情報処理学会論文誌　コンピューティングシステム　Vol. 1　No. 2　83–95 (Aug. 2008)

マルチコアプロセッサにおける非周期タスクの応答性向上

加藤真平[†1]　山崎信行[†1]

本論文では，オンチップマルチプロセッサにおいて，周期リアルタイムタスクがデッドラインまでに完了することを保証して，さらに非周期タスクの応答時間をできる限り短縮するためのテンポラルマイグレーション手法を提案する．テンポラルマイグレーション手法は，非周期タスクが到着した際に，そのプロセッサ上で優先度の高い周期タスクが存在する場合は，リアルタイム性を保証できる範囲でそれらの周期タスクを一時的にほかのプロセッサにマイグレーションし，プロセッサ時間を非周期タスクに譲ることで応答性を向上させる．本論文では，テンポラルマイグレーション手法を従来のシングルプロセッサ用 Total Bandwidth Server（TBS）アルゴリズムと組み合わせたアルゴリズムを設計する．シミュレーションによる評価では，テンポラルマイグレーション手法を用いることで，TBS アルゴリズムの非周期タスクに対する応答性を劇的に向上させることができることを示す．

Improving Responsiveness to Aperiodic Tasks on Multicore Processors

Shinpei Kato[†1] and Nobuyuki Yamasaki[†1]

This paper presents the temporal migration technique for reducing the response time of aperiodic tasks as much as possible, with guaranteeing periodic reak-time tasks to meet their deadlines on chip multiprocessors. In the temporal migration technique, periodic tasks with higher priorities than arriving aperiodic tasks are temporarily migrated onto another processor, as long as the periodic timing constraints can be guaranteed, in order to hand over processor time to the pending aperiodic tasks and improve the reponsiveness. This paper designs an algorithm that combines the temporal migration technique to the traditional Total Bandwidth Server (TBS) algorithm devised for single processors. Simulation studies show that the algorithm can improve the aperiodic responsiveness dramatically, compared to the case without temporal migration.

1. はじめに

リアルタイムシステムの研究では，しばしば周期リアルタイム処理に焦点が当てられてきた．実際，多くの制御およびマルチメディアアプリケーションが周期処理で実現されている．それゆえ，これまでは周期タスクのスケジューリングがリアルタイムシステムに関する研究の主な題材であった．今日，シングルプロセッサプラットフォームでは，Earliest Deadline First（EDF）アルゴリズム[8] を用いることで周期タスクを最適にスケジューリングできることが知られている．

近年のリアルタイムシステムでは，より複雑な計算モデルが必要とされている．たとえば，ロボットシステムは人間の行動や自然現象が動的に発生する環境で動作しなければならない．この場合，いくつかのタスクは非周期的に到着することになり，タスクの種類によって実行時間の予測は可能であるが，それらの到着時間を前もって知ることは不可能である．多くの非周期処理はデッドラインまでに完了するというリアルタイム性よりも，できる限り早く完了するという応答性を要求する．例としてロボットシステムにおける音声認識があげられる．人間がロボットに話しかけたとき，ある時間までに応答しなければならないという制約はないが，人間はロボットがなるべく早く応答してくれることを期待する．一方で，実行中の周期タスクのリアルタイム性は保持しなければならない．そのため，システムは周期タスクの時間制約を満たすことができる範囲で，非周期タスクの応答時間をできる限り短縮することが求められる．

非周期タスクを扱う最も単純な方法は，周期タスクが使用しないプロセッサ時間を非周期タスクに割り当てることである．このバックグラウンド手法は簡単ではあるが，十分な応答性を実現することはできない．一般的に，非周期タスクへの応答性を高めるために，システムは非周期サーバを実行する．非周期サーバとは，非周期タスクの処理を目的とした特別な周期タスクである．通常の周期タスクと同様に，サーバは周期と実行時間（サーバ容量）から特徴付けられる．サーバは周期タスクと同じアルゴリズムでスケジューリングされ，1度ディスパッチされると，サーバ容量の範囲内で非周期タスクを処理する．残念ながら，シングルコアプラットフォームを対象としたサーバアルゴリズムはこれまでに数多く提案されているが，マルチプロセッサプラットフォームの特性を有効利用できるサーバアルゴリズム

†1 慶應義塾大学
Keio University

はほとんど提案されていない．

本論文では，マルチプロセッサプラットフォーム，とりわけマルチコアプロセッサを対象として，周期リアルタイムタスクがデッドラインまでに完了することを保証して，さらに非周期タスクの応答時間をできる限り短縮するためのテンポラルマイグレーション手法を提案する．提案手法は，効果的にタスクマイグレーションを発生させることで，応答性を劇的に向上させる．本論文では，あらかじめ周期タスクを特定のコアに割り当ててから実行するパーティション方式[3] のスケジューリングを対象とする．別の方式として，グローバル方式があるが，実際のリアルタイム OS においてもその簡単さから，しばしばパーティション方式が採用されている．非周期タスクはデッドラインを持たないものと仮定し，非周期時間の応答時間を短縮することに焦点を当てる．

本論文の構成は以下のとおりである．次章では，本研究の背景および関連研究について言及する．3 章では，本研究が対象とするシステムモデルを説明する．そして，4 章において，テンポラルマイグレーション手法と，それを適用したサーバアルゴリズムを提案する．5 章では，提案手法およびアルゴリズムの性能をシミュレーションで評価する．最後に，6 章で本研究の結論と今後の課題を述べる．

2. 背景および関連研究

Spuri らは，EDF スケジューリングにおいて高い応答性を実現できる非周期サーバ手法を提案した[11),12)]．彼らはまず，Lehoczky らが静的優先度スケジューリング向けに提案した Deferrable Server（DS）アルゴリズムと Priority Exchange（PE）アルゴリズム[7] を動的優先度スケジューリング向けに応用した Dynamic DS（DDS）アルゴリズムと Dynamic PE（DPE）アルゴリズムを提案した．これら 2 つのアルゴリズムは，タスクの優先度付けに関して DS と PE の静的優先度割当てに起因する制限が課されるので，性能に限界がある．そこで，Spuri らは，EDF スケジューリングに対してより効率的な Total Bandwidth Server（TBS）アルゴリズムを提案した．TBS アルゴリズムは，割り当てられた帯域幅（CPU 使用率）の範囲内で非周期タスクに仮想的なデッドラインを割り当て，EDF でスケジューリングするアルゴリズムである．我々の知る範囲では，TBS アルゴリズムは，計算が簡単であり，かつ高い応答性を実現できるという点で最も効率的なサーバアルゴリズムである．Spuri らはさらに，応答時間に関して最適な Earliest Deadline as Late as possible server（EDL）アルゴリズムと，準最適な Improved Priority Exchange（IPE）アルゴリズムも提案した．EDL アルゴリズムは，Davis らが提案した静的優先度割当てに基づく Slack Stealing アルゴリズム[6] の動的優先度拡張である．一方，IPE アルゴリズムは，EDL アルゴリズムにおけるアイドル時間を効果的に利用するために DPE アルゴリズムの概念を利用したアルゴリズムである．これらのアルゴリズムは高い応答性を実現できるが，実行時の計算量やメモリ使用量が非常に大きいため実用性は低いとされている．

シングルプロセッサを対象としたサーバアルゴリズムが数多く提案されている一方で，マルチプロセッサプラットフォームの特性を有効利用できるサーバアルゴリズムはほとんど提案されていない．Baruah らは，TBS アルゴリズムのマルチプロセッサ向け拡張方法を考案し，グローバル方式に基づく EDF スケジューリングに TBS アルゴリズムを適用した場合のスケジュール可能条件を証明した[5]．しかしながら，アルゴリズムの設計やポリシ自体は TBS アルゴリズムをそのまま流用しており，Baruah らの目的は性能改善ではなく，むしろ TBS アルゴリズムをマルチプロセッサプラットフォームに適用することであった．

