社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－SLDM－124 (5)
2006／3／16

RMT Processor上のスキップ可能なタスクを扱うリアルタイムスケジューリング機構

船岡 健司† 加藤 真平†† 小林 秀典†† 山崎 信行†,††

† 慶應義塾大学 理工学部 情報工学科
†† 慶應義塾大学大学院 理工学研究科 開放環境科学専攻 コンピュータ科学専修
〒223-8522 横浜市港北区日吉3-14-1
E-mail: †{funaoka,shinpei,kobayashi,yamasaki}@ny.ics.keio.ac.jp

あらまし 本論文では，オープンリアルタイムシステムにおける，スキップ可能な周期ファームタスクのスケジューリングについて述べる．従来のSkip-Overモデルより時間制約の緩いタスクを扱えるよう，Skip-Overモデルを拡張する．また，タスクのスキップを制御することにより，従来のシステムより柔軟に時間制約を取り扱うことが可能なスケジューラの設計と実装を行なう．その際，RMT Processor上でスレッド間の優先度を考慮することにより，重要なタスクのデッドラインミスを低減させる手法を提案する．シミュレーションの結果，スレッド間の優先度を考慮することにより，重要なタスクのデッドラインミス率を最大で約22.2%減少させられることを示した．
キーワード リアルタイムシステム，スケジューリングアルゴリズム，Skip-Overモデル，RMT Processor

A Real-Time Scheduling Mechanism that Exploits Skips on RMT Processors

Kenji FUNAOKA†, Shinpei KATO††, Hidenori KOBAYASHI††, and Nobuyuki YAMASAKI†,††

† Department of Information and Computer Science, Faculty of Science and Technology, Keio University
†† Department of Computer Science, Graduate School of Science and Technology, Keio University
3-14-1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa 223-8522, JAPAN
E-mail: †{funaoka,shinpei,kobayashi,yamasaki}@ny.ics.keio.ac.jp

Abstract In this paper, we deal with the problem of scheduling task sets, consisting of "occasionally skippable" firm periodic tasks on open real-time systems. We extend the Skip-Over model to deal with tasks that have generous time constraints. We design and implement a scheduler that exploits skips to manage flexible time constraints. We propose a hardware priority assignment technique to reduce missing deadlines of important tasks on RMT Processors. Simulation results show that the deadline miss ratio decreases to about 22.2% in the maximum by considering priorities among threads when the system falls into overloaded conditions.
Key words real-time systems, scheduling algorithms, Skip-Over model, RMT Processor

1. 序 論

従来のリアルタイムシステムに対する研究は，デッドラインミスを許容できるかできないかという，非常に大雑把なモデルに基づいていた．リアルタイムシステムの多様化により，より柔軟なモデルの必要性が増している．ビデオアプリケーションの例を挙げると，表示領域を60Hzで更新することにより滑らかに見え，システムが過負荷状態となり，30Hzを下まわると違和感を覚えるようになるなど，システム開発者の目標となる数値がシステムの状態によって複数存在する．過負荷状態では，全てのデッドラインを必ずしも守る必要はないが，ある一定の割合を超えるとシステムの価値が著しく低下する場合がある．また，システムの時間制約はデッドラインミス率のみで表すことはできず，連続したデッドラインミスを起こしてしまうと，システムの価値が失われるシステムも存在する．しかし，従来のタスクモデルでは，デッドラインミスについて柔軟に扱うことができない．

コンピュータが小型化し，より身近なシステムに組み込まれるようになるにつれ，性能は高く，コストは低いシステムを構築することが求められている．Simultaneous Multithreading

(SMT) [1] は，細粒度のスーパースカラとマルチスレッディングを組み合わせたプロセッサアーキテクチャである．8 スレッド SMT において，従来のスーパースカラの約 2〜3 倍のスループットを達成した [2]．プロセッサの資源を共有して実行することにより，プロセッサの資源を効率良く使用することが可能となり，コストパフォーマンスに優れている．複数あるスレッドのうち，どのスレッドの命令を実行するかは，フェッチポリシによって決定され，フェッチポリシ次第で全体のスループットが大きく変化する [3]．また，資源を共有して実行している為，同時に実行するタスクによって，タスクの実行時間が変動する．リアルタイムシステムでは，スループットだけでなく時間予測性も重要である為，SMT を考慮したスケジューリングを行なうことにより，システムの価値を高められる可能性がある [4], [5].

