RMd2-SIP：マルチプロセッサにおける実時間スケジューリングアルゴリズム

加藤真平† 山崎信行†

本論文の目標は，マルチプロセッサにおいて，多くのプリエンプションを起こすことなくスケジュール可能性を向上させることである．そのために，ポーショニングという新しい方式に基づいた固定優先度スケジューリングアルゴリズムを提案する．ポーショニング方式はタスク割当てとタスク実行の2つのフェーズから構成される．本論文では，タスク割当てのための SIP アルゴリズムとタスク実行のための RMd2 アルゴリズムを提案する．スケジュール可能性解析では，RMd2 と SIP を組み合わせた RMd2-SIP アルゴリズムに対するシステムのスケジュール可能な使用率の上限が 50% であることを証明する．また，タスクセットがハーモニックである場合には，その上限は 100% であり最適なアルゴリズムであることを証明する．さらに，RMd2-SIP のための厳密なスケジュール可能性判定についても述べる．シミュレーションによる評価では，RMd2-SIP が多くのプリエンプションを起こすことなく既存のアルゴリズムよりも高いスケジュール可能性を達成できることを示す．

RMd2-SIP: A Real-time Scheduling Algorithm on Multiprocessors

Shinpei Kato† and Nobuyuki Yamasaki†

The goal of this paper is to improve schedulability with few preemptions on multiprocessors. For that goal, we propose a fixed-priority scheduling algorithm based on the new scheme called portioning. The portioning scheme comprises of the allocation phase and the execution phase. In this paper, we describe the SIP algorithm for allocation and the RMd2 algorithm for execution. The schedulability analysis proves that the least upper bound of the schedulable system utilization for RMd2 with SIP, so-called RMd2-SIP, is 50%. We also prove that the bound becomes 100% that means the algorithm is optimal if the given task set is harmonic. In addition, we describe the exact schedulability test for the algorithm. The simulation shows that RMd2-SIP achieves higher schedulability than the existing algorithms without increasing preemptions.

1. はじめに

近年，ロボットやユビキタスアプリケーション等の出現により組み込み実時間システムにおいても高い処理能力が要求されるようになった．一方で，組み込み機器を対象とすると，発熱や消費電力といった点が問題となり，従来のシングルプロセッサの動作周波数を上げることは難しいという現状がある．そのため，対称型マルチプロセッシング（SMP）や同時細粒度マルチスレッディング（SMT）[24]，チップマルチプロセッシング（CMP）[20]，チップマルチスレッディング（CMT）[21] 等の各種マルチプロセッシング技術による処理能力の向上が主流になりつつある．しかしながら，実時間処理を考えた場合には，Rate Monotonic（RM）[14] や Earliest Deadline First（EDF）[14] 等のシングルプロセッサにおける最適な実時間スケジューリングアルゴリズムは，マルチプロセッサにおいては最適ではないことが証明されている[9]．マルチプロセッサにおける実時間スケジューリング手法は今日広く議論されている．

実時間スケジューリングは，主に動的優先度方式と固定優先度方式に分類される．動的優先度方式は高いスケジュール可能性を達成できるがタスクのディスパッチ処理にかかるオーバヘッドが比較的大きくなってしまう．一方，固定優先度方式はスケジュール可能性は低いがタスクのディスパッチ処理にかかるオーバヘッドを比較的小さく抑えることができる．マルチプロセッサ実時間スケジューリングでは，さらにグローバルスケジューリング方式とパーティショニング方式に大別できる．グローバルスケジューリング方式とは，システムに与えられたタスク全体で大域的に一意にタスクの優先度を決定して優先度の高いタスクから順番に各プロセッサでスケジュールする方式である．一方，

† 慶應義塾大学
Keio University

パーティショニング方式とは，まず各タスクを特定のプロセッサに割り当て，それから各プロセッサにおいて独立にタスクの優先度を決定してスケジュールする方式である．グローバルスケジューリング方式は高いスケジュール可能性を達成できるが，実装や計算が複雑であり，プリエンプションやタスクマイグレーションの回数も増えてしまうので実行時オーバヘッドが大きくなるという欠点がある．一方で，パーティショニング方式は実装や計算が簡潔であり，タスクマイグレーションが発生しないので実行時オーバヘッドも抑制することができるが，スケジュール可能性を向上させることが困難であるという欠点がある．以上のように，スケジュール可能性とオーバヘッドの間にはトレードオフの関係が存在する．

本論文の目標は，マルチプロセッサにおいて，多くのプリエンプションを起こすことなくスケジュール可能性を向上させることである．そのために，ポーショニングという新しい方式に基づいた固定優先度スケジューリングアルゴリズムを提案する．提案アルゴリズムは，簡単な計算で済むように設計されており，プロセッサ間を移動するタスクはプロセッサ数を M として最大でも $M-1$ 個だけである．スケジュール可能性解析では，提案アルゴリズムに対するシステムのスケジュール可能な使用率の上限（スケジュール可能上限）を算出する．また，厳密なスケジュール可能性判定についても述べる．シミュレーションによる評価では，提案アルゴリズムが多くのプリエンプションを起こすことなく既存の固定優先度アルゴリズムよりも高いスケジュール可能性を達成できることを示す．

2. 関連研究

Baruah らが提案した Pfair スケジューリング手法[4]は，マルチプロセッサにおける最適な実時間スケジューリング手法であることが知られている．Pfair スケジューリングでは，各タスクを非常に短い時間（クォンタム）を持つサブタスクに分割し，クォンタムごとに新しいサブタスクを選択してスケジュールを行うので，プリエンプションやプロセッサ間でのタスクマイグレーションの多発にともなうオーバヘッドがしばしば問題となる．近年では各種 Pfair アルゴリズムの実用性を示す研究も報告されているが[6]，オーバヘッドも含んだ実際のパフォーマンスに関して，必ずしも Pfair アルゴリズムがその他のアルゴリズムよりも優れているとはいえない．LLREF[8] はマルチプロセッサにおけるもう 1 つの最適なスケジューリングアルゴリズムである．Pfair スケジューリングとは異なり，LLREF では各タスクの実行をサブタスクに分割することはしないので，クォンタムごとにタスクのプリエンプションを考慮する必要はない．しかしながら，タスクの実行時間や周期の関係によっては大きな計算量が必要になることもあり，Pfair スケジューリングよりも統計的なオーバヘッドは大きくなってしまう可能性がある．

これら 2 つのアルゴリズムは動的優先度グローバルスケジューリング方式に分類され，高いスケジュール可能性を実現できるが実装や計算が複雑になってしまう．一方で，RM-FF[9] や RM-FFDU[19] は固定優先度パーティショニング方式に分類され，その簡潔さの点で実際のシステムにおいてしばしば利用されるアルゴリズムである．しかしながら，システムのスケジュール可能上限は 50% かそれ以下であることが知られている．Oh らは最悪時の上限が約 41% であると証明した[18]．近年，Lopez らによってこの上限は改善されたが[15]，それでも Pfair アルゴリズムや LLREF に比べると理論的なスケジュール可能上限は非常に低い．このほかにも，EDF-US[10]（動的優先度グローバルスケジューリング）や EDF-FF[16]（動的優先度パーティショニング），RM-US[2]（固定優先度グローバルスケジューリング）等が提案されてきたが，スケジュール可能性とオーバヘッドのトレードオフの関係は今日でも存在する．実際，Pfair アルゴリズムと LLREF アルゴリズム以外のすべてのアルゴリズムのスケジュール可能上限はプロセッサ数を M として $(M+1)/2M$ である[7]．$M \to \infty$ で最悪となり，たかだか 50% となる．一方で，これらのアルゴリズムは Pfair アルゴリズムや LLREF よりも計算と実装が簡単でありオーバヘッドも小さいので実用性は高いとされている．

Anderson らは，マルチプロセッサにおけるスケジュール可能性とオーバヘッドのトレードオフを解決するために新しいスケジューリング方式を提案した[1]．Anderson らの方式では，従来のパーティショニング方式と同様にタスクの CPU 使用率を基に各プロセッサに割り当てていくが，あるプロセッサの使用率がそのプロセッサのスケジュール可能上限を超えてしまう場合には，最後に割り当てたタスクを 2 つに分割する．1 つは現在の割当て先のプロセッサに分配され，もう 1 つは次の割当て先のプロセッサに分配される．Anderson らはこの新しい方式の最初の試みとしてソフトリアルタイムシステムを対象としたスケジューリングアルゴリズム EDF-fm を提案し，その最大遅延時間の解析を行った．さらに，最大遅延時間を短縮するための発見的手法を提案した．Andersson