Andersson らは，グローバル方式とパーティション方式の両方に関して，マルチプロセッサプラットフォーム向けの非周期タスクスケジューリング手法を提案した[1),2)]．彼らの研究では，非周期タスクはデッドラインを持っており，非周期タスクをスケジューリングした場合に，スケジュール可能性を低下させないことが目的であった．彼らの設計したグローバル方式のスケジューリングでは，システム使用率が 50%以下であれば，あらゆる周期および非周期タスクをデッドラインまでに完了できることが保証できる．同様に，彼らの設計したパーティション方式のスケジューリングでは，システム使用率が 31%以下であれば，あらゆる周期および非周期タスクはデッドラインまでに完了できることが保証できる．非周期タスクが存在する環境でのリアルタイム性の保証はそれまでに証明されていなかったので，Andersson らの功績は大きいと考えられるが，非周期タスクへの応答性に関しては焦点が当てられていなかった．

我々の知る範囲では，マルチプロセッサパーティション方式を対象として，非周期タスクへの応答性を向上させるための効果的な手法は提案されていない．上述したとおり，文献 5) において，Baruah らは TBS アルゴリズムをマルチプロセッサ向けに拡張しているが，これはグローバル方式を対象としており，かつマルチプロセッサで TBS アルゴリズムを流用することが目的であったため，マルチプロセッサの並列性を有効利用して応答性を向上させるような手法ではない．そこで，本論文では，パーティション方式向けに，マルチプロセッサの並列性を有効利用して非周期タスクへの応答性を向上可能な手法を提案し，その提案手法に基づいて TBS アルゴリズムを改良する．

3. システムモデル

システムは M 個のプロセッシングユニット（コア）$P_1, P_2, \ldots, P_M$ を持つマルチコアプロセッサとする．プログラム（タスク）はプロセッサ間で共有可能である．本論文では，理論的なプロセッサ間通信のオーバヘッドは無視する．いい換えると，プロセッサ間のタスクマイグレーションのコストは考慮に入れない．近年のオンチップマルチプロセッサでは，ハードウェアのサポートによってタスクマイグレーションのコストを劇的に削減することも可能になっているため，このような仮定も妥当であると考えられる．たとえば，Yamasaki が開発した Responsive Multithreaded（RMT）Processor[16] は，ハードウェア内に 32 個分のコンテキスト専用のキャッシュ領域を有しており，そのキャッシュ領域と CPU を専用バスで接続しているために，マイグレーションを含めたすべてのコンテキストスイッチが 4 クロックで完了できる設計となっている．このようなハードウェア構成は RMT Processor 特有な制限ではないため，汎用的にマルチコアプロセッサにおいても同じ機能を導入することが可能であり，コア間を専用バスで接続することで，タスクマイグレーションにかかるコストを劇的に削減できると考えられる．

システムには周期タスクセット $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ を最初に与える．i 番目の周期タスクを $\tau_i(C_i, T_i)$ と表記する．C_i と T_i は，それぞれタスクの最悪実行時間と周期である．タスクの CPU 使用率は $U_i = C_i/T_i$ で表す．各タスクは一連のジョブを周期的に生成する．タスク τ_i の j 番目のジョブを $\tau_{i,j}$ と表記し，時刻 $r_{i,j}$ にリリースされて時刻 $d_{i,j} = r_{i,j} + T_i = r_{i,j+1}$ がデッドラインとする．時刻 t でのジョブ $\tau_{i,j}$ の残り実行時間を $\tilde{c}_{i,j}(t)$ と表記する．周期タスクセットの CPU 使用率の合計を $U(\Gamma) = \sum_{\tau_i \in \Gamma} U_i$ とする．各周期タスクは特定のプロセッサに割り当てられている（パーティションされている）ものとする．x 番目のプロセッサ P_x に割り当てられた周期タスクセットを Π_x とし，プロセッサ使用率は $U(\Pi_x) = \sum_{\tau_i \in \Pi_x} U_i$ となる．各プロセッサは EDF スケジューラを持ち，プロセッサ間で独立にタスクをスケジューリングする．

非周期タスクは順番に到着するものとする．プロセッサ P_x 上で k 番目の非周期タスクを $\alpha_{x,k}(a_{x,k}, E_{x,k})$ と表記する．$a_{x,k}$ はタスクの到着時刻であり，$E_{x,k}$ はタスクの最悪実行時間とする．タスクが時刻 $f_{x,k}$ に実行を完了したとして，タスクの応答時間を $R_{x,k} = f_{x,k} - a_{x,k}$ と定義する．すべての x および k に対して，$a_{x,k} < a_{x,k+1}$ が成り立つものとする．非周期タスクセットの負荷 $U(\alpha)$ は，平均サービス率 μ と平均到着率 λ によって決定され，$U(\alpha) = \lambda/\mu$ とする．

すべてのタスクはプリエンプティブで独立であるものとする．すなわち，I/O 処理やタスク間同期は本論文では考慮に入れない．このような性質を考慮する場合には，優先度上限プロトコル（PCP）[10] やスタックリソースポリシ（SRP）[4] を適用すればよい．また，厳密に周期タスクのリアルタイム性を保証するためには，最悪実行時間を正確に見積もる必要があるが，タスクマイグレーションを行うとキャッシュや分岐予測などの性能が変動し，時間解析が複雑になる．本研究もタスクマイグレーションを利用するので，時間解析が問題となる．そのため，提案手法はキャッシュではなくスクラッチパッドメモリを搭載し，単純なパイプライン機構を採用している組み込みプロセッサに特に効果的である．キャッシュを搭載するプロセッサに提案手法を適用する場合には，文献 13) にあるような実験に基づく時間解析手法と合わせて利用する必要がある．非周期タスクに関しては，厳密なデッドラインは持っていないので最悪実行時間を正確に見積もる必要はない．しかしながら，できる限り早く実行を完了する必要があるので，タスクマイグレーションによる性能低下は問題となる．そこで，非周期タスクはマイグレーションを起こさないという方針で提案手法を設計する．

4. テンポラルマイグレーション手法

本章では，マルチコアプロセッサにおいて，周期タスクのリアルタイム性を保証して，さらに非周期タスクの応答性を向上させるためのテンポラルマイグレーション手法を提案する．本手法は，シングルプロセッサ用の様々なサーバアルゴリズムと組み合わせることが可能である．本論文では，TBS アルゴリズムと組み合わせた TBS-TM（TBS with Temporal Migration）アルゴリズムを設計する．

4.1 アプローチ

テンポラルマイグレーション手法は，非周期タスクが到着した際に，そのプロセッサ上で優先度の高い周期タスクが存在する場合は，リアルタイム性を保証できる範囲でそれらの周期タスクを一時的にほかのプロセッサにマイグレーションし，プロセッサ時間を非周期タスクに譲ることで応答性を向上させるという方針に基づいている．**図 1** に概念図を示す．この例では，プロセッサ P_1 に非周期タスクよりも優先度の高い 2 つの周期タスクがキューイングされており，マイグレーションなしのリアルタイムスケジューリングでは，非周期タスクは 2 つの周期タスクの実行完了を待たなくてはならない．テンポラルマイグレーション手法では，これら 2 つの周期タスクを一時的に P_2 や P_3 にマイグレーションすることで，P_1 上の非周期タスクは即座に実行することができる．ここで問題となるのが，一時的にマイグレーションを起こす周期タスクの時間制約であり，マイグレーション先のプロセッサでデッ

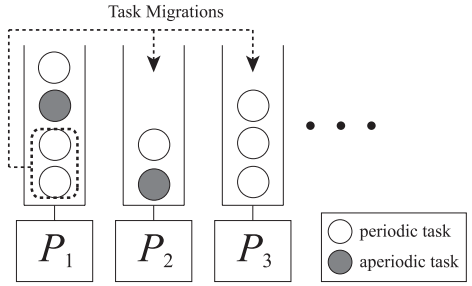

図 1　テンポラルマイグレーションの概念
Fig. 1　Concept of temporal migration.