本論文の構成は，以下の通りである．2. では，時間制約を柔軟に表現可能なタスクモデルを紹介した後に，本スケジューリング機構の前提となる RMT Processor の特長について述べる．3. では，Skip-Over モデルについて，時間制約の緩いタスクでも扱えるように拡張する．また，過負荷状態時にシステムの価値が著しく低下することを防ぐスケジューラの設計と実装を述べる．4. では，本スケジューリング機構の評価を示す．最後に，5. で結論と今後の課題を述べる．

2. 背　　景

2.1 スキップを想定したタスクモデル

デッドラインまでにタスクの実行を完了できないことはスキップと呼ばれ，スキップからタスクの時間制約を表現し，柔軟にスキップを扱うタスクモデルとアルゴリズムが提案されている．Hamdaoui と Ramanathan [6] は，(m,k) ファームモデルを提案した．しかし，(m,k) ファームモデルを対象とした DBP アルゴリズムは，ヒューリスティックな手法に基づいており，十分なスケーラビリティに関する分析は行なわれていない．

Koren と Shasha [7] は，Skip-Over モデルを提案した．Skip-Over モデルは，タスク τ_i のスキップ可能な最小間隔をスキップパラメータ $s_i(\geqq 2)$ で表すことにより，タスクの時間制約をより柔軟に表すことが可能なタスクモデルである．$s_i = \infty$ のタスクは，スキップすることができない．

タスクは，レッドとブルーの状態を遷移し，時間制約を守ることを目標とする．レッドタスクは，デッドラインまでに実行を完了させなくてはならない．ブルータスクは，いつでもスキップ可能である．$s_i = 3$ のタスクの場合，ブルータスクの状態でデッドラインミスを起こすと，レッドタスクへ状態遷移する．また，レッドタスクの状態の時，$s_i - 1 = 2$ 回連続してデッドラインを満たすと，ブルータスクへ状態遷移する．これを繰りかえすことにより，システムの時間制約を守る．

タスクをスケジュールする際，レッドタスクとブルータスクの扱いは，Skip-Over アルゴリズムに依存する．基本的なアルゴリズムとして Red Tasks Only (RTO) と Blue When Possible (BWP) が提案されている [7]．RTO は，レッドタスクのみをスケジュールする Skip-Over アルゴリズムである．全てのタスクのブルータスクが，s_i 番目に初めて出現するシステムを deeply red モデルと呼び，RTO は deeply red モデルにおいて最適である．BWP は，実行可能なレッドタスクが存在しない場合にのみ，ブルータスクを実行する Skip-Over アルゴリズムである．BWP は，RTO でスケジュールできないタスクセットをスケジュール可能な場合があるという観点から，RTO よりも優れている．また，Marchand と Silly-Chetto [8] は，レッドタスクの実行を可能な限り遅らせて空き時間を計算することにより，ブルータスクを実行する機会を増やす Red tasks as Late as Possible (RLP) を提案した．

スキップ可能なタスクを扱う際，最初のブルータスクがいつ現れるかということが問題となる．適切にブルータスクの出現位置を選択することで，より多くのタスクを実行することが可能となる．しかし，この問題を箱詰問題に還元することにより，実行可能判定は NP 困難であることが示されている [7]．よって，確実に実行可能判定を行なうには，判定の基準を緩めるか，NP 困難問題を解かなければならない．

現在までに提案されているスキップを想定したタスクモデルは，従来通りの利用率から実行可能か判定することが困難であり，判定に十分な時間を割けないシステムにとっては問題となる．Skip-Over モデルでは，実行可能である為の十分条件も示されているが，利用率からの判定よりも非常に多くのコストがかかり，システムによっては利用することができない．もし，利用率から判定を行なうとすると，スキップを想定しない従来のタスクモデルと同じ利用率までしか保証できない．よって，過負荷により時間制約を満たせない可能性がある場合，それを少しでも回避することが求められる．