らもまた EDF-fm と同じ方式に基づくスケジューリングアルゴリズム EKG を提案した[3]．EDF-fm とは異なり，EKG はハードリアルタイムシステムを対象としたアルゴリズムである．また，EKG におけるタスク割当てアルゴリズムは，EDF-fm におけるタスク割当てアルゴリズムと似ているが，タスクの割当て自体が Next-Fit アルゴリズム[9] に基づいている点が異なる．EKG に対するシステムのスケジュール可能上限はアルゴリズム特有のパラメータ k に依存している．$k = 2$ で最悪となり上限は 66% となる．一方，$k = M$ で最適となり上限は 100% となるが，プリエンプションとタスクマイグレーションの回数が増加してしまう．本論文では，EDF-fm や EKG のようにパーティショニング方式に基づいているが必要に応じてタスクを分割できる方式をポーショニング方式と定義する．

3. システムモデル

本論文では実時間スケジューリングにおける一般的なシステムモデルを仮定する．システムは M 個のプロセッサ $P_1, P_2, ..., P_M$ から構成するものとする．そして，N 個のタスクから構成されるタスクセット $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$ をシステムに与える．各タスク τ_i は (C_i, T_i) というタプルで定義する．C_i は最悪実行時間であり，T_i は周期である．τ_i の CPU 使用率を $U_i = C_i/T_i$ で表す．また，あるタスクの集合 Λ に含まれるタスクの CPU 使用率の合計を $U(\Lambda) = \sum_{\tau_i \in \Lambda} U_i$ で表す．すなわち，$U(\Gamma)$ はタスクセットの CPU 使用率を意味する．$U(\Gamma)$ を 0〜100%で正規化した $U(\Gamma)/M$ をシステム使用率と定義する．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 k 番目のジョブを $\tau_{i,k}$ と表し，$\tau_{i,k}$ は時刻 $r_{i,k}$ でリリースされ，デッドライン $d_{i,k}$ は次のジョブのリリース時刻とする．すなわち，$d_{i,k} = r_{i,k+1} = r_{i,k} + T_i$ となる．ジョブ $\tau_{i,k}$ の実行開始時刻および実行終了時刻は各々 $s_{i,k}$ および $f_{i,k}$ で表す．

アルゴリズムは以下の仮定のもとで設計する．システムはメモリ共有型のマルチプロセッサであり，各プロセッサはコードとデータを共有できるものとする．すべてのタスクはプリエンプト可能で互いに独立しており並列性はないものとする．よって，1 つのタスクを複数のプロセッサ上で並列に実行することはできない．システムの実行中に新たなタスクが到着したり，すでにシステムに存在するタスクが消滅したりすることはないものとする．また，本論文の対象はシステム全体の設計ではなくスケジューリングアルゴリズムの設計であるため，プリエンプションやタスクマイグレーションの時間コストは考えないものとする．これらのコストはプロセッサの性能に大きく依存し，スケジューリングアルゴリズムの観点ではその発生頻度の方が重要になる．また，各プロセッサ間でコードとデータを共有していることを仮定すると，プロセッサ内のコンテキスト切替えとプロセッサ間のコンテキスト切替え（タスクマイグレーション）のコストはほぼ同一であると見なすことができる．本論文では最悪実行時間に基づいてスケジューリングを行うので，実システムにおいてタスクマイグレーションを行うことでキャッシュのヒット率等が低下し，タスクの実行時間が延びてしまうことは問題としない．そのため，スケジューリングアルゴリズムのオーバヘッドを比較する場合にはプリエンプション数を評価指標とする．

4. スケジューリングアルゴリズム

本章では，ポーショニング方式に基づいた実時間スケジューリングアルゴリズムを提案する．従来のパーティショニング方式では各タスクを特定のプロセッサに割り当て，タスクのプロセッサ間の移動を禁止しているのに対して，ポーショニング方式では各タスクの CPU 使用率を特定のプロセッサに割り当て，複数のプロセッサにその使用率が割り当てられたタスクはプロセッサ間を移動することができる．すなわち，いくつかのタスクに関しては，複数のプロセッサに分割され各プロセッサにおいて割り当てられた使用率が実行のために予約される．スケジューリングアルゴリズムは，分割されたタスクに対して複数のプロセッサで並列に実行しないように設計する必要がある．本論文では固定優先度のポーショニング方式に焦点を当て，動的優先度への対応は今後の課題とする．ポーショニング方式はタスク割当てとタスク実行の 2 つのフェーズから構成される．本章では，4.1 節および 4.2 節でそれぞれアルゴリズムを提案する．

4.1 タスク割当てフェーズ

まず，各プロセッサにタスクの CPU 使用率を割り当てるアルゴリズムとして ***S**equential allocation in **I**ncreasing **P**eriod*（SIP）を提案する．**図 1** に SIP のアルゴリズムを示す．各タスクは $T_1 \leq T_2 \leq \cdots \leq T_N$ となるように整列しているものとする．また，プロセッサ P_j に割り当てられたタスクから構成されるタスクセットを Λ_j と定義する．

SIP は，各プロセッサ P_j に対して，CPU 使用率 $U(\Lambda_j)$ が P_j のスケジュール可能上限よりも小さい間はタスクを順に P_j に割り当てる（3〜6 行目）．スケ

Input: task set τ ($T_1 \le T_2 \le ... \le T_N$)
Output: per-processor task sets $\{\Lambda_j\}$
Initial states: $i \leftarrow 1$, $j \leftarrow 1$ and $\Lambda_1 \leftarrow \emptyset$

1. $\Lambda_j^* \leftarrow \Lambda_j \cup \tau_i$;
2. **if** $U(\Lambda_j^*) \le U_{lub}(\Lambda_j^*)$
3. $\quad \Lambda_j \leftarrow \Lambda_j \cup \tau_i$;
4. $\quad$ **if** $U(\Lambda_j^*) = U_{lub}(\Lambda_j^*)$
5. $\quad\quad j \leftarrow j + 1$;
6. $\quad\quad \Lambda_j \leftarrow \emptyset$;
7. **else if** $j = M$
8. $\quad$ allocation fails;
9. **else**
10. $\quad$ split τ_i into $\tau_i'(T_i, C_i')$ and $\tau_i''(T_i, C_i'')$ where $C_i' = T_i\{U_{lub}(\Lambda_j^*) - U(\Lambda_j)\}$ and $C_i' = C_i - C_i''$;
11. $\quad \Lambda_j \leftarrow \Lambda_j \cup \tau_i'$;
12. $\quad j \leftarrow j + 1$;
13. $\quad \Lambda_j \leftarrow \emptyset \cup \tau_i''$;
14. **if** $i = N$
15. $\quad$ allocation exits;
16. $i \leftarrow i + 1$;
17. go back to step 1.;

図 1 SIP アルゴリズム
Fig. 1 Algorithm SIP.

ジュール可能上限については 5 章で詳しく述べる．そして，あるタスク τ_i を P_j に割り当てた際に，$U(\Lambda_j)$ がスケジュール可能上限を超えてしまうようなら，P_j へのタスク割当てを終了する．このとき，これ以上割り当てられるプロセッサがなければアルゴリズムは失敗する（8 行目）．そうでなければ，τ_i を τ_i' と τ_i'' に分割する．本論文では，τ_i' を **division-1** タスク，τ_i'' を **division-2** タスクと定義する．ここで，「分割」ということが，実際にタスクの内容を 2 つの部分に分離することではないことに注意されたい．τ_i' と τ_i'' は，P_j および P_{j+1} において，各々 τ_i の実行時間を予約しておくための実時間スケジューリング上の仮想的なタスクにすぎない．よって，τ_i は P_j と P_{j+1} の間を移動しながら実行することになる．τ_i' と τ_i'' の予約された実行時間 C_i' および C_i'' は以下のように表せる．

$$C_i' = T_i\{U_{lub}(\Lambda_j \cup \tau_i) - U(\Lambda_j)\}$$
$$C_i'' = C_i - C_i'$$