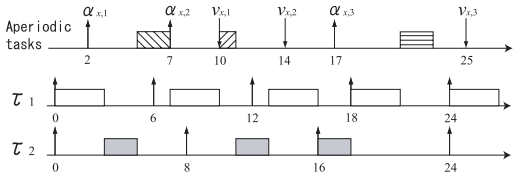

図 2　TBS アルゴリズムの例
Fig. 2　The TBS algorithm example.

ドラインまでに完了できることを保証する必要がある．

4.2　TBS アルゴリズムの概要

本節は，TBS アルゴリズムを簡単に説明する．TBS アルゴリズムでは，非周期タスクが到着したときに，その非周期タスクに仮想的にデッドラインを割り当て，周期タスクと一緒に EDF スケジューリングする．プロセッサ P_x でサーバ τ_x^{srv} が実行されているとする．サーバ τ_x^{srv} は（プロセッサ P_x 上で）k 番目に到着した非周期タスク $\alpha_{x,k}$ に対して，到着時刻と実行時間，前回の非周期タスクの仮想デッドライン，サーバの帯域幅から式 (1) で求められる仮想デッドライン $v_{x,k}$ を割り当てる．ここで，$v_{x,0} = 0$ とする．

$$v_{x,k} = \max\{a_{x,k}, v_{x,k-1}\} + \frac{E_{x,k}}{U_x^{srv}} \tag{1}$$

図 2 は，P_x 上で 2 つの周期タスク $\tau_1(3,6)$ と $\tau_2(2,8)$ を EDF スケジューリングしているときの TBS アルゴリズムの実行例を示している．サーバの帯域幅は $U_x^{srv} = 1-(U_1+U_2) = 0.25$ としている．1 番目の非周期タスク $\alpha_{x,1}(2,2)$ が時刻 2 に到着したとき，式 (1) に基づいて仮想デッドライン $v_{x,1} = a_{x,1} + E_{x,1}/U_x^{srv} = 2 + 2/0.25 = 10$ が割り当てられる．$v_{x,1}$ は実行中の周期ジョブ $\tau_{1,1}$ と $\tau_{2,1}$ よりも遅いデッドラインなので，$\alpha_{x,1}$ は $\tau_{1,1}$ と $\tau_{2,1}$ が完了してから実行される．同様に，2 番目の非周期タスク $\alpha_{x,2}(7,1)$ が時刻 7 に到着したとき，仮想デッドライン $v_{x,2} = \max\{a_{x,2}, v_{x,1}\} + E_{x,2}/U_x^{srv} = 10 + 4 = 14$ が割り当てられるが，このとき $\tau_{1,2}$ の方がデッドラインが早いので，$\tau_{1,2}$ の完了を待たなければならない．最後に，3 番目の非周期タスク $\alpha_{x,3}(17,2)$ が時刻 17 に到着し，仮想デッドラインが $v_{x,3} = a_{x,3} + E_{x,3}/U_x^{srv} = 25$ が割り当てられるが，実行中の τ_1 と τ_2 のジョブの方が早いデッドラインを持っているので，それらのジョブが完了してから実行される．

4.3　アルゴリズム設計

本節では，TBS アルゴリズムにテンポラルマイグレーション手法を適用した TBS-TM アルゴリズムを設計する．任意の非周期タスク $\alpha_{x,k}$ が時刻 t にプロセッサ P_x に到着したとする．プロセッサ P_x では，帯域幅 U_x^{srv} のサーバが実行されており，式 (1) に基づいて $\alpha_{x,k}$ に仮想デッドライン $v_{x,k}$ が割り当てられたものとする．このとき，以下の手続きに従って周期タスクの一時的なマイグレーションが発生する．

(1)　$\tau_{i,j}$ を最も早いデッドラインを持つ実行可能な周期ジョブとし，現在の周期以内でまだマイグレーションしていないものとする．もし，このようなジョブが存在しなければ，テンポラルマイグレーションは発生せず，手続きは終了する．

(2)　条件式 (2) を満たす任意のプロセッサ P_y を検索する．ここで，$v_{y,l}$ は時刻 t 以前に P_y に到着した最後の非周期タスク $\alpha_{y,l}$ の仮想デッドラインとし，$\tilde{c}_{i,j}(t)$ は $\tau_{i,j}$ の残り実行時間とする．

$$\max\{t, v_{y,l}\} + \frac{\tilde{c}_{i,j}(t)}{U_y^{srv}} \le d_{i,j} \tag{2}$$

もし，このようなプロセッサが見つからなければ，テンポラルマイグレーションは発生せず，手続きは終了する．

(3)　$\tau_{i,j}$ を一時的に P_y にマイグレーションする．ここで，$\tau_{i,j}$ は，P_y 上での $l+1$ 番目の非周期タスクと見なされ，式 (3) で求められる仮想デッドライン $v_{y,l+1}$ が割り当てられる．よって，時刻 t 以降に P_y に到着する非周期タスクのインデックス付けは $l+2$ から始まる．

$$v_{y,l+1} = \max\{t, v_{y,l}\} + \frac{\tilde{c}_{i,j}(t)}{U_y^{srv}} \tag{3}$$

(4)　$\tau_{i,j}$ が一時的に P_x から消えるので，P_x のサーバ帯域幅 U_x^{srv} を一時的に増加させることができる．よって，$\alpha_{x,k}$ には式 (4) で求められる新しい仮想デッドラインが割

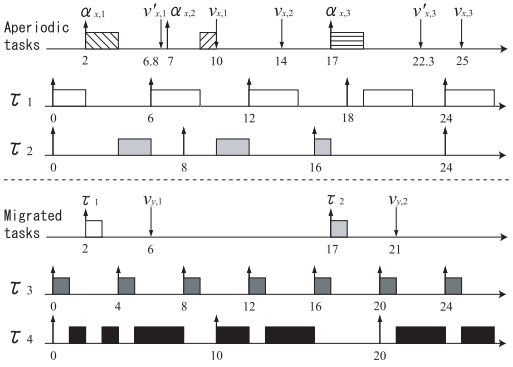

図 3　TBS-TM のスケジューリング例
Fig. 3　Scheduling example of TBS-TM.

り当てられる．

$$v'_{x,k} = \max\{t, v_{x,k-1}\} + \frac{E_{x,k}}{U_x^{srv} + \frac{\tilde{c}_{i,j}(t)}{T_i}} \tag{4}$$

(5)　タスク τ_i が P_y でスケジューリングされる期間はジョブの周期以内であり，次のジョブ，すなわち $\tau_{i,j+1}$ は再び P_x に戻されて実行される．

TBS-TM アルゴリズムでは，マイグレーションされるジョブ $\tau_{i,j}$ は非周期タスク $\alpha_{y,l+1}$ として扱われる．よって，次に P_y に到着する非周期タスク $\alpha_{y,l+2}$ の仮想デッドラインを式 (1) で求める場合には $v_{y,l}$ ではなく $v_{y,l+1}$ を使う必要があることに注意されたい．また，$\tau_{i,j}$ は周期ジョブなので，応答時間を短縮する必要はない．よって，P_y 上で仮想デッドライン $v_{y,l+1}$ ではなくて，本当のデッドライン $d_{i,j}$ に基づいてスケジューリングしてもよいが，このようなデッドライン割当てを行うと P_y のサーバ帯域幅が一時的に減少し，リアルタイム性保証のために現在割り当てられている非周期タスクの仮想デッドラインを再計算する必要が生じる．そのため，本論文では簡単に TBS アルゴリズムに基づくデッドライン割当てを採用する．また，ステップ (1) において，すでにマイグレーションを起こしたジョブは次のマイグレーションの対象にはならないことが約束される．仮に，2 段階以上のマイグレーションを許したとすると，非周期タスクへの応答性は向上させることができるかもしれないが，理論と実装が複雑になってしまうため，本論文では 1 段階のマイグレーションに限定する．デッドライン割当てや複数段階のマイグレーションの検討は今後の課題とする．

