確認されなかったため省略する．）2 つの図の比較から確認できるように，相対ジッタと絶対ジッタは同じ傾向を示している．

RM，DM，EDF に関しては平均応答時間の評価と同様の傾向が確認できる．TBS もまた，仮想リリース遡及法との組み合わせにより向上が確認できる．平均応答時間の評価結果と異なるのは，TBS と仮想リリース遡及法の組み合わせが適応型 TBS よりも効果的である点である．このことは次のように考察できる．

平均応答時間は各ジョブの応答時間を短縮することにより向上する．適応型 TBS はジョブの実行時間にしたがっ

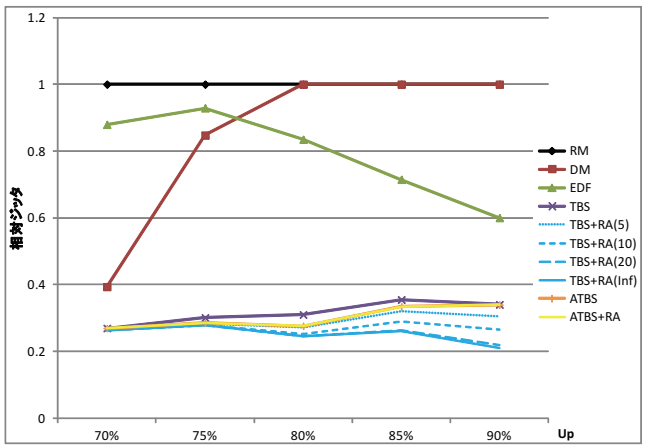

図 5 相対ジッタ（最長周期タスクを対象とした場合）.

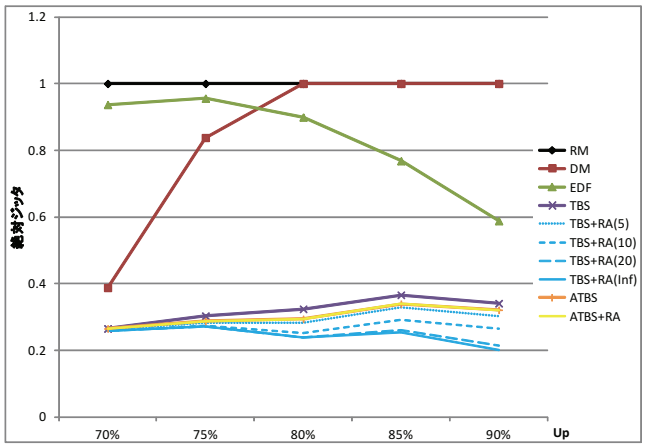

図 6 絶対ジッタ（最長周期タスクを対象とした場合）.

て最適なデッドラインを与えることにより，これを実現する．一方，ジッタを削減するためには最長となる応答時間を短縮することが重要であるが，ジョブが最悪実行時間を費やす場合，適応型 TBS は（段階的なデッドライン延長を経て）結果的に TBS と同一のデッドラインを与えることになり，TBS に対する優位性は無い．このことから最悪実行時間を費やす実行が最長応答時間となる場合のジッタに関しては適応型 TBS は有効であるとはいえない．ただし他タスクの存在により最悪実行時間のジョブが必ずしも最長応答時間となるとは限らないため，平均して応答時間を削減する適応型 TBS は TBS よりも若干短いジッタとなっている．

仮想リリース遡及法は TBS と組み合わせることにより大きな効果を生んでいる．これは仮想リリース遡及法が，たとえ最悪実行時間を費やすジョブに対する場合でもリリース時刻とデッドラインを前進させることにより応答時間を削減する性質を持っているためである．アルゴリズムの繰り返し回数の上限値にしたがって効果が変化することが確認でき，上限値が 20 のときで，繰り返し回数を無限大とした場合に近い効果を得ている．このことから，3.3.3 節における配列 dl に対し，現実的なサイズで効果を得ることが可能であるといえる．上限値を 20 とする仮想リリース遡及法を伴う TBS は，既存の手法である TBS の適用に対し，U_p が 90%のときに相対ジッタを最大 35.4%削減している．

なお，適応型 TBS の場合は，4.2 節の平均応答時間の結果と同様，仮想リリース遡及法によって更なる改善は確認されなかった．これは 4.2 節と同様の理由による．