次に，これ以上割り当てるタスクがなければアルゴリズムは終了する（15 行目）．そうでなければ，同じ処理を繰り返す（17 行目）．

例 1 周期の短い順に整列された以下の 8 個のタスクの割当てを考える．$\tau_1(1,5)$，$\tau_2(2,5)$，$\tau_3(1,8)$，$\tau_4(5,10)$，$\tau_5(3,12)$，$\tau_6(2,12)$，$\tau_7(12,20)$，$\tau_8(4,20)$．ここで，同じ優先度を持ったタスクが複数存在した場合の優先付け（Tie breaking）には特に規則は設けていない．最善の優先付け規則に関しては今後の課題とする．これら 8 個のタスクは SIP によって**図 2** に示すように 3 つのプロセッサに割り当てられる．その結果，3 つのタスクセット $\Lambda_1 = \{\tau_1, \tau_2, \tau_3, \tau_4'(1,10)\}$，$\Lambda_2 = \{\tau_4''(4,10), \tau_5, \tau_6'(1,12)\}$，$\Lambda_3 = \{\tau_6''(1,12), \tau_7, \tau_8\}$ が生成される．それぞれ $U(\Lambda_1) \simeq 0.83$，$U(\Lambda_2) \simeq 0.73$，$U(\Lambda_3) \simeq 0.88$ となる．スケジュール可能上限の計算式は 5.2 節で詳しく述べるが，各々 $U_{lub}(\Lambda_1) \simeq 0.83$，$U_{lub}(\Lambda_2) \simeq 0.78$，$U_{lub}(\Lambda_3) \simeq 0.92$ となる． □

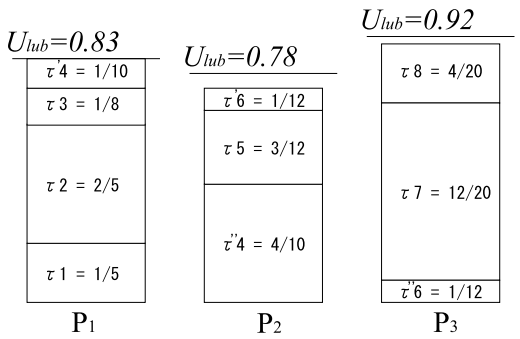

図 2 SIP のタスク割当て例
Fig. 2 Allocation example of SIP.

SIP は EDF-fm におけるタスク割当てアルゴリズムに似ている．しかしながら，EDF-fm がタスクを無作為，CPU 使用率の高い順，CPU 使用率の低い順，または実行時間の長い順で割り当てる一方で，SIP はタスクを周期の短い順に割り当てる．よって，τ_i が P_j と P_{j+1} に分割されたとして，SIP では以下の関係が必ず成り立っている．

$$\max\{T_l \mid \tau_l \in \Lambda_j\} \le T_i \le \min\{T_k \mid \tau_k \in \Lambda_{j+1}\}$$

この性質が 5 章でスケジュール可能性解析を行うときに重要になる．また，EDF-fm では各タスクの CPU 使用率は 50% 以下でなければならないが，提案アルゴリズムでは各タスクの CPU 使用率は 100% まで扱うことができる．さらに，EKG のように各タスクを heavy か light かに分類する必要もないことに注意されたい．

4.2 タスク実行フェーズ

本節では，SIP によって割り当てられたタスクを各プロセッサごとにスケジュールするアルゴリズムとして ***R**ate **M**onotonic with a restriction of **d**ivision-2* (RMd2) を提案する．SIP と同様に，RMd2 アルゴリズムも**図 3** に示すように簡潔である．ここでは，プロセッサ P_j に割り当てられたタスクセットをスケジュールすることを仮定している．RMd2 は P_j 上でつねに division-2 タスクに最も高い優先度を与え，division-1 タスクに最も低い優先度を与える．アルゴリズムの動

Definition: Λ'_j is a set of ready tasks in Λ_j

1. select task τ_k from Λ'_j based on RM where ties are broken in favor of a division-2 task;
2. **if** τ_k is a division-2 task on P_j and is running on P_{j-1}
3. execute a task that has the second priority in Λ'_j;
4. **else**
5. execute τ_k;

図 3 RMd2 アルゴリズム
Fig. 3 Algorithm RMd2.

作は RM とほとんど同じであるが，P_j における実行可能な最高優先度タスクが division-2 タスクであり，かつその division-1 タスクが P_{j-1} で実行している場合には，P_j 上で 2 番目の優先度を持つタスクを実行する．

RMd2 の特徴としてあらゆるプリエンプションやマイグレーションのタイミングが RM のスケジュールによって生成できることである．付加的な作業として，division-1 タスクの実行によって division-2 タスクの実行を中断・再開するために，プロセッサ間で同期をとる必要がある．本論文ではメモリ共有型マルチプロセッサを仮定しているため，各タスクの状態はすべてのプロセッサから参照可能である．よって，P_j および P_{j-1} に分割されたタスク τ_k をプロセッサ P_j で実行する場合に，τ_k が P_{j-1} で実行されているか否かを確認することのコストは非常に小さいと考えられる．一方で，EDF-fm や EKG では，EDF のスケジュールには存在しないタイミングで付加的なイベントが必要になる．具体的には，EDF-fm では分割したタスクはそのほかのタスクよりも高い優先度が静的に割り当てられるので，RM のスケジュールにはないタイミングで分割したタスクを実行しなければならない．EKG では **timea** と **timeb** と呼ばれる特別なタイミングを各タスクがリリースされたときに計算し，それらのタイミングでプリエンプションを発生させなければならない．

例 2 **図 4** は 4.1 節の例 1 で与えられた 8 個のタスクが RMd2 によってスケジュールされる様子を示している．プロセッサ P_1 には division-2 タスクは存在しないので，タスクセット Λ_1 は RM に基づいてスケジュールされる．一方，プロセッサ P_2 には division-2 タスク τ''_4 が存在するので，そのスケジュールはプロセッサ P_1 で実行される τ'_4 から干渉を受ける可能性がある．まず，τ''_4 は時刻 0 に P_2 でスケジュールされる．τ'_4 は時刻 4 に P_1 でスケジュールされるので，τ''_4 は τ'_4 の干渉を受けることなく時刻 4 に実行を終了する．ここで，τ''_4 の終了は実際のタスク τ_4 の終了ではないので，タイマを使って管理する必要があることに注意されたい．次に，τ_4 の 2 周期目の実行に関しては，τ''_4 が時刻 10 に P_2 でスケジュールされるが，τ'_4 が時刻 13 に P_1 でスケジュールされるため，τ''_4 の実行は一時中断される．代わりに 2 番目の優先度を持つ τ_5 がスケジュールされる．時刻 14 に τ'_4 の実行が P_1 で終了するので，τ''_4 の実行がこの時刻に再開される．τ_4 の 3 周期目の実行に関しても，τ''_4 の実行は τ'_4 の実行から干渉を受ける．この場合，時刻 23 に τ''_4 の実行は中断されるが，P_2 で実行可能なタスクは存在しない．よって，タイムスロットはアイドル状態となる．タスクセット Λ_3 もまた division-2 タスク τ''_6 を含んでいるが，図 4 の範囲では，τ'_6 と τ''_6 の実行はオーバラップしないため，τ''_6 はいっさいの干渉を受けずに実行される． □

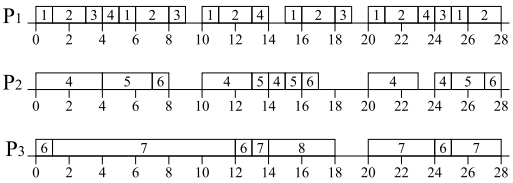

図 4 RMd2 のスケジュール例
Fig. 4 Schedule of RMd2.

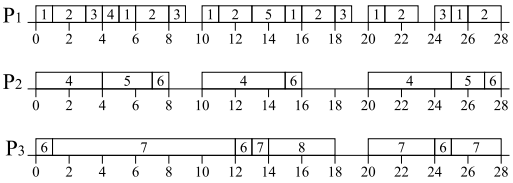

図 5 プリエンプションを考慮した RMd2 のスケジュール例
Fig. 5 Schedule of preemption-care RMd2.