2 つのプロセッサ P_x と P_y における，TBS-TM アルゴリズムの動作例を**図 3** に示す．周期タスクは EDF アルゴリズムに従ってスケジュールされる．図 2 と同様に，2 つの周期タスク $\tau_1(3,6)$ と $\tau_2(2,8)$ が P_x 上でスケジュールされ，さらに 2 つの周期タスク $\tau_3(1,4)$ と $\tau_4(5,10)$ が P_y 上でスケジュールされているものとする．P_x と P_y のサーバ帯域幅はともに 0.25 とする．アルゴリズムの説明を簡単にするために，P_y には非周期タスクは到着しないものとする．P_x 上で最初の非周期タスク $\alpha_{x,1}$ が時刻 2 に到着したとき，テンポラルマイグレーション手法を用いない場合は，図 2 で説明したように仮想デッドライン $v_{x,1} = 10$ が割り当てられる．一方で，テンポラルマイグレーション手法を導入すると，$\tau_{1,1}$ は $t + \tilde{c}_{1,1}(t)/U_y^{srv} = 2 + 1/0.25 = 6 \leq d_{1,1} = 6$ となり条件式 (2) を満たすので P_y に一時的にマイグレーションされる．そして，式 (3) で求められる仮想デッドライン $v_{y,1} = 6$ に基づいてスケジュールされる．そして，$\alpha_{x,1}$ は式 (4) より新しい仮想デッドライン $v'_{x,1} = 2 + 2/(0.25 + 1/6) = 6.8$ が割り当てられる．$\alpha_{x,1}$ の仮想デッドラインは $\tau_{2,1}$ のデッドラインよりも早く，また $\tau_{1,1}$ は一時的に P_y にマイグレーションされているので，$\alpha_{x,1}$ は即座に実行することができる．対照的に，2 番目の非周期タスク $\alpha_{x,2}$ が到着したときには，$\tau_{1,2}$ は $t + \tilde{c}_{1,2}(t)/U_y^{srv} = 7 + 2/0.25 = 15 > d_{1,2} = 12$ となり条件式 (2) を満たさないので，マイグレーションされない．$\tau_{2,1}$ もすでに実行が完了しており実行可能状態ではないので，$\alpha_{x,2}$ の仮想デッドラインは $v_{x,2} = 14$ で変わらない．しかしながら，時刻 2 で $\tau_{1,1}$ が P_y にマイグレーションされ，P_x 全体のスケジュールが早まっているため，$\alpha_{x,2}$ の実行も時間 1 だけ早まっていることに注意されたい．最後に，3 番目の非周期タスク $\alpha_{x,3}$ が到着したとき，$\tau_{2,3}$ が条件式 (2) を満たし，P_y にマイグレーションできる．それゆえ，$\alpha_{x,3}$ の仮想デッドラインは $v'_{x,3} = 17 + 2/(0.25 + 1/8) = 22.3$ に短縮され，周期ジョブのデッドラインよりも早いので即座に実行することができる．図 2 に示した従来の TBS アルゴリズムの例と比べると，TBS-TM を用いることでプロセッサ P_x に到着した非周期タスクの応答時間が短縮されており，テンポラルマイグレーション手法の効果が確認できる．

4.4　プロセッサ選択

4.3 節で示したマイグレーション手続きでは，条件式 (2) を満たす任意のプロセッサにジョブ $\tau_{i,j}$ をマイグレーションすると述べた．しかしながら，実際には条件式 (2) を満たすプロセッサの中でも，できる限り応答時間を短縮できるプロセッサにマイグレーションすべきである．ここで，全体として応答時間を最も短縮できる最適なプロセッサの選択は，スケジューリングの先読みや組合せ最適化などの問題により非常に困難であると考えられる．それゆえ，本論文ではマイグレーション先のプロセッサ選択には，以下の 3 つの発見的手法を利用する．

- First-Fit（FF）手法：条件式 (2) を満たす最初の（最もインデックスの小さい）プロセッサに $\tau_{i,j}$ をマイグレーションする．
- Best-Fit（BF）手法：条件式 (2) を満たし，かつ条件式 (2) の両辺の差を最小にする，すなわち式 (3) で求められる $v_{y,l+1}$ と $d_{i,j}$ の差を最小にするプロセッサに $\tau_{i,j}$ をマイグレーションする．
- Worst-Fit（WF）手法：条件式 (2) を満たし，かつ条件式 (2) の両辺の差を最大にする，すなわち式 (3) で求められる $v_{y,l+1}$ と $d_{i,j}$ の差を最大にするプロセッサに $\tau_{i,j}$ をマイグレーションする．

FF 手法を採用する TBS-TM アルゴリズムを TBS-TM-F と表記することにする．同様に，BF 手法および WF 手法を採用する TBS-TM アルゴリズムをそれぞれ TBS-TM-B および TBS-TM-W と表記することにする．これら 3 つのアルゴリズムの性能を理論的に解析し，比較することは発見的手法の性質から困難であるため，これらのアルゴリズムの性能の優劣はシミュレーションによって評価することにする．より洗練された発見的手法や，より応答時間を短縮できるプロセッサの選択方法に関しては今後の課題とする．

5. 評　　価

本章では，テンポラルマイグレーション手法と組み合わせた TBS アルゴリズムのシミュレーション結果を示す．純粋な TBS アルゴリズムと比較することで，応答性に関してテンポラルマイグレーション手法の効果を評価する．応答性の評価指標は，様々なシステム負荷における非周期タスクの平均応答時間とした．

5.1 シミュレーション環境

シミュレーションには，RMT Processor 用の命令レベルシミュレータ rmtsim [⋆1]を使用した．RMT Processor は，最大 8-way の Simultaneous Multithreading（SMT）[15] 機能を備えている．SMT とマルチコアは，ハードウェア資源の利用効率の点でアーキテクチャが大きく異なるが，rmtsim では演算器の個数を無限とし，演算の遅延はすべて 1 サイクルという理想的な設定で設計されているので，マルチコアシミュレータとしても利用可能である．タスクの実行には，rmtsim 用に開発した軽量リアルタイムカーネル TF-light [⋆2]を使用した．TF-light には，単純な優先度スケジューラしか用意されていないので，EDF アルゴリズムと TBS アルゴリズムを別途実装して使用した．本シミュレーションでは，すべてのタスクを前もって TF-light に組み込み，それを rmtsim で実行することでスケジューラのオーバヘッドも含めて性能計測を行った．

プロセッサの動作速度は 100 MHz に設定した．rmtsim は最大 8 スレッドまで同時実行が可能なので，プロセッサ数が 2，4，8 の場合を想定して評価を行った．ここでは，アルゴリズムの性能を計測するためのシミュレーションを考えているため，以下に述べる方法で生成させる各タスクは意味のあるプログラムを実行するのではなく，CPU の時間情報をポーリングして決められた時間が経過したら実行を終了するものとした．

評価用のタスクセットは，我々の研究室で開発を行っているロボット [14] のアプリケーションを参考にした．周期タスクセットのシステム使用率は 60%，すなわちタスクの CPU 使用率の合計は $0.6 \times M$ と設定した．対象ロボットでは，1 つのタスクが 1 つのプロセッサの 50%以上を占有することはほとんど考えられないので，各タスクの CPU 使用率は $[0.01, 0.5]$ の範囲で無作為に決定することにした．また，対象ロボットのタスクの周期は，アプリケーションの種類に依存し，時間粒度の小さい制御では $1\,ms$，時間粒度の大きいマルチメディア処理では $30\,ms$ 程度に設定されるので，各タスクの周期は $[1\,ms, 30\,ms]$ の範囲で無作為に決定した．実行時間は，$C_i = U_i T_i$ で求められる．プロセッサ数が 2，4，8 の場合をシミュレーションするので，タスクの CPU 使用率の合計が各々1.2，2.4，4.8 となるような 3 つのタスクセットを上述した方法に従って生成した．3 つのタスクセットはシステム使用率が異なるので，それぞれタスク数も異なることに注意されたい．

非周期ワークロードは M/M/1 待ち行列モデルに基づいて生成した．非周期タスクの到着時刻などは前もって予測できないので，到着時刻はポアソン分布に従い，実行時間は指数分布に従う一般的なポアソン到着モデルを採用した．各非周期タスクが到着するプロセッサは無作為に選択することにした．非周期タスクの応答時間は，到着間隔や実行時間に依存すると考えられる．そのため，評価では平均サービス率を $\mu = 0.1$ と $\mu = 0.2$ の 2 通り用意し，各平均サービス率に関して平均到着率を徐々に増加させていって非周期タスクの平均応答時間を測定した．具体的には，非周期タスクセットの負荷が $U(\alpha) = \lambda/\mu$ が 5%から 35%になるまで平均到着率を増加させていった．そして，各負荷に対して，非周期タスクの平均応答時間を測定した．同じ負荷であっても，アルゴリズムが変わるごとにタスクセットが変わってしまっては公平な評価ができないため，プロセッサのメモリ領域が許す範囲で，上述した方法に従って生成された非周期タスクをあらかじめ静的に非周期タスク用のキューに格納しておくことにした．各プロセッサにスケジューラが 1 つ用意されるので，非周期タ

⋆1 http://drtp.dip.jp/svn/RmtSimulator/よりダウンロード可能.
⋆2 http://drtp.dip.jp/svn/tflight/よりダウンロード可能.