4.4 結果 – 平均応答時間とジッタ（複数対象タスク）

4.2 節および 4.3 節では，単一のタスクを提案手法の適用対象として評価を行った．本節では周期の長いほうから 2 つのタスクを適用対象とする．余剰バンド幅を利用する方式（RM，EDF 以外全て）では，各対象タスクに余剰バンド幅を半分ずつ割り当てる．評価結果として対象の 2 つのタスクの結果値同士の平均を示す．平均応答時間，相対ジッタをそれぞれ**図 7** と**図 8** に示す．（絶対ジッタは紙面の都合上省略する．）

平均応答時間に関して，図 7 は図 4 と同様の傾向が確認できるが，対象タスク数の増加によりタスク当たりの余剰バンド幅割当てが縮小されるため，改善率が小さくなっている．また，適応型 TBS はプロセッサ使用率が低いときに，TBS よりも良い結果とはなっていない．これは，段階的にデッドラインを再設定していく過程で，同時間帯に発生した 2 つの対象タスクのジョブ同士の間で繰り返しプリエンプションが発生し，特に実行時間の長いジョブで応答時間が延長されるためである．一方，TBS では一度デッドラインが設定されるとジョブ同士の間で優先度は変化しないため，上記のプリエンプション現象は発生しない．

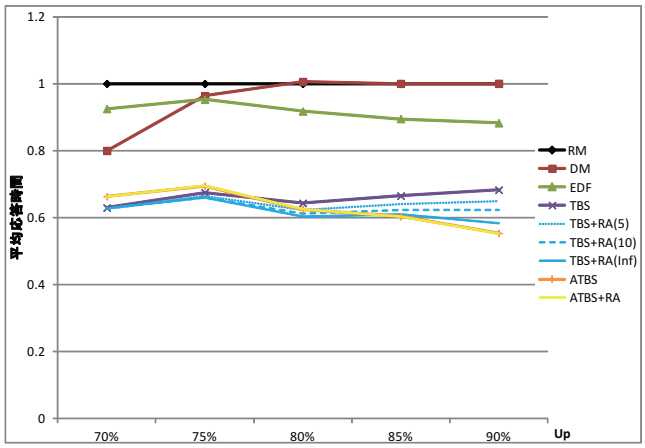

図 7 平均応答時間（最長周期の 2 つのタスクを対象とした場合）．

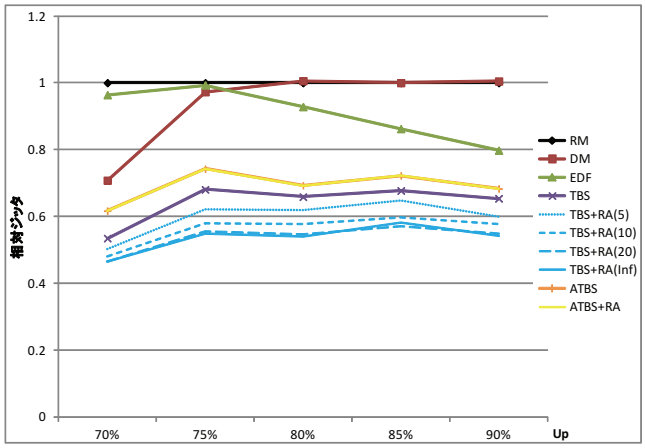

図 8 相対ジッタ（最長周期の 2 つのタスクを対象とした場合）．

相対ジッタに関して，図 8 では単一対象タスクの場合（図 5）と比較し，改善率が小さくなっている．また，適応型 TBS は前述のプリエンプション発生により，実行時間の長いジョブの応答時間が長くなることがジッタに与える影響が大きいため，TBS を適用した方法よりも改善率が小さくなっている．一方，TBS に対する仮想リリース遡及法は対象タスク数が増加した場合も効果をもたらすことが確認できる．

4.5 考察

4.2 節～4.4 節における評価結果より，対象タスク数，目的（平均応答時間短縮，あるいはジッタ削減），使用するスケジューリングアルゴリズムの間の関係について以下のように考察することができる．

- 1 つのタスクの平均応答時間の短縮を重視する場合は，適応型 TBS を対象タスクに適用することが有効である．この際，仮想リリース遡及法の付加は効果が小さいため，併用する必要性は無い．
- 2 つのタスクの平均応答時間の短縮を重視する場合は，プロセッサ使用率によって最適な適用方法が変わる．

使用率が低い場合（図 7 では 80%まで）は TBS と仮想リリース遡及法の併用，高い場合（図 7 では 85%以上）は適応型 TBS（仮想リリース遡及法無し）の使用が有力である．