2.2 Responsive Multitheraded Processor

Responsive Multithreaded Processor (RMT Processor) [9] は，SMT へスレッド間の優先度制御機構を加えることにより，ハードウェアレベルで優先度を制御可能なリアルタイム処理用プロセッサである．RMT Processor でタスクを実行する様子を図 1 に示す．スレッドに与えられた優先度に基づいてフェッチを行なうことにより，優先度の高いスレッドに多くの資源を割り当てる．これにより，最高優先度スレッドの実行時間の変動を抑制することが可能である．優先度をつけることにより，マルチスレッディングによるレイテンシの隠蔽の効果は薄れ，全体のスループットは低下する．しかし，時間予測性が重要なリアルタイムシステムにおいて，実行時間の変動を抑制することは非常に重要である．

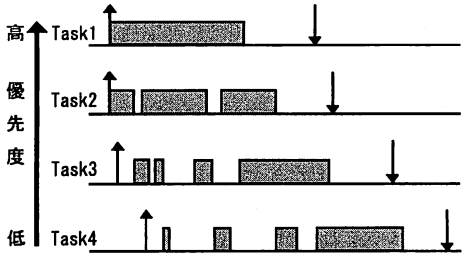

図 1　RMT Processor でタスクを実行する様子

RMT Processor の命令発行ユニットの概要を表 1 に示す．RMT Processor には，タスクを実行可能なアクティブスレッドと，ハードウェアでコンテキストを保持することが可能なキャッシュスレッドが存在する．タスク数がアクティブスレッドとキャッシュスレッドに収まりきる 40 までならば，ハードウェアでコンテキストの情報を保持することにより，高速にコンテキストスイッチを行なうことが可能である．アクティブスレッドとキャッシュスレッドをスワップする swapth 命令により，4 クロックでコンテキストスイッチを実現することが可能である．

表 1　RMT Processor の命令発行ユニットの概要

アクティブスレッド数	8
キャッシュスレッド数	32
優先度の指定	256 段階
命令フェッチ数	8
同時命令発行数	4
同時命令完了数	4
整数レジスタ数	32bit × 32entry × 8set
整数リネームレジスタ数	32bit × 64entry
浮動小数点レジスタ数	64bit × 8entry × 8set
浮動小数点リネームレジスタ数	64bit × 64entry

3. 設計と実装

3.1 Skip-Over モデルの拡張

従来の Skip-Over モデルは，許容可能なスキップの最小間隔からスキップパラメータ $s_i(\geq 2)$ を定義した．スキップ可能な最大の割合は $s_i = 2$ の場合であり，最大で 50%までのスキップを認めるタスクしか扱うことができない．すなわち，連続したデッドラインミスを許容可能なタスクを扱うことができない．本節では，より時間制約の緩いタスクでも取り扱えるように，Skip-Over モデルを拡張する．

[定義 1]（拡張 Skip-Over モデル） タスク τ_i のスキップパラメータを $s_i(\geq 1)$ とする．レッドタスクは連続して $\lceil s_i - 1 \rceil$ 回成功するとブルータスクへ，ブルータスクは連続して $\left\lceil \frac{1}{s_i - 1} \right\rceil$ 回失敗するとレッドタスクへ状態遷移する．但し，$s_i = 1$ のタスクは，常にブルータスクである．□

従来の Skip-Over モデルでは，ブルータスクがスキップされると，常にレッドタスクへと状態遷移していた．これは，スキップの最小間隔からスキップパラメータを定義していた為である．スキップパラメータを成功の回数から定義することにより，ブルータスクが複数回デッドラインミスした時に，レッドタスクへと状態遷移するタスクを表現可能である．これにより，スキップの割合が 50%を超える場合も認めるタスクを扱えるようになる．$s_i \geq 2$ の場合，従来の Skip-Over モデルと同様である．拡張した Skip-Over モデルの例を以下に示す．

[例 1] $s_i = 3$ のタスクは，レッドタスクが連続して 2 回成功するとブルータスクへ，ブルータスクが連続して 1 回失敗するとレッドタスクへ状態遷移する．