スクキューも各プロセッサに 1 つ用意されることになる．周期タスクセットと同様に，プロセッサ数が 2，4，8 の場合に対して，それぞれ非周期タスクセットを用意した．

周期タスクセットは EDF-FF アルゴリズム[9] でスケジューリングされる．EDF-FF アルゴリズムは，FF 手法に基づいて各周期タスクを特定のプロセッサに割り当て，すべてのタスクがパーティションされたあとは，各プロセッサで EDF スケジューラを実行してタスクをスケジューリングする．通常，FF 手法は各プロセッサを 100%使用する最適な割当ては実現できない．そのため，各プロセッサで利用可能な残りの CPU 使用率をサーバの帯域幅に割り当てる．いい換えると，プロセッサ P_x のサーバ帯域幅は $U_x^{srv} = 1 - U(\Pi_x)$ となる．

5.2 実　　装

アルゴリズムの実装には様々な方法が考えられる．本節では，評価用に実装した TF-light のスケジューラについて述べる．EDF スケジューリングでは，コンテキストスイッチは優先度の高いタスクがリリースされた場合かタスクの実行が終了した場合にのみ発生する．前者に対してはタイマを使ってスケジューラを呼び出すように実装し，後者に対してはタスクの実行が終了したらつねにスケジューラを呼び出すように実装した．スケジューラは各プロセッサに 1 つ用意されている．

各スケジューラは，周期タスク用のキューと非周期タスク用のキューを持っており，起動されるたびに非周期タスクキューを調べ，到着時刻を過ぎている非周期タスクがいれば仮想デッドラインを割り当てる．そして，必要であればテンポラルマイグレーションを発生させる．ここで，この実装方法は，すべての非周期タスクがあらかじめキューに格納されているという本評価用のためのものであることに注意されたい．実際には，非周期タスクは到着時に非周期タスクキューに格納され，割込みを用いてスケジューラを起動するか，次のスケジュール時刻まで待ってスケジュールされることになる．スケジューラは，非周期タスクキューに非周期タスクが存在すれば仮想デッドラインを割り当て，必要であればテンポラルマイグレーションを発生させることになる．

図 4 にスケジューラの疑似コードを示す．疑似コードは可読性を重視しており，実際のコーディングは図 4 の論理を変えずにできる限りの高速化を図っていることに注意されたい．スケジューラは，まず実行可能なタスクが格納されているレディキューをロックする (2 行目)．ロックを獲得できなかった場合にはビジーウェイトしてロックが解放されるのを待つことになる．ロックの対象がレディキューであるため，ほかのプロセッサのスケジューラ以外にはブロックされることはない．よって，デッドロックしない限りは大きな待ち時間にはならないと考えられる．次に，TBS アルゴリズムに従って前回のスケジュール時より後に到着した非周期タスクにデッドラインを割り当てる（3〜9 行目）．理論的には，非周期タスクが到着したら即座に TBS アルゴリズムを実行したほうが応答時間は最小化できるが，その分スケジューラ呼び出した多くなりオーバヘッドが大きくなってしまうため，スケジューラの呼び出しは周期タスクのリリース時か実行完了時に限定した．比較対象のすべてのアルゴリズムがこのような実装になっているため，相対的な性能は大きく変わらないと考えられる．テンポラルマイグレーションを用いない TBS は 10 行目を実行せずに 11 行目に進む．一方，TBS-TM-F/B/W は 10 行目でテンポラルマイグレーションを発生させる．

```
Let t be the current time.
Let Q_x be the ready queue of P_x.
Let β_x be a set of the aperiodic tasks on P_x.
 1.  function schedule do
 2.    lock Q_x.
 3.    for each α_{x,k} ∈ β_x do
 4.      if a_{x,k} ≤ t then
 5.        assign Equation (1) to α_{x,k}.
 6.        remove α_{x,k} from β_x.
 7.        insert α_{x,k} into Q_x.
 8.      end if
 9.    end for
10.    call temporal_migration.
11.    select task with earliest deadline in Q_x.
12.    unlock Q_x.
13.    if any task is selected then
14.      execute selected task.
15.    end if
16.  end when
```

図 4　スケジューラ関数
Fig. 4　Scheduler function.

図 5 にテンポラルマイグレーションの疑似コードを示す．基本的には，4.3 節で示した手順と同様であるが，7 行目でマイグレーション先のプロセッサを見つける処理は FF 手法，BF 手法，WF 手法でそれぞれ異なることに注意されたい．紙面の都合上，それぞれのアルゴリズム記述については省略する．また，マイグレーション先のプロセッサ P_y が見つかった場合に，P_y のレディキューをロックして，もう一度条件式 (2) が成り立っているかどうかを確認する．これは，マイグレーション先のプロセッサを検索中に P_y のタスク状態が変化している可能性があり，もし条件式 (2) が成り立たなくなっているとマイグレーションする周期タスクの実時間性が満たせなく恐れがあるので，その場合には P_y へのマイグレーショ

```
Let γ_x be a set of the arrived aperiodic tasks on P_x
1.  function temporal_migration do
2.    select periodic task τ_i with earliest deadline
      in Q_x, which has not been migrated yet.
3.    if there is no such τ_i then
4.      return.
5.    end if
6.    for each α_{x,k} ∈ γ_x do
7.      find destination processor P_y
8.      if P_y is found out then
9.        lock Q_y.
10.       if Condition (2) still holds then
11.         remove τ_i from Q_x.
12.         assign Equation (3) to τ_i.
13.         insert τ_i into Q_y.
14.         unlock Q_y
15.         invoke schedule on P_y.
16.         assign Equation (4) to α_{x,k}.
17.         return.
18.       else
19.         unlock Q_y
20.       end if
21.     end if
22.   end for
23. end function
```

図 5　テンポラルマイグレーション関数
Fig. 5　Temporal migration function.