- ジッタの削減を重視する場合は，対象タスクの数によらず，TBS と仮想リリース遡及法の併用が効果的である．

また，仮想リリース遡及法を使用する場合は，アルゴリズムのループ繰り返し回数の上限を 20 程度とすることで，上限を無限大とした場合の最大効果に近い結果が得られることがわかった．

なお，本稿における評価では周期が最も長い 1〜2 つのタスクを対象とした結果を示したが，中間的な長さの周期を持つタスクを対象とした場合は，改善率が低くなるが方式間の関係に同様の傾向が認められた．（紙面の都合上，詳細は割愛する．）

5. おわりに

本研究では，重要度の高い周期タスクの平均応答時間の短縮およびジッタの軽減を達成する方式として，適応型 TBS の適用法と仮想リリース遡及法を提案した．適応型 TBS は繰り返し実行されるタスクの実行時間の変動に対して適応（段階）的にデッドラインを設定する技法であり，仮想リリース遡及法は過去のスケジュールに影響を与えない範囲でリリース時刻を仮想的に前倒しさせることにより短いデッドラインを与える方法である．評価において，既存手法である TBS の適用と比較し，適応型 TBS は応答時間を最大 20.5% 短縮した．また，仮想リリース遡及法を併用する方法は TBS のみの適用と比較し，相対ジッタを最大 35.4% 改善した．

組込みシステムにおいてクリティカリティ混在化が進むにつれ，システム内に指向の異なるタスクが存在することになる．本研究の結果から，各タスクの改善／向上すべき性能指標にあわせて適用するスケジューリング方式を選択することが有効であるといえる．

本稿のシミュレーション評価では，仮想リリース遡及法が現実的な記憶オーバヘッドで実現可能であることは示されたが，アルゴリズム内のループの実行オーバヘッドによるシステム性能への影響は未評価であるため，これが今後の課題の一つである．また，今回は実行時間や周期に関して確率分布モデルにしたがって生成したタスクセットを対象としたシミュレーションを行ったが，実際の実行時間の変動を表現するためには，実アプリケーションプログラムを使用した評価が望まれる．これも併せて今後の課題とする．

参考文献

[1] Åström, K.J., Wittenmark, B.: *Computer-Controlled Systems: Theory and Design*, 3rd ed., Prentice Hall (1997).
[2] Martí, P., Fuertes, J.M., Fohler, G., Ramamritham, K.: *Jitter Compensation for Real-Tiime Control Systems*, Proc. of IEEE Real-Time Systems Symposium, pp.39–48 (2001).
[3] Buttazzo, G.C.: *Rate Monotonic vs. EDF: Judgment Day*, Journal of Real-Time Systems, Vol.29, No.1, pp.5–26, (2005).
[4] 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 (2009).
[5] 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 (2010).
[6] Lluesma, M., Cervin, A., Balbastre, P., Ripoll, I., Crespo, A.: *Jitter Evaluation of Real-Time Control Systems*, Proc. of IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications, pp.257–260 (2006).
[7] 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).
[8] Tanaka, K.: *Adaptive Total Bandwidth Server: Using Predictive Execution Time*, Proc. of Intl. Embedded Systems Symposium, pp.250–261 (2013).
[9] 田中 清史：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 ―，組込みシステムシンポジウム（ESS） 2013 論文集（2013）．
[10] Buttazzo, G., Cervin, A.: *Comparative Assessment and Evaluation of Jitter Control Methods*, Proc. of Intl. Conf. on Real-Time and Network Systems, pp.137–144 (2007).
[11] Balbastre, P., Ripoll, I., Vidal, J., Crespo, A.: *A Task Model to Reduce Control Delays*, Journal of Real-Time Systems, Vol.27, No.3, pp.215–236 (2004).
[12] Buttazzo, G.C.: *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, Third Edition, Springer, (2011).
[13] Baruah, S., Buttazzo, G., Gorinsky, S., Lipari, G.: *Scheduling periodic task systems to minimize output jitter*, Proc. of IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications, pp. 62–69 (1999).
[14] Spuri, M., Buttazzo, G.C.: *Efficient Aperiodic Service under Earliest Deadline First Scheduling*, Proc. of IEEE Real-Time Systems Symposium, pp.2–11 (1994).
[15] Spuri, M., Buttazzo, G., Sensini, F.: *Robust Aperiodic Scheduling under Dynamic Priority Systems*, Proc. of IEEE Real-Time Systems Symposium, pp.210–219 (1995).
[16] Leung, J., Whitehead, J.: *On the Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks*, Performance Evaluation, Vol.2, No.4, pp.237–250 (1982).