プリエンプションの削減． ここで，RMd2 における不必要なタスクマイグレーションによって引き起こされるプリエンプションを削減する方法について考える．本論文では，具体的なアルゴリズムおよび実装方法に関しては言及しない．図 4 において，2 つのタスク τ_4 と τ_6 が分割されているが，不必要にプロセッサ間を移動していることが分かる．たとえば，**図 5** に示すように，時刻 13 に τ_4 の代わりに τ_5 を P_2 から P_1 へ移動させればプリエンプションを削減できる．また，時刻 23 に τ_4 を P_1 でスケジュールする代わりに P_2 でスケジュールすればプリエンプションを削減できる．

5. スケジュール可能性解析

本章では，RMd2 と SIP を組み合わせた RMd2-SIP アルゴリズムに対するシステムのスケジュール可能上

限 U_{lub} およびスケジュール可能性判定式を求める．以下の解析では，タスク割当てフェーズにおいて，あるタスクが分割され，その division-2 タスクとそれに続く n 個のタスクがプロセッサ P_j に割り当てられたものと仮定する．そして，これら $n+1$ 個のタスクからなるタスクセット Λ_j のスケジュール可能上限を解析する．ここで，分割されたタスクを τ_s と表記し，その division-2 タスクを τ_s'' と表記する．そのほかの n 個のタスクに関しては，表記の簡略化のため，$\tau_1, \tau_2, ..., \tau_n$ と表記する．また，T_s と T_1 の比率を $R_s = T_1/T_s$ で表し，そのほかの連続する 2 つのタスク τ_i と τ_{i+1} の周期の比率を $R_i = T_{i+1}/T_i$ で表す．ここで，$R_1 R_2 \cdots R_{k-1} = T_k/T_1$ となることに注意されたい．Λ_j のスケジュール可能上限が求まれば，そのほかのプロセッサ上のタスクセットのスケジュール可能上限も同様にして求められる．

5.1 スケジュール可能上限

RMd2 において，division-2 タスクは各プロセッサでの最高優先度タスクとなるが，その実行は division-1 タスクによって延期される可能性がある．この現象は非周期タスクを扱う場合にしばしば利用される Deferrable Server （DS）[22] に最高優先度を与えた場合と同じである．Strosnider らによると，DS が存在する環境における最悪のスケジュールは，DS の実行が最も長く延期され，かつ DS 以外のすべてのタスクが DS の実行開始と同時にリリースされる場合に発生する[22]．また，この最悪のスケジュールは T_s，T_1 および T_n の関係によって 3 つのケースを解析する必要がある．RMd2 におけるこれら 3 つのケースは**図 6**，**図 7** および**図 8** のようになり，それぞれ 5.1.1 項，5.1.2 項および 5.1.3 項で解析を行う．

スケジュール可能上限を求めるためには，τ_s'' の k 番目のジョブの終了時刻から $k+1$ 番目のジョブのリリース時刻までの最小のスラック S を知る必要がある．SIP の性質から，つねに τ_s には Λ_j の中で最高優先度が与えられる．よって，その実行は P_{j-1} で実行される τ_s' 以外からは影響を受けないので，τ_s'' の終了時刻は遅くても $r_{s,k} + C_s' + C_s'' = r_{s,k} + C_s = r_{s,k+1} - (T_s - C_s)$ となり，最小スラックは $S = T_s - C_s$ となる．

ここで，あるプロセッサ P_j に division-2 タスクが存在しない場合には，P_j 上のすべてのタスクのスケジュールは完全に RM に従うことに注意されたい．よって，この場合は RMd2 と RM のスケジュール可能上限は同一となる．4.1 節の例 1 におけるタスクセット Λ_1 はこの場合に属することになる．近年，RM のスケジュール可能上限は改善されてきたが[5),11),13),17)]，

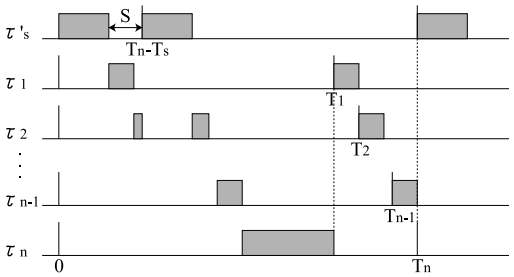

図 6 τ_s'' が T_1 および T_n 以内に 2 回実行（ケース 1）
Fig. 6 τ_s'' is executed twice within T_1 and T_n (Case 1).

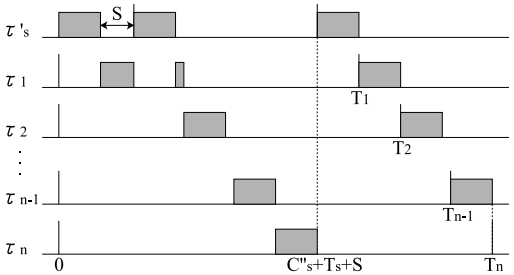

図 7 τ_s'' が T_1 および T_n 以内に 3 回実行（ケース 2）
Fig. 7 τ_s'' is executed three times within T_1 and T_n (Case 2).

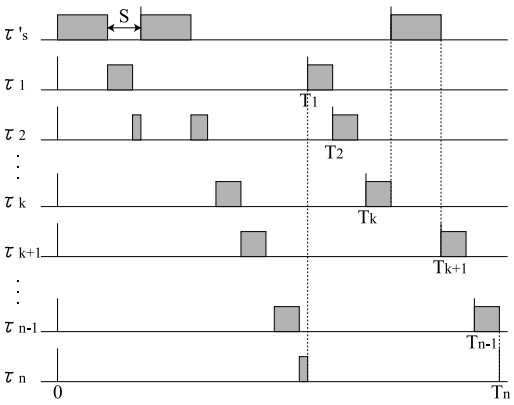

図 8 τ_s'' が T_1 以内に 2 回，かつ T_n 以内に 3 回実行（ケース 3）
Fig. 8 τ_s'' is executed twice within T_1 and three times within T_n (Case 3).

本論文では Liu らが提案した以下の公式を利用する．

$$U_{lub} = n(2^{1/n} - 1) \quad (1)$$

すなわち，我々が考慮する必要があるのは P_j に division-2 タスクが存在する場合のみである．

5.1.1 ケース 1 の解析

各タスクの実行時間は以下のように定義できる．

$$C_i = T_{i+1} - T_i \ \ (1 \le i \le n-1)$$
$$C_n = T_1 - 2C_s'' - \sum_{j=1}^{n-1} C_j = 2T_1 - 2C_s'' - T_n$$

よって，タスクの CPU 使用率の合計は式 (2) で表せる．

$$\begin{aligned} U &= \frac{C_s''}{T_s} + \frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \\ &= U_s'' + \sum_{i=1}^{n-1} \frac{T_{i+1} - T_i}{T_i} + \frac{2T_1 - 2C_k'' - T_n}{T_n} \\ &= U_s'' + \sum_{i=1}^{n-1} \frac{T_{i+1}}{T_i} + 2\left(1 - \frac{C_s''}{T_1}\right)\frac{T_1}{T_n} - n \\ &= U_s'' + \sum_{i=1}^{n-1} R_i + \frac{2 - \frac{2U_s''}{R_s}}{R_1 R_2 \cdots R_{n-1}} - n \quad (2) \end{aligned}$$

ここで，式 (2) を R_i に関して偏微分する．

$$\frac{\partial U}{\partial R_i} = 1 - \frac{2 - \frac{2U_s''}{R_s}}{R_i^2(\prod_{j \ne i}^{n-1} R_j)}$$

このことから，$P = R_1 R_2 \cdots R_{n-1}$ と置くと，U は最小となるのは，各 R_i（$1 \le i \le n-1$）に対して以下の等式が成り立つ場合であることが分かる．

$$R_i P = 2 - \frac{2U_s''}{R_s}$$

すなわち，すべての R_i が同じ値を持つことになるので，各 R_i は以下のように表せる．

$$R_1 = R_2 = \cdots = R_{n-1} = \left(2 - \frac{2U_s''}{R_s}\right)^{1/n}$$

よって，U の上限値は以下のように書ける．余白の都合上，$K = 2 - 2U_s''/R_s$ とおく．

$$\begin{aligned} U_{lub} &= U_s'' + (n-1)K^{1/n} + \frac{K}{K^{n-1/n}} - n \\ &= U_s'' + nK^{1/n} - K^{1/n} + K^{1/n} - n \\ &= U_s'' + n(K^{1/n} - 1) \end{aligned}$$

U_{lub} を最小にするには R_s を最小にする必要がある．ここで，図 6 から以下の不等式は明らかに成り立つ．

$$\sum_{i=1}^{n} C_i = T_1 - 2C_s'' \ge S = T_s - C_s$$

両辺を T_s で割ると R_s の範囲が求まる．

$$R_s - 2U_s'' \ge 1 - U_s$$
$$R_s \ge 2U_s'' - U_s + 1$$

$T_s \le T_1$ より $R_s \ge 1$ が導かれるので，U_{lub} は $R_s = \max\{1, 2U_s'' - U_s + 1\}$ のときに最小値となる．最終的に，$R_s = \max\{1, 2U_s'' - U_s + 1\}$ として，U_{lub} は U_s'' と U_s の関数として式 (3) で表される．

$$U_{lub} = U_s'' + n\left\{\left(2 - \frac{2U_s''}{R_s}\right)^{1/n} - 1\right\} \quad (3)$$

$n \to \infty$ の極限をとることで U_{lub} をさらに最小化できる．

$$\lim_{n\to\infty} U_{lub} = U_s'' + \ln\left(2 - \frac{2U_s''}{R_s}\right) \quad (4)$$