ンは行わないことにする．FF 手法は最初に見つかったプロセッサを利用するため，7 行目から 11 行目に到達するまでの実行時間は比較的短いことが予想されるが，BF 手法と WF 手法はすべてのプロセッサを検索するので，11 行目に到達するまでの時間は長くなる可能性があるため，10 行目で再度条件式 (2) を確認する必要があると考えた．ここで，9 行目で O_y のロックを取得する際に，P_y 上のスケジューラがすでにテンポラルマイグレーションのために Q_x のロックを待っている状態の場合には P_x と P_y 間でデッドロックが発生してしまう．このようなデッドロックを簡単に回避するために，各プロセッサにロック待ち状態を示すフラグを用意し，P_y の検索はスケジューラがテンポラルマイグレーションのためのロックを待っていないプロセッサに限定した．

P_y へのマイグレーションは，τ_i を格納するレディキューを変更し，ロックを解放してからソフトウェア割込みを用いて P_y のスケジューラを明示的に呼び出すことで実現した（11〜15 行目）．ほかにもマイグレーション方法は考えられるが，シミュレーションに用いた RMT Processor は，3 章でも述べたようにハードウェアでコンテキストを管理できるため，上述した方法で簡単にマイグレーションを行うことができる．最後に，非周期タスクのデッドラインを更新して関数は終了する．ここで，到着した非周期タスクが複数あった場合には，理論的にはすべての非周期タスクに対してテンポラルマイグレーションを行ったほうが応答時間を短縮することができると考えられるが，1 つのプロセッサから同時に複数の周期タスクをテンポラルマイグレーションするとサーバ帯域幅の増加量のトラッキングが困難になってしまうため，1 回のスケジューラ起動に対してテンポラルマイグレーションは 1 回に限定した．このような制限を設けても，次節で示すように応答時間を劇的に短縮できることを確認した．

実装における問題としては，図 4 および図 5 中で，式 (1)，式 (2)，式 (3)，および式 (4) の計算にスケジューラが起動した時刻 t を使っていることがあげられる．理論的には，デッドラインを割り当てられた周期タスクおよび非周期タスクは，その時刻 t からスケジュールが再開されることを仮定しているが，実際にはスケジューラ関数やテンポラルマイグレーション関数のオーバヘッドが存在する．本論文におけるシミュレーションでは，周期タスクのシステム使用率が 60%なのでデッドラインミスは発生しなかったが，より高いシステム使用率の場合には計算にかかる時間を考慮に入れてそれらの式を計算する必要があると考えられる．このことは，テンポラルマイグレーションを用いない TBS に対しても同様であり，また本論文はアルゴリズムの設計に焦点を当てているので，オーバヘッドを考慮したシステムの全体設計に対する詳細な吟味は論文の範囲外とした．

5.3　応答時間に関する結果と考察

図 6，**図 7**，**図 8** は，平均到着率が $\mu = 0.1$ のときの TBS アルゴリズムに対する各アルゴリズムの平均応答時間の短縮倍率を示している．プロセッサ数が $M = 2$ の場合は，マイグレーション先のプロセッサはつねに隣のプロセッサになるのでテンポラルマイグレーション手法を用いる TBS アルゴリズムの性能はすべて同じであることに注意されたい．図 6 を見ると，負荷が低いうちはアルゴリズムの性能差は見られなかった．これは，負荷が低い場合にはマイグレーションを起こさなくても TBS アルゴリズムによって比較的早い仮想デッドラインを割り当てることができるためであると考えられる．しかしながら，負荷が高くなると，テンポラルマイグレーション手法を用いることで最終的に平均応答時間を最大で約 6 倍改善することができた．ここで，$U(\alpha)$ が 22〜27%の範囲では，テンポラルマイグレー

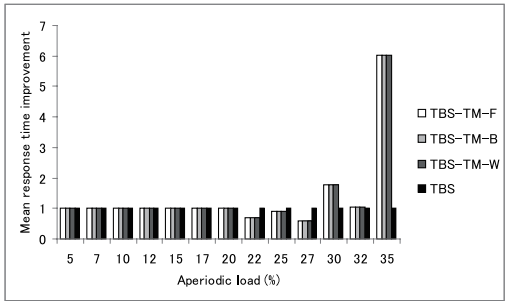

図 6　平均応答時間：$(m, \mu) = (2, 0.1)$
Fig. 6　Mean response time: $(m, \mu) = (2, 0.1)$.

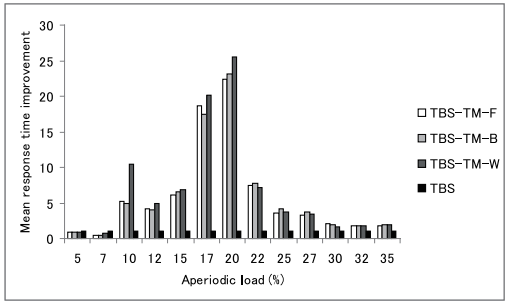

図 7　平均応答時間：$(m, \mu) = (4, 0.1)$
Fig. 7　Mean response time: $(m, \mu) = (4, 0.1)$.

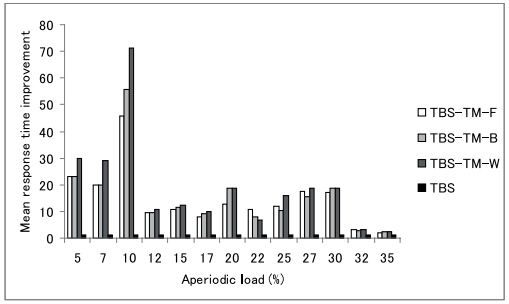

図 8　平均応答時間：$(m, \mu) = (8, 0.1)$
Fig. 8　Mean response time: $(m, \mu) = (8, 0.1)$.

ションを用いると性能が低下してしまったことに注意されたい．プロセッサ数および平均到着率が異なるほかのシミュレーションにおいても，テンポラルマイグレーションによって性能が低下しまう場合があることが確認された．このことについては，後ほど考察を行う．

プロセッサ数が $M = 4$ になると，各アルゴリズムの性能差には違いが見られた．図 7 を見ると，負荷が低いうちは性能差は小さかったが，非 $U(\alpha)$ が 10%を超えるとテンポラルマイグレーション手法を用いることで応答時間を劇的に短縮することができた．このことから，プロセッサ数が多い方がテンポラルマイグレーション手法によって非周期タスクにより早い仮想デッドラインを割り当てることが可能になり，負荷が低い場合でもその効果は大きかったことが分かる．また，マイグレーションのプロセッサ選択に WF 手法を用いた場合が最も応答時間を短縮できた．このことから，一時的に周期タスクをマイグレーションする場合は，最も余裕のあるプロセッサを選択するのが効率的であったことが分かる．TBS-TM-W アルゴリズムは $U(\alpha)$ が 20%のとき平均応答時間を最大で約 25 倍改善できた．$U(\alpha)$ が 22%を超えると，今度は性能改善の程度は徐々に小さくなっていった．これは，システム負荷が 100%に近づくとシステムが飽和状態となり，いくらマイグレーションを行っても性能改善には限界があるためであると思われる．最終的に非周期タスクの負荷が 35%になると，システム負荷は 95%となり，性能改善の大きさはプロセッサ数が $M = 2$ の場合とほとんど同じ結果となった．

プロセッサ数が $M = 8$ になると，テンポラルマイグレーション手法の効果はさらに大きくなった．マイグレーション先のプロセッサ候補が増えたことで，負荷が低い場合でもかなりの性能改善が確認できた．TBS-TM-W アルゴリズムは，$U(\alpha)$ が 10%のときに平均応答時間を最大で 70 倍短縮することができた．この結果から，プロセッサ数が多い方がテンポラルマイグレーション手法の効果は大きいことが分かる．

図 9，**図 10**，**図 11** は，平均到着率が $\mu = 0.2$ のときの TBS アルゴリズムに対する各アルゴリズムの平均応答時間の短縮倍率を示している．$\mu = 0.2$ であるため，非周期タスクは比較的短い間隔で到着することになる．先の $\mu = 0.1$ のときと同様に，テンポラルマイグレーション手法の効果が大きかったことが分かるが，全体的に TBS アルゴリズムに対する相対的な性能改善は $\mu = 0.1$ のときよりも小さくなっていた．これは，頻繁に非周期タスクが到着することで，各々の非周期タスクに対して短縮できる仮想デッドラインの長さが短くなり，結果として各々の非周期タスクの応答時間も長くなったためと思われる．しかしながら，TBS アルゴリズムに比べると平均応答時間を劇的に短縮できていたことからテンポラルマイグレーション手法自体の効果の大きさが確認できた．

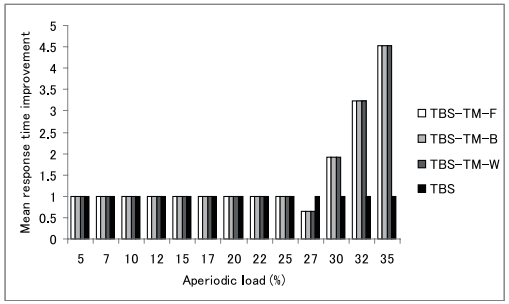

図 9　平均応答時間：$(m, \mu) = (2, 0.2)$
Fig. 9　Mean response time: $(m, \mu) = (2, 0.2)$.