[例 2] $s_i = \frac{4}{3}$ のタスクは，レッドタスクが連続して 1 回成功するとブルータスクへ，ブルータスクが連続して 3 回失敗するとレッドタスクへ状態遷移する．

[例 3] $s_i = 1$ のタスクは，常にブルータスクである．

[例 4] $s_i = \infty$ のタスクは，常にレッドタスクである．

3.2 スケジューラ

本研究では，複数コンテキストにおけるスキップ可能な周期ファームタスクを扱う．全てのタスクは，いつでも横取・中断可能であり，タスク間の相互作用はないと仮定する．$\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ をタスクセットし，それぞれのタスク $\tau_i = (C_i, P_i, s_i)$ は，RMT Processor で同じ優先度を設定した際の 8 スレッド同時実行時の最悪実行時間 C_i，最小周期 P_i，スキップパラメータ s_i で特徴付けられる．タスク τ_i によるプロセッサ利用率を $U_i \stackrel{\text{def}}{=} C_i/P_i$ と定義する．タスクはコンテキスト $X = \{X_1, X_2, \ldots, X_m\}$ に振り分けられ，それぞれのコンテキスト $X_k = \{\tau_a, \tau_b, \ldots\}$ において独立してスケジュールされる．コンテキスト X_k の利用率を $U(X_k) \stackrel{\text{def}}{=} \sum_{\tau_i \in X_k} U_i$ と定義する．

設計したスケジューラの概念図を図 2 に示す．複数コンテキストにおけるスケジューリング手法として，パーティショニングとグローバルスケジューリングが考えられる．パーティショニングは，シングルプロセッサ用のアルゴリズムをそのまま適用可能である為，本スケジューラはパーティショニングで行ない，各スレッドで独立してスケジュールする．各スレッドに余裕を持たせる Worst-Fit (WF) によって各スレッドに割り振る．WF のアルゴリズムをアルゴリズム 1 に示す．Skip-Over モデルの場合，タスクが実行可能とは，全てのレッドタスクが実行可能ということを表す．この判定には，従来の判定方法をそのまま用いることはできない．利用率からの条件より，Skip-Over モデルにおけるスケジュール可能条件である為の必要条件である式 (1) が導かれる．

$$\sum_{i=1}^{n} \frac{C_i(s_i - 1)}{P_i s_i} \leqq 1 \qquad (1)$$

実行可能判定を十分に行なうことができないオープンシステムを想定している為，式 (1) を用いて判定を行なう．それぞれのコンテキストでは，全てのプロセッサ時間を利用可能な Earliest Deadline First (EDF) [10] と，実装が単純でコストの少ない Blue When Possible (BWP) を用いた．これらを組み合わせ，EDF-BWP-WF によるスケジュールを行なう．

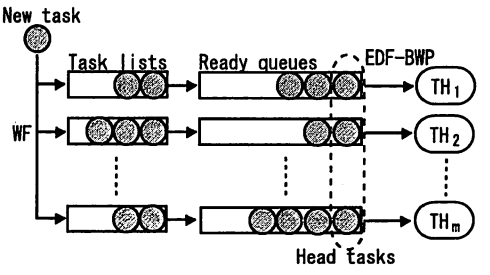

図 2　スケジューラ概念図

新しくタスクが到着すると，WF に基づいてタスクリストへ挿入する．タスクが実行可能状態となると，タスクをレディ

キューに移し，実行するまで待機させる．レディキューには，EDF-BWPに基づいた順序でタスクを整列させる．即ち，レッドタスクでデッドラインの早いタスクから遅いタスク，ブルータスクでデッドラインの早いタスクから遅いタスクの順序となる．レディキューの先頭のタスクをヘッドタスクと呼び，次に実行するタスクの候補である．タスクの実行が終了するとタスクリストに戻し，次に実行可能状態になるまで待機させる．これを繰り返すことにより，タスクのスケジュールを行なう．