この値が $R_s = 1$ のときに最小化されることは明白であり，式 (4) の最小値は以下に示す $\hat{U}_{lub}$ となる．

$$\hat{U}_{lub} = U_s'' + \ln\{2(1 - U_s'')\}$$

結果として，最悪時のスケジュール可能上限は，$U_s'' = 1/2$ および $U_s = 1$ のときに $\hat{U}_{lub} = 0.5$ となる．ここで，この上限値は $T_s = T_1 = T_2 = \cdots = T_n$ および $C_1 = C_2 = \cdots = C_n = 0$ という特別な場合にのみ得られることに注意されたい．

5.1.2 ケース 2 の解析

各タスクの実行時間は以下のように定義できる．

$$C_i = T_{i+1} - T_i \ \ (1 \le i \le n-1)$$
$$C_n = T_1 - 3C_s'' - \sum_{j=1}^{n-1} C_j = 2T_1 - 3C_s'' - T_n$$

式 (2) を求めたときと同様に考えて，タスクの CPU 使用率の合計は式 (5) のように表せる．

$$\begin{aligned} U &= U_s'' + \sum_{i=1}^{n-1} \frac{T_{i+1}}{T_i} + \left(2 - \frac{3C_s''}{T_1}\right)\frac{T_1}{T_n} - n \\ &= U_s'' + \sum_{i=1}^{n-1} R_i + \frac{2 - \frac{3U_s''}{R_s}}{R_1 R_2 \cdots R_{n-1}} - n \quad (5) \end{aligned}$$

ケース 1 と同じ手順により，スケジュール可能上限は式 (6) で求まる．

$$U_{lub} = U_s'' + n\left\{\left(2 - \frac{3U_s''}{R_s}\right)^{1/n} - 1\right\} \quad (6)$$

ここで，U_{lub} を最小にするには R_s を最小にする必要がある．図 7 から T_1 は以下のように書ける．

$$T_1 = T_s + 2C_s'' + S = 2T_s + 2C_s'' - C_s$$

両辺を T_s で割ると R_s の最小値が求められる．

$$R_s = 2U_s'' - U_s + 2$$

よって，U_{lub} の最小値は式 (7) で表される．

$$\begin{aligned} U_{lub} &= U_s'' + n\left\{\left(2 - \frac{3U_s''}{2U_s'' - U_s + 2}\right)^{1/n} - 1\right\} \\ &= U_s'' + n\left\{\left(\frac{U_s'' - 2U_s + 4}{2U_s'' - U_s + 2}\right)^{1/n} - 1\right\} \quad (7) \end{aligned}$$

$n \to \infty$ の極限をとることで U_{lub} をさらに最小化で

きる．

$$\lim_{n\to\infty} U_{lub} = U_s'' + \ln\left(\frac{U_s''-2U_s+4}{2U_s''-U_s+2}\right) \quad (8)$$

ここで，式 (8) を U_s'' に関して偏微分する．

$$\frac{\partial U_{lub}}{\partial U_s''} = \frac{2U_s''^2+(10-5U_s)U_s''+2U_s^2-5U_s+2}{(U_s''-2U_s+4)(2U_s''-U_s+2)}$$

これにより，U_{lub} を最小化する U_s'' が求まる．

$$\hat{U}_s'' = \frac{5U_s-10+\sqrt{9U_s^2-60U_s+84}}{4}$$

$U_s'' \geq 0$ なので，最悪時のスケジュール可能上限は，$U_s'' \simeq 0.186$ および $U_s = 1$ のときに $\hat{U}_{lub} \simeq 0.652$ となる．

5.1.3 ケース 3 の解析

最後に，ケース 1 とケース 2 を組み合わせたケース 3 について解析を行う．ケース 3 では，各タスクの実行時間は以下のように定義できる．

$$\begin{aligned}
C_i &= T_{i+1} - T_i \ \ (1 \leq i \leq k-1)\\
C_k &= T_{k+1} - C_s'' - T_k\\
C_i &= T_{i+1} - T_i \ \ (k+1 \leq i \leq n-1)\\
C_n &= T_1 - 2C_s'' - \sum_{j=1}^{n-1} C_j = 2T_1 - C_s'' - T_n
\end{aligned}$$

タスクの CPU 使用率の合計は式 (9) のように表せる．

$$\begin{aligned}
U &= U_s'' + \sum_{i=1}^{k-1}\frac{T_{i+1}-T_i}{T_i} + \frac{C_k}{T_k} + \sum_{i=k+1}^{n-1}\frac{T_{i+1}-T_i}{T_i} + \frac{2T_1-C_s''-T_n}{T_n}\\
&= U_s'' + \sum_{i=1}^{k-1}\frac{T_{i+1}}{T_i} + U_k + \sum_{i=k+1}^{n-1}\frac{T_{i+1}}{T_i} + \left(2-\frac{C_s''}{T_1}\right)\frac{T_1}{T_n} - (n-1)\\
&= U_s'' + \sum_{i=1}^{k-1} R_i + U_k + \sum_{i=k+1}^{n-1} R_i + \frac{2-\frac{U_s''}{R_s}}{R_1R_2\cdots R_{n-1}} - (n-1) \quad (9)
\end{aligned}$$

ここで，式 (9) には n と k の 2 つの変数が存在するので，U を最小化する R_i 求めるのは難しい．そこで，k の値を固定する．$C_k = T_{k+1} - C_s'' - T_k$ より $U_k = R_k - C_s''/T_k - 1$ が導かれる．よって，式 (9) は以下のように書き直せる．

$$U = U_s'' + \sum_{i=1}^{n-1} R_i - \frac{C_s''}{T_k} + \frac{2-\frac{U_s''}{R_s}}{R_1R_2\cdots R_{n-1}} - n$$

$T_1 \leq T_2 \leq \cdots \leq T_n$ であることを考慮すると，$k=1$ のときに式 (9) が最小値をとることは明らかである．よって，U は k に関して以下のように最小化できる．

$$U = U_s'' + U_1 + \sum_{i=2}^{n-1} R_i + \frac{2R_s - U_s''}{R_sR_1R_2\cdots R_{n-1}} - (n-1)$$

次に，U を最小にする R_i を求めるために，$2 \leq i \leq n-1$ として，上式を R_i に関して偏微分する．

$$\frac{\partial U}{\partial R_i} = 1 - \frac{2R_s - U_s''}{R_sR_1R_i^2\prod_{j\neq i}^{n-1} R_j}$$

これにより，すべての R_i が以下の値を持つときに U が最小になることが分かる．

$$R_2 = R_3 = \cdots = R_{n-1} = \left(\frac{2R_s-U_s''}{R_sR_1}\right)^{1/(n-1)}$$

よって，U は以下のように最小化される．

$$\begin{aligned}
U &= U_s'' + U_1 + (n-2)\left(\frac{2R_s-U_s''}{R_sR_1}\right)^{1/(n-1)} + \frac{2R_s-U_s''}{R_sR_1\left(\frac{2R_s-U_s''}{R_sR_1}\right)^{1-1/(n-1)}} - (n-1)\\
&= U_s'' + U_1 + (n-1)\left\{\left(\frac{2R_s-U_s''}{R_sR_1}\right)^{1/(n-1)} - 1\right\}
\end{aligned}$$

$C_k = T_{k+1} - C_s'' - T_k$ および $k=1$ であることから，$R_1 = U_1 + U_s''/R_s + 1$ が導かれる．また，T_s と T_1 の関係はケース 1 と同じなので，上記の式を最小にする R_s は $\max\{1, 2U_s'' - U_s + 1\}$ となる．最終的に，$R_s = \max\{1, 2U_s'' - U_s + 1\}$ とおいて，U_{lub} は U_s'' および U_s，U_1 の関数として式 (10) で表される．

$$U_{lub} = U_s'' + U_1 + (n-1)\left[\left\{\frac{2R_s-U_s''}{R_s(U_1+1)+U_s''}\right\}^{1/(n-1)} - 1\right] \quad (10)$$

$n \to \infty$ の極限をとることで U_{lub} をさらに最小化できる．

$$\lim_{n\to\infty} U_{lub} = U_s'' + U_1 + \ln\left\{\frac{2R_s-U_s''}{R_s(U_1+1)+U_s''}\right\} \quad (11)$$

T_s と T_1 の関係はケース 1 と同様なので，式 (11) は

$R_s = 1$ のときに最小値をとる．

$$\hat{U}_{lub} = U_s'' + U_1 + \ln\left(\frac{2 - U_s''}{U_s'' + 1 + U_1}\right)$$

この値は $U_s'' = 1/2$ および $U_1 = 0$ のときに絶対的な最小値 $\hat{U}_{lub} \simeq 0.5$ となる．この場合のスケジュールはケース 1 と同一であり，$T_s = T_1 = T_2 = \cdots = T_k$ および $T_{k+1} = T_{k+2} = \cdots = T_n = T_s + C_s''$，$C_1 = C_2 = \cdots = C_n = 0$ という特別な場合にのみ得られる．