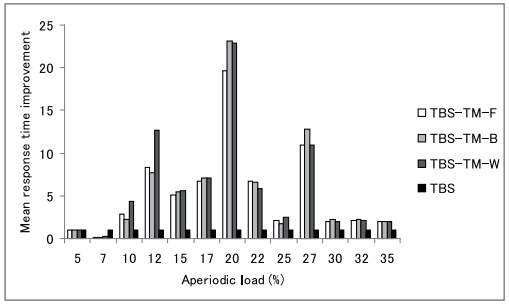

図 10　平均応答時間：$(m, \mu) = (4, 0.2)$
Fig. 10　Mean response time: $(m, \mu) = (4, 0.2)$.

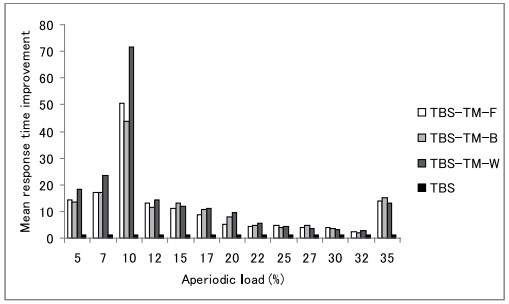

図 11　平均応答時間：$(m, \mu) = (8, 0.2)$
Fig. 11　Mean response time: $(m, \mu) = (8, 0.2)$.

ここで，テンポラルマイグレーションによって必ずしも応答時間が短縮できるわけではないことに注意されたい．図 6 の $\alpha = 22$〜27%，図 7 の $\alpha = 5$〜7%，図 9 の $\alpha = 27\%$，および図 10 の $\alpha = 7\%$の場合では，従来の TBS よりも TBS-TM-F/B/W のほうが平均応答時間が悪くなっていた．テンポラルマイグレーションは，現在到着している非周期タスクの応答時間を短縮するために周期タスクをほかのプロセッサにマイグレーションする手法である．そのため，周期タスクのマイグレーション先のプロセッサは一時的に負荷が増加することになる．一時的なプロセッサ負荷の増加が大きくなりすぎると，近い将来このプロセッサに到着する非周期タスクに対して，新たにテンポラルマイグレーションを行うことでそのときの状態に比べるとプロセッサ負荷を低減できたとしても，元々のプロセッサ負荷までは低減できない可能性も大いにありうる．この現象が多く発生すると，短縮できる応答時間よりも延長してしまう応答時間のほうが全体的に大きくなってしまい，結果として平均応答時間も低下してしまうことがあるのだと考えられる．

対応策としては，ある一定の応答時間短縮が得られる場合にのみテンポラルマイグレーションを発生するという方法があげられる．たとえば，式 (4) で得られる仮想デッドラインが式 (1) で得られる元の仮想デッドラインと大差がない場合，もしくは式 (4) で得られる仮想デッドラインよりも早いデッドラインを持った周期タスクが多く存在する場合には，テンポラルマイグレーションを発生させても応答時間の劇的な短縮は得られない．その一方で，マイグレーション先のプロセッサの負荷を一時的に大きくしてしまうので，そのプロセッサに関しては応答性を低下させてしまう要因となる．よって，全体的な応答性向上の確率が高く見込める場合にのみテンポラルマイグレーションを発生させるという手法が効果的であると考えられるが，この手法を用いても性能改善を保証できるわけではない．100%の性能改善を保証するためには，スケジューリングの先読みが必要になりオーバヘッドとのトレードオフとなる．本論文では，この点を今後の課題とする．

5.4　マイグレーション数に関する結果と考察

図 12，**図 13**，**図 14**，**図 15**，**図 16**，**図 17** は，非周期タスクの到着回数に対するマイグレーションの相対回数を示している．テンポラルマイグレーションが 1 回発生すると，周期タスクは一時的にほかのプロセッサに移動され，次のリリース時刻になると元のプロセッサに再び戻されるので，2 回のマイグレーションが発生することになる．評価結果には，両方のマイグレーション回数が含まれているので，テンポラルマイグレーション自体の発生回数は評価結果に示された回数の半分ということになる．

前節の応答時間に関する評価結果と照らし合わせると，応答時間が短縮できた非周期ワー

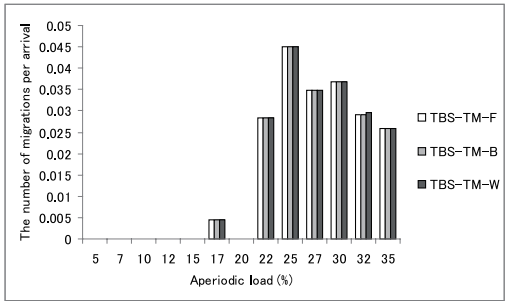

図 12　マイグレーション数：$(m, \mu) = (2, 0.1)$
Fig. 12　The number of migrations: $(m, \mu) = (2, 0.1)$.

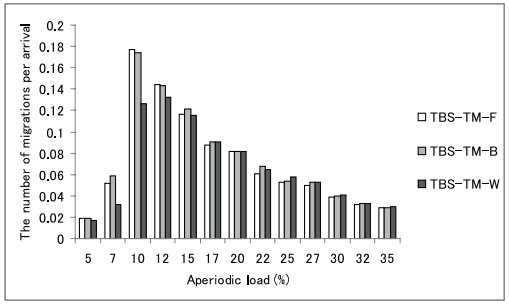

図 13　マイグレーション数：$(m, \mu) = (4, 0.1)$
Fig. 13　The number of migrations: $(m, \mu) = (4, 0.1)$.

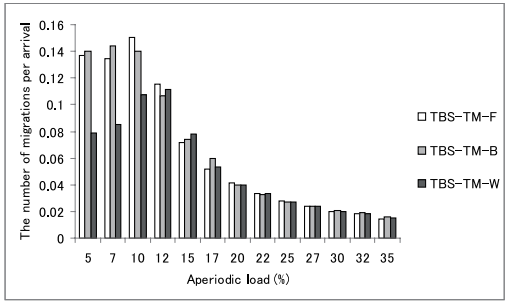

図 14　マイグレーション数：$(m, \mu) = (8, 0.1)$
Fig. 14　The number of migrations: $(m, \mu) = (8, 0.1)$.

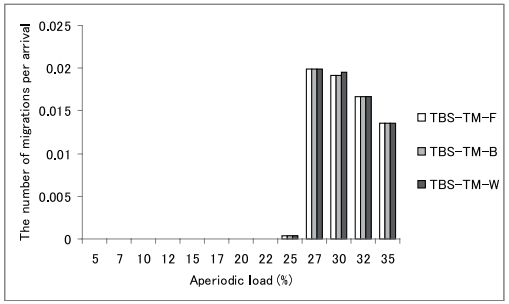

図 15　マイグレーション数：$(m, \mu) = (2, 0.2)$
Fig. 15　The number of migrations: $(m, \mu) = (2, 0.2)$.

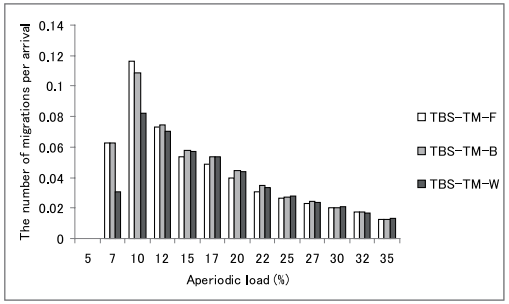

図 16　マイグレーション数：$(m, \mu) = (4, 0.2)$
Fig. 16　The number of migrations: $(m, \mu) = (4, 0.2)$.