アルゴリズム 1 Worst Fit

Require: τ_i is a new task;
1: $c \Leftarrow 1$;
2: **for** $k = 2$ to m **do**
3: **if** $U(X_k) < U(X_c)$ **then**
4: $c \Leftarrow k$;
5: **end if**
6: **end for**
7: **if** $\frac{C_i(s_i-1)}{P_i s_i} + \sum_{\tau_j \in X_c} \frac{C_j(s_j-1)}{P_j s_j} \leqq 1$ **then**
8: insert τ_i to X_c;
9: **else**
10: reject τ_i;
11: **end if**

SMTでは，資源を共有してタスクを実行する為，レッドタスクがデッドラインミスを起こす可能性がある場合でも，フェッチポリシによっては，ブルータスクがレッドタスクよりも優先して実行される可能性がある．よって，過負荷状態でレッドタスクの実行を保証できない時，RMT Processorの優先度機構を用いて，レッドタスクを実行中のスレッドの優先度を上げることにより，レッドタスクを優先して実行する．SMT Processorにおいても，ブルータスクの実行を止めることにより，レッドタスクを優先して実行することが可能である．しかし，使用可能な命令演算ユニットがある場合でも，一切使用することが不可能となってしまう．よって，RMT Processorの優先度付きSMTを使用することにより，プロセッサ資源を有効に使用することが可能となる．優先度を決定するアルゴリズムThread Priorityをアルゴリズム2に示す．パーティショニングによって，各スレッドで独立してスケジュールしている為，レッドタスクが他のレッドタスクの実行を妨害しないよう，2段階の優先度を用いた．本機構では，レッドタスクが実行できない場合の判定として，EDFでスケジュール不可能な利用率が1を超えたコンテキストが存在するかどうかで判定した．これは非常に緩い条件であり，本来は優先度を考慮しなくても実行できる可能性がある．更に，優先度を考慮することにより，マルチスレッディングによるレイテンシ隠蔽の効果が薄れ，全体のスループットが低下し，ブルータスクの実行される機会が減る．この判定を，コストはかかるがより厳密な手法に変更することにより，ブルータスクが実行される機会を増やすことが可能である．よって，タスクの到着頻度と判定のコストのトレードオフにより，判定方法を決定する必要がある．

タスクのスケジュールは，タスクの終了時と，タイマ割り込みにより100μ秒毎に関数scheduleを呼び出すことにより実現した．関数scheduleの概要を関数3に示す．関数scheduleは，特定のスレッドが全てのスレッドの管理を一元的に行なうのではなく，各スレッドにおいて独立して呼び出される．前述の優先度機構により，ブルータスクが実行中に割り込みが起こった時，関数scheduleにおいて優先度が低いまま処理され，レッドタスクの実行開始が大きく遅れてしまう可能性がある．よって，一時的にスレッドの優先度を上げることにより，この問題を回避する．次に，タスクリスト中のタスクを確認し，実行可能状態となったタスクをレディーキューに加える．そして，コンテキストをスワップさせるか判断する．コンテキストをスワップするのは，以下の条件のうち1つでも満たした場合である．1つ目は，BWPに基づき，ブルータスク実行中にレッドタスクが実行可能状態になった場合である．2つ目は，実行中のタスクの状態とヘッドタスクの状態が等しく，ヘッドタスクの優先度の方が高い場合である．最後に，スレッドの優先度を適切な値に再設定し，タスクの実行に戻る．これを繰り返すことにより，タスクの実行を切り替えていく．

アルゴリズム 2 Thread Priority

1: **if** $^{\exists}X_k \in X : U(X_k) > 1$ && τ_i is red **then**
2: thread priority $\Leftarrow$ PRIORITY_HIGH;
3: **else**
4: thread priority $\Leftarrow$ PRIORITY_NORMAL;
5: **end if**

関数 3 schedule

```
1: thread priority ⇐ PRIORITY_HIGH;
2: check periodic tasks;
3: if ready queue is not empty then
4:    if current task is blue && head task is red then
5:       abort current task;
6:    else if current task is red && head task is blue then
7:       return;
8:    else if current task priority ≧ head task priority then
9:       return;
10:   end if
11:   swap contexts;
12: end if
13: restore thread priority;
```