以上の解析より，RMd2 に対するスケジュール可能上限はケース 1 の 50% となる．プロセッサごとの上限が 50% なので，RMd2-SIP に対するシステムのスケジュール可能上限も 50% であることが分かる．

5.2 スケジュール可能性判定

前節で求めた上限値は絶対的な最悪時を想定している．タスク割当てフェーズでプロセッサごとのスケジュール可能性判定を行うときには，各タスクの周期や実行時間は既知であるため，上限値 50% をスケジュール可能性判定に用いるのは得策ではない．本節では，厳密なスケジュール可能性判定について述べる．

まず，τ_s'' がタスクセット Γ の中の最後のタスクであった場合，プロセッサ P_j で τ_s'' がスケジュール可能であることは自明である．よって，$n \geq 1$ の場合を考えればよい．まず，各タスクの周期は既知であるからスケジュール可能性判定時に $R_s = T_1/T_s$ は算出可能である．このことを考慮すると，τ_s'' が T_1 および T_n 以内に 2 回実行される場合は式 (3)，3 回実行される場合は式 (6) を判定式として利用すればよい．より一般的に，τ_s'' が T_1 および T_n 以内に L 回（$L \geq 2$）実行される場合を考える．各タスクの周期および実行時間は既知であるから，図 6 および図 7 を参考にして L は以下のように求められる．

$$L = 2 + \max\left\{\left\lfloor \frac{T_1 - 2C_s'' - (T_s - C_s)}{T_s} \right\rfloor, 0\right\}$$

この L を用いて，より厳密なプロセッサ P_j のスケジュール可能上限 U_{lub} は式 (12) で表される．

$$U_{lub} = U_s'' + n\left\{\left(2 - \frac{LU_s''}{R_s}\right)^{1/n} - 1\right\} \quad (12)$$

前節のケース 3 の解析結果から，τ_s'' が T_1 以内に $L-1$ 回，T_n 以内に L 回実行されるケースは，τ_s'' が T_1 および T_n 以内に L 回実行されるケースに包含される．よって，$n \geq 1$ であるすべての場合に式 (12) は有効である．$n \to \infty$ のとき，式 (12) は式 (13) となる．

$$U_{lub} = U_s'' + \ln\left(2 - \frac{LU_s''}{R_s}\right) \quad (13)$$

4.1 節の例 1 における P_2 のスケジュール可能上限は，$U_s'' = 4/10$，$U_s = 5/10$，そしてハーモニックチェイン[12)☆]を考慮した $n = 1$ を式 (12) に代入することで $U_{lub}(\Lambda_2) \simeq 0.78$ と求まる．また，P_3 に関しては，$U_s'' = 1/12$，$U_s = 2/12$，そしてハーモニックチェイン $n = 1$ を代入して $U_{lub}(\Lambda_3) \simeq 0.92$ と求まる．

5.3 ハーモニックケースの解析

Kuo らはハーモニックチェインを考慮することによって，RM のスケジュール可能上限を改善できることを示した[12)]．RMd2 は RM に基づいているので，式 (3)，式 (7)，および式 (10) における n の値をハーモニックチェインの数と考えることができる．しかしながら，n の値に division-2 タスクが含まれていないことを考慮しなければならない．すなわち，各式における $n = 1$ は，ハーモニックタスクセットのスケジュール可能上限を示しているわけではない．本節では，RMd2 のハーモニックタスクセットに対するスケジュール可能上限が 100% であることを証明する．

すでに述べたように，プロセッサ P_j 上のタスクセット Λ_j に division-2 タスクが存在しなければ，RMd2 による Λ_j のスケジュール可能上限は式 (1) で求められる．すなわち，$n = 1$ としてその上限は 100% となる．次に，division-2 タスクが存在する場合について考える．ある division-2 タスク τ_i'' の実行が，その division-1 タスクの実行によって時刻 t に中断され，その後 division-1 タスクが終了したときに時刻 s に再開されたとする．この場合，τ_i'' は一時的に $s - t$ 時間だけ RM のスケジュールから遅れることになる一方で，Λ_j に含まれるそのほかのタスクは RM のスケジュールから遅れることはない．その理由を以下に述べる．タスクセットはハーモニックであり，かつ τ_i'' の周期は Λ_j の中で最も短いので，K を正の整数として，つねに $\forall\tau_l \in \Lambda$，$T_l = KT_i$ が成り立つ．よって，τ_i'' のジョブがリリースされた時間に，必ずそのほかのすべてのタスクのジョブ $\{\tau_l \in \Lambda_j\}$ もリリースされる．結果として，τ_i'' が失う $s - t$ 時間を $\{\tau_l \in \Lambda_j\}$ に割り当てることができる．すなわち，$\{\tau_l \in \Lambda_j\}$ は RM のスケジュールよりも早く実行される可能性はあるが，遅れて実行されることは決してない．それゆえ，τ_i'' がデッドラインまでに実行を完了することができれば，$U(\Lambda_j) = 1$ であっても Λ_j はつねにスケジュール可能となる．τ_i' の実行により，τ_i'' は最大で C_i' 時間だけスケジュールから遅れることになるので，

☆ 周期が互いに整数倍のタスクをまとめて 1 つのタスクと見なした場合のタスク数．

$C'_i + C''_i \leq T_i$ が満たされれば τ''_i はデッドラインを守れることが保証される．$C'_i + C''_i = C_i \leq T_i$ であることから，この条件はつねに成立する．よって，ハーモニックタスクセットに対する RMd2 のスケジュール可能上限は 100% である．

6. シミュレーションによる評価

本章では，RMd2-SIP が大きなオーバヘッドを要することなく高いスケジュール可能性を達成できることを示す．スケジュール可能性に関する評価指標は文献 2) でも用いられているスケジュール成功率（Success Ratio）☆とする．また，オーバヘッドに関する評価指標はプリエンプション数とする．比較対象のアルゴリズムとしては，グローバルスケジューリング方式である RM および RM-US，パーティショニング方式である RM-FF および RM-FFDU とする．RM はシングルプロセッサでのアルゴリズムをマルチプロセッサに応用したアルゴリズムである．RM-US は CPU 使用率が $M/(3M-2)$ 以上のタスクに最高優先度を与えて，そのほかのタスクは RM に従ってスケジュールするアルゴリズムである．RM-FF および RM-FFDU は，各々 First-Fit（FF）および FF in Decreasing Utilization（FFDU）[19] に基づいて各プロセッサにタスクを割り当て，プロセッサごとに RM に従ってスケジュールするアルゴリズムである．

6.1 評価環境

本論文では，一般的なタスクセットとハーモニックタスクセットの両方に対して RMd2-SIP の有効性を示す．シミュレーションは M，U_{max}，U_{min}，U_{total} の 4 つのパラメータによって実行する．M はプロセッサ数である．U_{max} と U_{min} はタスクセットに含まれるタスクの CPU 使用率の最大値と最小値であり，U_{total} はタスクセットに含まれるタスクの CPU 使用率の合計である．システム使用率は U_{total}/M である．(M，U_{max}，U_{min}) の組合せに対して，システム使用率 30% から 100% まで各々1,000 個のタスクセットを投入してスケジュール成功率を計測した．これらのパラメータに関しては様々な組合せが考えられるが，既存の組み込み用プロセッサの規模を考慮して，プロセッサ数は $M = 2$，$M = 4$，$M = 8$ の 3 通りとした．タスクの CPU 使用率の範囲は，我々の研究室で開発を行っているヒューマノイドロボット[23] のアプリケーションに基づいて設定した．本ロボットでは，単純な行動しか必要ない場合には CPU 使用率の低いタスクのみで実現される．一方で，精度の高い行動が必要になると CPU 使用率の高いタスクも必要になる．よって，CPU 使用率の低いタスクしか含まない場合を想定した $(U_{max}, U_{min}) = (0.1, 0.01)$ および CPU 使用率の高いタスクも含む場合を想定した $(U_{max}, U_{min}) = (1.0, 0.01)$ の 2 通りを用意した．タスクセット Γ は以下のように生成した．$U(\Gamma) \leq U_{total}$ である限り，新しいタスクを Γ に追加する．ヒューマノイドロボットには様々な動作が存在するので，タスクの CPU 使用率も様々である．よって，各タスクの CPU 使用率は文献 1) と同様に $[U_{max}, U_{min}]$ の範囲で一様分布によって決定することにした．最後に生成されるタスクに関してのみ，$U(\Gamma) = U_{total}$ となるように調節した．各タスクの周期に関しては，想定するヒューマノイドロボットに実装されるアプリケーションの周期が 1〜30 ms 程度であるため，タスクセットが一般タスクセットの場合には $[100, 3000]$ の範囲で無作為に選択し，ハーモニックである場合には $\{100, 200, 400, 800, 1600, 3200\}$ から無作為に選択した．実行時間は $C_i = U_i T_i$ として得られる．シミュレーション時間は $[0, \max\{lcm(\{T_i \mid \tau_i \in \Gamma\}), 2^{32}\})$ とした．