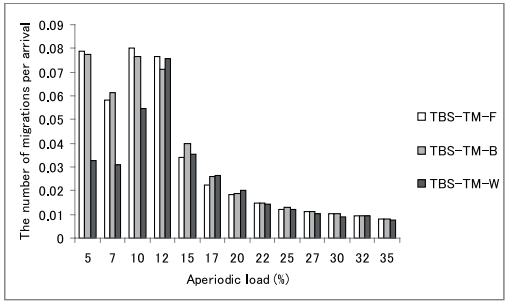

図 17　マイグレーション数：$(m, \mu) = (8, 0.2)$
Fig. 17　The number of migrations: $(m, \mu) = (8, 0.2)$.

クロードに対してはマイグレーションが発生していたことが分かる．また，1 回のテンポラルマイグレーションで短縮できる応答時間は，そのときの各プロセッサの負荷や周期タスクの残り実行時間，到着した非周期タスクの実行時間などに依存するので，相対マイグレーション数が大きいからといって必ずしも応答時間が大きく短縮できたわけではなかったことが分かる．相対マイグレーション数は最大でも 0.2 回以下であった．すなわち，平均的には 10 個の非周期タスクが到着する間に多くて 1 回のマイグレーションが発生したということになる．前節で示した応答時間短縮の大きさを考慮すると，最大で 0.2 回の相対マイグレーション数は許容範囲であると考えられる．しかしながら，前節でも述べたように，マイグレーションを発生させたにもかかわらず応答時間が悪くなる場合も確認されたので，マイグレーション方針に関してはさらなる吟味が必要である．

非周期ワークロードが比較的低い場合には，TBS-TM-F と TBS-TM-B よりも TBS-TM-W のほうがマイグレーション数が少ない場合が多かった．これは，WF 手法でマイグレーション先のプロセッサを選択したほうが各プロセッサの負荷が分散されるためであると考えられる．前節の評価結果より，非周期ワークロードが低い場合には，応答時間に関しても TBS-TM-W のほうが高い性能を発揮していたことが確認できた．よって，非周期ワークロードが低い場合には，各プロセッサの負荷を分散できる WF 手法を用いてマイグレーション先を選択したほうが，応答時間を短縮でき，かつマイグレーション回数も抑制できるため，効率が良いといえる．

6. 結　　論

本論文では，マルチコアプロセッサを対象として，周期タスクの時間制約を破綻させることなく，非周期タスクの応答時間を劇的に短縮することができるテンポラルマイグレーション手法を提案し，従来の TBS アルゴリズムと組み合わせた TBS-TM アルゴリズムを設計した．また，さらなる応答性の向上を目的として，マイグレーション先のプロセッサを選択するための 3 つの発見的手法（FF 手法，BF 手法，WF 手法）を導入した．命令レベルシミュレーションによる評価では，WF 手法を採用した TBS-TM アルゴリズムが最も性能が良く，平均応答時間を最大で約 70 倍短縮できたことを示した．このことから，マルチコアプロセッサでリアルタイム性を保証して非周期タスクを扱う場合に，テンポラルマイグレーション手法が非常に効果的であることが証明できた．

今後は，まずマイグレーション時のプロセッサ選択手法を考える必要がある．本論文では，3 つの簡単な発見的手法を用いたが，より洗練された手法を導入することで非周期タスクへの応答性をさらに改善できると考えられる．しかしながら，実装や計算も複雑になることが予想され，性能と実用性のトレードオフを考慮する必要があると考えられる．また，本論文で提案したマイグレーション方針では，5.3 節でも述べたように応答性を低下させてしまう場合も存在する．よって，今後はより高い確率で応答性向上を得られるマイグレーション方針を考えていく予定である．さらに，本論文ではすべての周期タスクはプリエンプティブであることを仮定した．そのため，I/O 処理を行う制御タスクなどが実行されている場合には，即座にプリエンプションしてマイグレーションをすることはできない．今後は，クリティカルセクションが存在する場合のマイグレーション方針も考えていく予定である．

謝辞　本研究は，JST CREST の支援による．また，本研究の一部は，日本学術振興会の支援による．

参　考　文　献

1) Andersson, B., Abdelzaher, T. and Jonsson, J.: Global Priority-Driven Aperiodic Scheduling on Multiprocessors, *Proc. IEEE International Parallel and Distributed Processing Symposium* (2003).
2) Andersson, B., Abdelzaher, T. and Jonsson, J.: Partitioned Aperiodic Scheduling on Multiprocessors, *Proc. IEEE International Parallel and Distributed Processing Symposium* (2003).
3) Andersson, B. and Jonsson, J.: Fixed-Priority Preemptive Multiprocessor Scheduling: To Partition or not to Partition, *Proc. International Conference on Real-Time Systems and Applications*, pp.337–346 (2000).
4) Baker, T.: Stack-Based Scheduling of Real-Time Processes, *Real-Time Systems*, Vol.3, No.1, pp.67–99 (1991).
5) Baruah, S. and Lipari, G.: A Multiprocessor Implementation of the Total Bandwidth Server, *Proc. IEEE International Parallel and Distributed Processing Symposium*, pp.40–47 (2004).
6) Davis, R., Tindell, K. and Burns, A.: Scheduling Slack Time in Fixed Priority Preemptive Systems, *Proc. IEEE Real-Time Systems Symposium*, pp.222–231 (1993).
7) Lehoczky, J., Sha, L. and Strosnider, J.: Enhanced Aperiodic Responsiveness in Hard Real-Time Environments, *Proc. IEEE Real-Time Systems Symposium*, pp.261–270 (1987).
8) Liu, C. and Layland, J.: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *J. ACM*, Vol.20, No.1, pp.46–61 (1973).
9) Lopez, J., Carcia, M., Diaz, J. and Carcia, D.F.: Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems, *Proc. Euromicro Con-*

ference on Real-Time Systems, pp.25–33 (2000).
10) Sha, L., Rajkumar, R. and Lehoczky, J.: Priority Inheritance Protocols: An Approach to Real-Time Synchronization, *IEEE Trans. Comput.*, Vol.39, No.9, pp.1175–1185 (1990).
11) Spuri, M. and Buttazo, G.: Efficient Aperiodic Service under Earliest Deadline Scheduling, *Proc. IEEE Real-Time Systems Symposium*, pp.2–11 (1994).
12) Spuri, M. and Buttazo, G.: Scheduling Aperiodic Tasks in Dynamic Priority Systems, *Real-Time Systems*, Vol.10, No.2, pp.179–210 (1996).
13) Stohr, J., Bulow, A. and Farber, G.: Bounding Worst-Case Access Times in Modern Multiprocessor Systems, *Proc. Euromicro Conference on Real-Time Systems*, pp.189–198 (2005).
14) Taira, T., Kamata, N. and Yamasaki, N.: Design and Implementation of Reconfigurable Modular Robot Architecture, *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.3566–3571 (2005).
15) Tullsen, D., Eggers, S. and Levy, H.: Simultaneous Multithreading: Maximizing On-Chip Parallelism, *Proc. Annual International Symposium on Computer Architecture*, pp.392–403 (1995).
16) Yamasaki, N.: Responsive Multithreaded Processor for Distributed Real-Time Systems, *Journal of Robotics and Mechatronics*, Vol.17, No.2, pp.130–141 (2005).

(平成 20 年 1 月 29 日受付)
(平成 20 年 5 月 20 日採録)

加藤　真平（正会員）

1982 年生．2004 年慶應義塾大学理工学部情報工学科卒業．2006 年同大学大学院理工学研究科開放環境科学専攻修士課程修了．現在同大学訪問研究員．リアルタイムシステム，オペレーティングシステム等の研究に従事．

山崎　信行（正会員）

1966 年生．1991 年慶應義塾大学理工学部物理学科卒業．1996 年同大学大学院理工学研究科計算機科学専攻博士課程修了．博士（工学）．同年電子技術総合研究所入所．1998 年 10 月慶應義塾大学理工学部情報工学科助手．同専任講師を経て 2004 年 4 月より同助教授．現在産業技術総合研究所特別研究員を兼務．並列分散処理，リアルタイムシステム，システム LSI，ロボティクス等の研究に従事．