4. 評　価

4.1 評価環境

評価は，RMT ProcessorのRegister Transfer Levelシミュレーションで行なった．8スレッド全てを有効とし，動作周波数は250MHzと仮定して，クロックをカウントするレジスタから各時刻を算出した．実行するプログラムの生成には，gccのクロス開発環境を使用した．gccのバージョンは，3.4.3である．

4.2 評価方法

評価に使用したタスクの生成方法について述べる．タスクの数は，40までとする．タスクの最悪実行時間は，RMT Processor

において，同じ優先度で 8 スレッド同時実行時の値を用いる．スキップパラメータは 2，タスクの状態はブルータスク，周期は 1ms 以下の値から一様分布となるよう選択する．タスクによる利用率が $U_i \leqq 0$ もしくは $U_i > 1$ となった場合，そのタスクを破棄し，タスクを再度生成する．生成したタスクは WF でコンテキストに割り振り，タスクが破棄された場合，タスクを再度生成する．利用率の和が目標値 −0.1 から目標値の間になった段階でタスクの生成を終了し，これをタスクセットとする．

評価は，上記の方法で生成した各利用率のタスクセットを 10ms 実行し，全てのタスク，レッドタスク，ブルータスクそれぞれにつき，デッドラインミスを起こしたタスクの数からデッドラインミス率を算出した．比較対象としたアルゴリズムは，タスクの重要度もスレッド間の優先度も考慮しない EDF-WF，タスクの重要度は考慮するがスレッド間の優先度は考慮しない EDF-BWP-WF，タスクの重要度もスレッド間の優先度も考慮する EDF-BWP-WF-TP である．

4.3 評価結果

本節では，全てのタスク，レッドタスク，ブルータスクのデッドラインミス率をそれぞれ図 3，図 4，図 5 に示す．図の横軸は負荷を示し，負荷 100%とは，全タスクがデッドラインミスせずに EDF-WF でスケジュール可能である理論値の上限である．縦軸はデッドラインミス率を示す．