タスクセットのスケジュール成功の定義はスケジューリングアルゴリズムによって異なる．ポーションニング方式の RMd2-SIP およびパーティショニング方式の RM-FF/RM-FFDU は，タスク割当てに成功したらデッドラインミスを起こすことなくスケジュールできるように設計されている．よって，これらのアルゴリズムでは，タスクセットに含まれる全タスクをプロセッサに割り当てることができればスケジュール成功であると定義した．スケジュール可能性判定におけるプロセッサごとのスケジュール可能上限として，RMd2-SIP は式 (12) を利用し，RM-FF および RM-FFDU は式 (1) を利用した．公平のため，一般タスクセットを対象としたシミュレーションでは，RMd2-SIP，RM-FF および RM-FFDU のスケジュール可能上限を求める際にはハーモニックチェインは考慮にいれないことにした．すなわち，式 (12) および式 (1) における n は実際のタスク数を指す．ここで，これらの式にはタスク数 n が含まれるため，タスクを 1 つ追加するごとにスケジュール可能かどうかを判定しなくてはならない．実際のシステムでは，この再計算のオーバヘッドが問題となることもあり，しばしば $n \to \infty$ としてスケジュール可能性判定を行うことがある．そこで，RMd2-SIP に対しては式 (12) の代わりに式 (13) を利用するバージョンを用意し，シミュ

☆ Success Ratio = $\frac{\#\text{ of successfully scheduled task sets}}{\#\text{ of scheduled task sets}}$

レーションではRMd2-SIP-INFと表記した．RM-FFおよびRM-FFDUに関しても同様に，式(1)において$n \to \infty$として$\log 2 = 69\%$を上限とするバージョンを用意し，シミュレーションでは各々RM-FF-INFおよびRM-FFDU-INFと表記した．一方，RMおよびRM-USに関しては，実際にタスクセットに含まれる全タスクをスケジュールし，デッドラインミスを起こすことなくシミュレーションを終了できた場合に，そのタスクセットはスケジュール成功であると定義した．RMおよびRM-USにおけるタスクセットのスケジュール成功を上記のように定義した理由は以下のとおりである．RM-USに対する理論的なシステムのスケジュール可能上限は$M/(3M-2)$とされているが，実際にはこれよりも高いシステム使用率でスケジュール可能なことが報告されている[2]．RMに関しても，Dhall's effect[9]と呼ばれる特殊な状況が発生するためにシステムのスケジュール可能上限は$1/M$とされているが，実際にはそれよりも高いシステム使用率でスケジュール可能である．そのため，RMおよびRM-USに関しては，実際にタスクをスケジュールしてスケジュール成功率を計測することとした．

6.2 スケジュール可能性に関する評価

図9は，タスクセットはハーモニックではない場合の各アルゴリズムのスケジュール成功率を示している．まず，$U_{max} = 0.1$の場合について考察する．この場合，$U_{max} \leq M/(3M-2)$なのでRM-USの動作はRMと同じであり，これら2つのスケジュール成功率は非常に高かった．これは，CPU使用率の低いタスクしか存在しなかったのでDhall's effectが発生しなかったためである．しかしながら，プロセッサ数Mの増加にともない，RMのスケジュール成功率は減少していき，$M = 8$の場合のスケジュール成功率はRMd2-SIPとほとんど同じになった．シミュレーションではRMのスケジュール成功率は高かったが，CPU使用率の低いタスクしか存在しない環境では，RMに対するシステムのスケジュール可能上限はたかだか$M/(3M-2)$であることに注意されたい．すなわち，$M = 2$，$M = 4$，$M = 8$のとき，システム使用率は各々50%，40%，36%までしかスケジュール可能であることが保証できない．対照的に，そのほかのアルゴリズムのスケジュール成功率は理論的にもMに依存しないため，ほぼ一定であることが確認できた．それらのアルゴリズムのスケジュール成功率はほとんど同じであったが，RMd2-SIPがわずかに高いスケジュール成功率を示した．RMd2-SIPは，Mの値に関係なくシステム使用率70%までは100%のスケジュール成功率を達成できた．一方で，RM-FFおよびRM-FFDUはシステム使用率65〜67%付近でRMd2-SIP-INFよりも早い段階でスケジュール成功率が100%を下回った．

次に，$U_{max} = 1.0$の場合について議論する．RMd2-SIPは，$M = 2$，$M = 4$，$M = 8$のとき，各々67%，70%，72%のシステム使用率まで100%のスケジュール成功率を達成できた．$n \to \infty$の場合でさえも，RMd2-SIP-INFはMの値に関係なく65%のシステム使用率まで100%のスケジュール成功率を達成できた．これらのシステム使用率はRM-FFDUを除くすべてのアルゴリズムよりも高いものであった．以上の結果からRMd2-SIPはプロセッサ数やタスクのCPU使用率の範囲によらず安定して高いスケジュール可能性を達成できるアルゴリズムであるといえる．一方で，RM-FFは，$M = 2$，$M = 4$，$M = 8$のとき，システム使用率が各々45%，50%，52%を超えるとスケジュール成功率が100%を下回った．RM-FFDUは，$M = 2$，$M = 4$，$M = 8$のとき，システム使用率が各々60%，57%，60%を超えるとスケジュール成功率が100%を下回った．さらに，RM-FFおよびRM-FFDUでは，$n \to \infty$の場合にスケジュール可能性が劇的に低下してしまった．実際，システム使用率が47%に達するとスケジュール成功率は100%を下回ってしまい，とりわけ$M = 2$の場合にはシステム使用率35%の時点でスケジュール成功率は100%を下回ってしまった．このようなスケジュール可能性の低下は以下のことに起因すると考えられる．CPU使用率の高いタスクが存在すると，各プロセッサはすぐに埋まってしまい，割り当てられるタスク数nは小さくなる傾向にある．式(1)より，nが小さいとプロセッサごとのスケジュール可能上限は大きくなり，結果として，実際のnの値を用いた場合と$n \to \infty$とした場合のスケジュール可能性に差が生じたのだと考えられる．この結果から，$n \to \infty$の場合に，タスクを分割することの効果が最も顕著に現れるといえる．タスク分割不可能なRM-FFやRM-FFDUでは，タスク割当てフェーズにおいてCPU使用率の高いタスクの割当てに失敗したことがしばしば確認できた．これにより，RM-FFおよびRM-FFDUのスケジュール可能性は低下してしまったのだと考えられる．RMは，$M = 2$，$M = 4$，$M = 8$のとき，システム使用率が各々60%，47%，42%を超えるとスケジュール成功率は100%を下回ってしまったが，RM-USに比べるとかなり高い性能を示した．この結果は，文献2)で報告されている結果と異なる．これは，文献2)ではタ