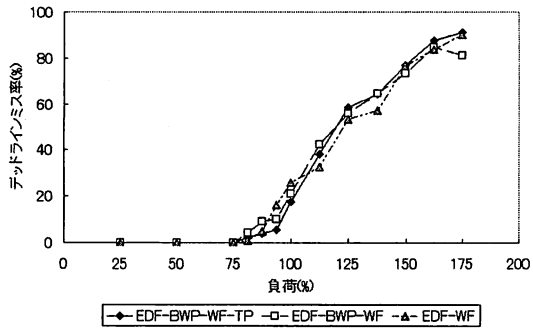

図 3 全タスクのデッドラインミス率

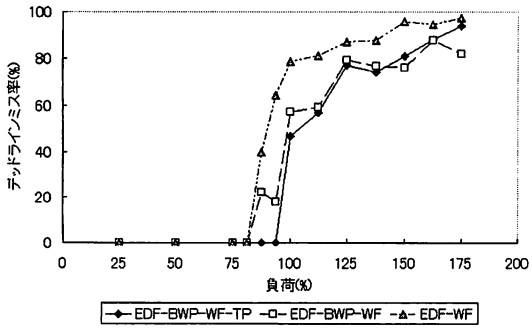

図 4 レッドタスクのデッドラインミス率

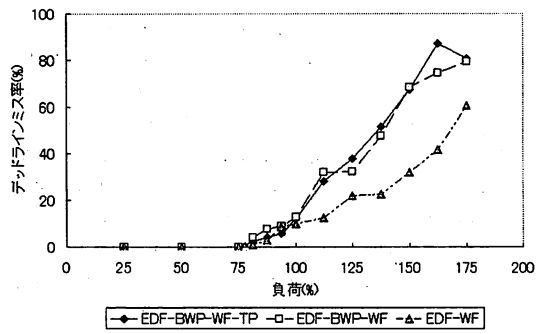

図 5 ブルータスクのデッドラインミス率

図 3 を見る限りでは，どのアルゴリズムも似た傾向を示しており，負荷 81.25%以上でデッドラインミスを起こしている．また，負荷に比例してデッドラインミス率が増加している．アルゴリズムの違いによる部分的な差異は確認できるが，負荷によって優劣が大きく変化している為，全体として特定の傾向は見受けられない．図 4 と図 5 では，アルゴリズムによって差が現れている．EDF-WF は，レッドタスクのデッドラインミス率が他のアルゴリズムよりも高い代わりに，ブルータスクのデッドラインミス率が他のアルゴリズムよりも低い．この傾向は，負荷 81.25%以上で常に成立している．EDF-BWP-WF-TP と EDF-BWP-WF について，レッドタスクのデッドラインミス率を比較すると，負荷 87.5%以上 137.5%以下では，EDF-BWP-WF-TP が EDF-BWP-WF よりデッドラインミスが低い．特に，EDF-BWP-WF-TP は，負荷 81.25%までデッドラインミスを起こさずに実行できているのに対し，EDF-BWP-WF-TP は，負荷 93.75%まで実行できている．EDF-BWP-WF-TP は EDF-BWP-WF と比較して，最大で負荷 87.5%においてデッドラインミス率が約 22.2%低い．負荷 150%以上では，デッドラインミス率が逆転している部分も見受けられる．過負荷状態に陥った時の全体的な傾向として，負荷が軽い場合に EDF-BWP-WF-TP が優れており，負荷が重くなっていくにつれて優劣の差がなくなっている．

4.4 考　　察

負荷が増加する際に，デッドラインミスが減少している部分が存在する．これは，タスクの生成とパーティショニングによって，スレッドの負荷の状況が変化する為であると考えられる．よって，各アルゴリズムのデッドラインミス率の絶対的な値ではなく，アルゴリズム間の相対的な差に注目する必要がある．

全タスクの場合，アルゴリズムによる特定の傾向は認められなかったが，レッドタスクとブルータスクを個別に見た場合，アルゴリズムによる差異を確認できた．Skip-Over モデルでは，システムの時間制約を守る為，レッドタスクの実行をデッドラインまでに完了させなくてはならない．レッドタスクがデッドラインミスを起こしているということは，システムの時間制約を守れていないということを意味しており，レッドタスクのデッドラインミス率を低く抑えることが重要である．

EDF-WF は，レッドタスクであることを一切考慮していない為，他のアルゴリズムよりレッドタスクのデッドラインミス率が高く，ブルータスクのデッドラインミス率が低い．ブルータスクを多く実行したとしても，レッドタスクのデッドラインミス率が高いということは，システムの価値が著しく低下していることを意味する．EDF-BWP-WF と EDF-BWP-WF-TP は，レッドタスクかどうか考慮してスケジュールしている為，EDF-WF よりレッドタスクのデッドラインミス率が低く，ブルータスクのデッドラインミス率が高い．ブルータスクは，デッドラインミスが許容されるタスクである為，システムの価値は EDF-WF の場合より低下していない．よって，レッドタスクかブルータスクかを考慮してスケジュールすることにより，システムの価値を高めることが可能である．

EDF-BWP-WF-TP と EDF-BWP-WF を比較すると，負荷が 137.5%以下において，EDF-BWP-WF-TP のレッドタスクのデッドラインミス率が低い．これは，スレッド間の優先度を考慮することにより，レッドタスクが優先的に実行される為である．逆に，EDF-BWP-WF は，スレッド間の優先度を考慮していない為，レッドタスクとブルータスクがフェッチポリシに基づいて実行され，レッドタスクのデッドラインミス率が高い結果となった．負荷が 112.5%以上の場合，EDF-BWP-WF-TP と EDF-BWP-WF の差は，非常に小さいか逆転している部分も見受けられる．この場合，どのスレッドも過負荷状態となり，レッドタスクに変化しやすい為，優先度機構が有効に働かなくなったと考えられる．また，レッドタスクを優先して実行することにより，ブルータスクがレッドタスクに状態遷移しやすくなり，過負荷状態を悪化させていると考えられる．よって，現在の負荷状況を判断して，本優先度機構を使うか切り替えることにより，システムの価値を高めることが可能である．

4.5 優先度決定機構のオーバヘッド

本優先度変更機構は，実行可能判定と優先度変更の 2 段階からなる．今回は，EDF でスケジュールが不可能な利用率から実行可能判定を行なった．これは，$O(1)$ で実現可能である．また，実行可能判定は，タスク受け入れ時に行なう為，より複雑な手法を用いた場合でも，実行時のオーバヘッドは分岐命令のオーバヘッドと等しい．RMT Processor では，優先度の変更はハードウェアでサポートされており，chgpr 命令を発行すれば良い．よって，優先度変更も $O(1)$ で実現可能である．優先度を与えることによるハードウェアのオーバヘッドはない為，オーバーヘッドは上記のオーバーヘッドのみである．

5. 結　　論

本論文では，より柔軟に時間制約を表現するタスクモデルを扱うことが可能なスケジューリング機構について述べた．Skip-Over モデルをスキップの割合が 50%を超えるの場合も扱えるように拡張した．また，過負荷状態において，スレッド間の優先度を考慮することにより，連続したデッドラインミスを回避する手法を提案した．評価の結果，スレッド間の優先度を考慮しない場合よりも，重要なタスクのデッドラインミス率を最大で 22.2%減少させることに成功した．

本研究では，RMT Processor 上で実行する際にパーティショニングを用いたが，他の手法として，グローバルスケジューリングも考えられる．よって，パーティショニングとグローバルスケジューリングについて比較する必要がある．グローバルスケジューリングでは，レッドタスクが到着した際に，どのブルータスクを追い出すかという問題を考えなくてはならない．

また，全てのタスクは周期タスクの場合を扱った．システムによっては，非周期タスクも存在する為，非周期タスクも扱うことが可能なように拡張しなくてはならない．どのようにスキップを想定したタスクモデルで，効率的に非周期タスクの実行時間を確保するかを考える必要がある．

更に，全てのタスクは常に横取・中断可能であり，タスク間の相互作用はないと仮定した．しかし，I/O 等の資源を保有している時に横取・中断してしまうと，資源管理の問題が発生する．資源を保持しているタスクには，十分な実行時間を確保したり，資源解放のメカニズムを提供する必要がある．

謝辞　本研究は，科学技術振興機構 CREST の支援による．

文　　献

[1] D.M. Tullsen, S.J. Eggers, and H.M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," In Proc. of the 22nd Annual International Symposium on Computer Architecture, pp.392–403, June 1995.

[2] S.J. Eggers, J.S. Emer, H.M. Levy, J.L. Lo, R.L. Stamm, and D.M. Tullsen, "Simultaneous Multithreading: A Platform for Next-Generation Processors," IEEE Micro, vol.17, pp.12–19, Sept-Oct 1997.

[3] D.M. Tullsen, S.J. Eggers, J.S. Emer, H.M. Levy, J.L. Lo, and R.L. Stamm, "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor," In Proc. of the 23rd IEEE Annual International Symposium on Computer Architecture, pp.191–191, May 1996.

[4] R. Jain, C.J. Hughes, and S.V. Adve, "Soft Real-Time Scheduling on Simultaneous Multithreaded Processors," In Proc. of the 23rd IEEE Real-Time Systems Symposium, pp.134–145, Dec. 2002.

[5] S. Kato, H. Kobayashi, and N. Yamasaki, "U-Link Scheduling: Bounding Execution Time of Real-Time Tasks with Multi-Case Execution Time on SMT Processors," In Proc. of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp.193–197, Aug. 2005.

[6] M. Hamdaoui and P. Ramanathan, "A Dynamic Priority Assignment Technique for Streams with (m,k)-Firm Deadlines," IEEE Transactions on Computers, pp.1443–1451, Dec. 1995.

[7] G. Koren and D. Shasha, "Skip-Over: Algorithms and Complexity for Overloaded Systems that Allow Skips," In Proc. of the 16th IEEE Real-Time Systems Symposium, pp.110–117, Dec. 1995.

[8] A. Marchand and M. Silly-Chetto, "RLP: Enhanced QoS Support for Real-Time Applications," In Proc. of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp.241–246, Aug. 2005.

[9] N. Yamasaki, "Responsive Multithreaded Processor for Distributed Real-Time Systems," Journal of Robotics and Mechatronics, vol.17, no.2, pp.130–141, April 2005.

[10] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," Journal of the ACM, pp.46–61, Jan. 1973.