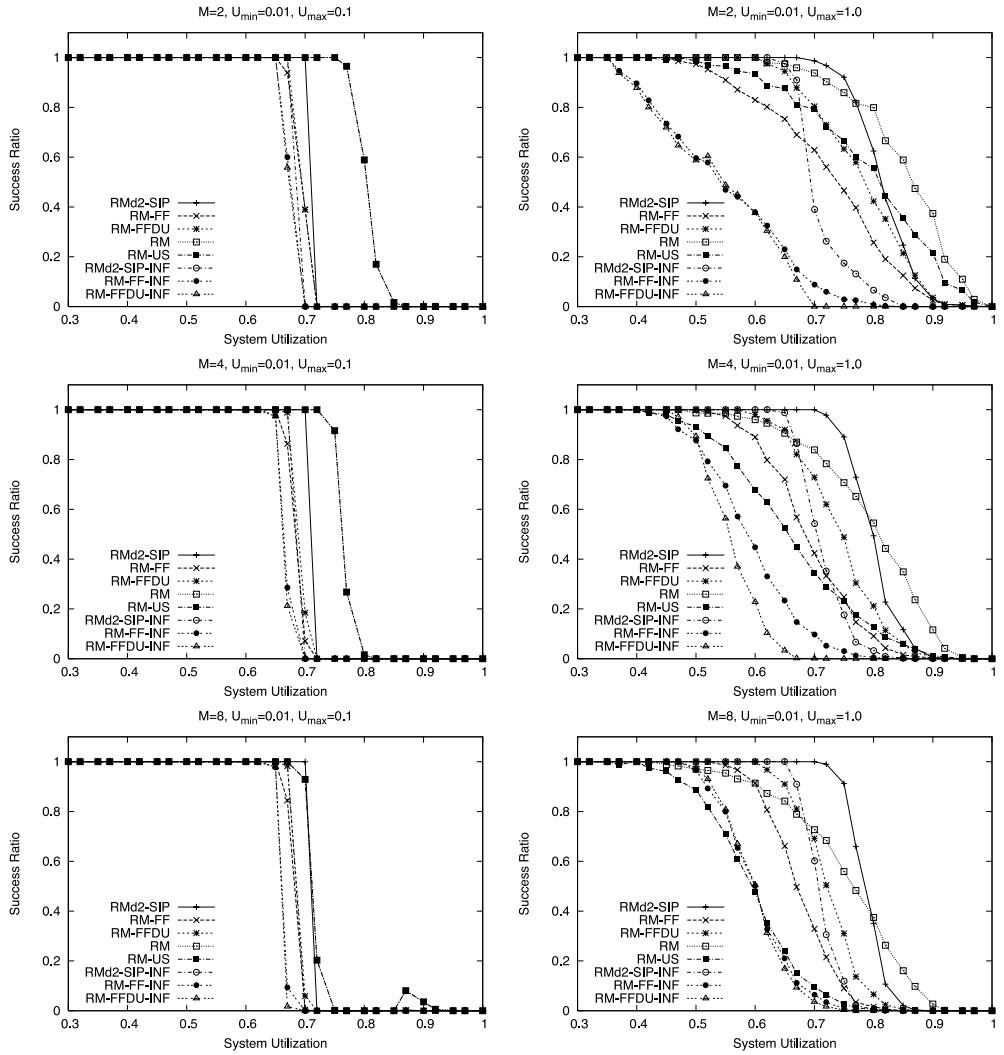

図 9 一般タスクセットにおけるシステム使用率に応じたスケジュール成功率
Fig. 9 Success ratio as a function of system utilization with generic task sets.

スクの CPU 使用率の決定に 2 項分布を利用していたが，本論文では一様分布を利用したため，生成されたタスクセットの性質が異なり，結果としてスケジュール成功率にも違い生じたのだと考えられる．本論文のシミュレーションでは，RM の方が RM-US よりも優れた性能を発揮したが，CPU 使用率の高いタスクが存在する環境では，RM に対する理論的なシステムのスケジュール可能上限はたかだか $1/M$ であることに注意されたい．すなわち，$M = 2$，$M = 4$，$M = 8$ のとき，システム使用率は各々 50%，25%，12.5% までしかスケジュール可能であることを保証できない．

100% のスケジュール成功率を達成できた最大のシステム使用率の観点では，RMd2-SIP はそのほかのアルゴリズムよりも優れていたが，スケジュール成功率自体はシステム使用率が高くなるに従ってそのほかのアルゴリズムよりも低くなってしまった．この事実から，RMd2-SIP のスケジュール可能性は，システム使用率がある境界を超えてしまうと急激に低下してしまう性質を持つといえる．しかしながら，実時間システムの価値はシステムがデッドラインミスを発生し始めた瞬間に 0 になる，または急激に低下してしまうので，RMd2-SIP が他のアルゴリズムよりも高いシステム使用率で 100% のスケジュール成功率を達成できたという事実は非常に重要であるといえる．結果として，一般タスクセットに対しては，100% のスケジュール成功率を達成できるシステム使用率の観点で，RMd2-SIP は RM-FF，RM-FFDU，RM，RM-US に比べて，最大で各々 22%，13%，30%，37% ほど性能を向上さ

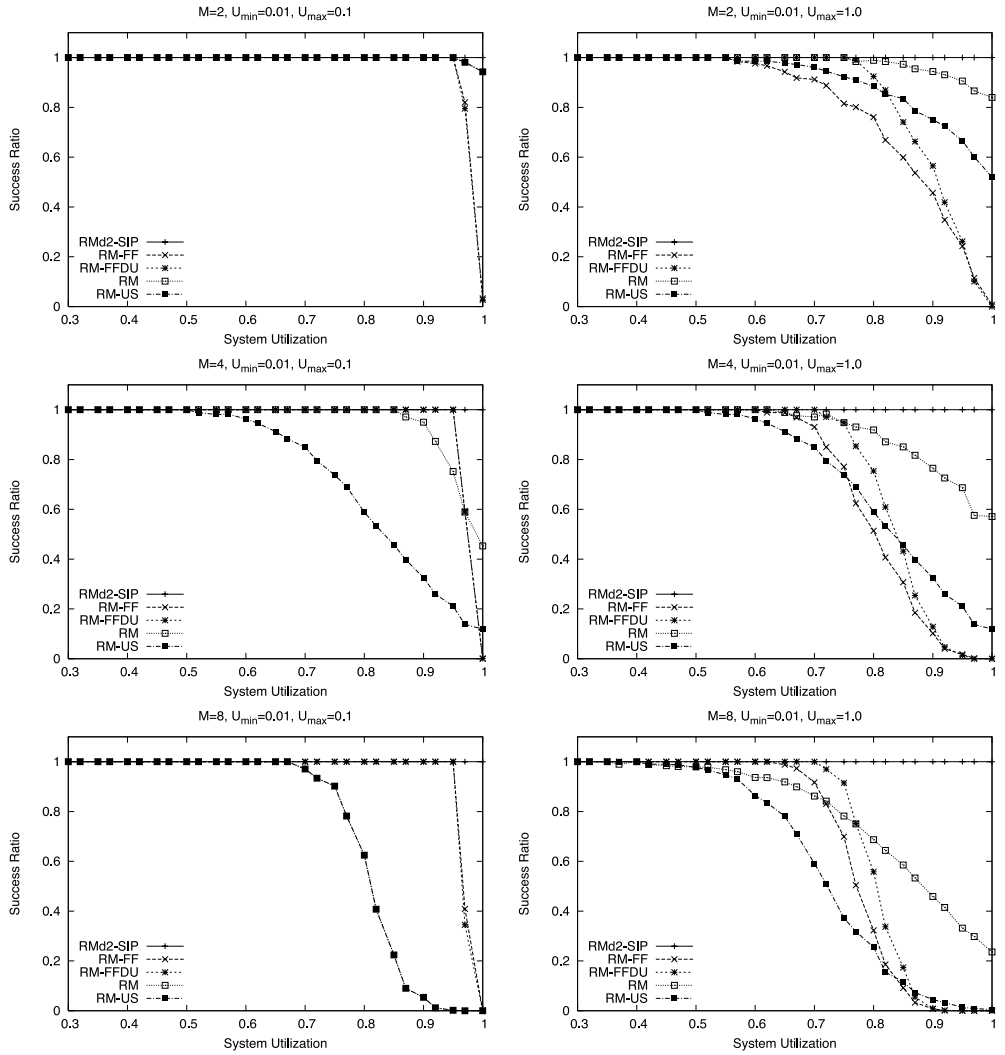

図 10 ハーモニックタスクセットにおけるシステム使用率に応じたスケジュール成功率
Fig. 10 Success ratio as a function of system utilization with harmonic task sets.

せることができた．

最後に，タスクセットがハーモニックである場合の各アルゴリズムのスケジュール成功率を**図 10** に示す．5 章で示したように，RMd2-SIP はシステム使用率に関係なくつねに 100% のケジュール成功率を達成できた．RM-FF および RM-FFDU は RMd2-SIP には劣ったものの，RM および RM-US に比べると比較的良い性能を発揮した．RM および RM-US に対する RM-FF および RM-FFDU の優位性は，タスク割当てフェーズにおいてプロセッサごとのスケジュール可能上限を 100% に設定できることに起因する．それでも，とりわけ $U_{max} = 1.0$ の場合には，RM-FF および RM-FFDU のスケジュール成功率は RMd2-SIP に大きく劣っていた．実際，RM-FF は，$M = 2$，$M = 4$，$M = 8$ のとき，システム使用率が各々 57%，65%，62% の時点でスケジュール成功率は 100% を下回った．また，RM-FFDU は，システム使用率が各々 75%，70%，70% の時点でスケジュール成功率は 100% を下回った．RM および RM-US に関しては，M が増加するにつれて，スケジュール成功率は劇的に低下してしまった．とりわけ，$M = 8$ の場合に，RM はシステム使用率が 35% の時点でスケジュール成功率が 100% を下回り，RM-US はシステム使用率が 40% の時点でスケジュール成功率が 100% を下回った．結果として，与えられたタスクセットがハーモニックであった場合には，100% のスケジュール成功率を達成できるシステム使用率の観点で，RMd2-SIP は RM-FF，RM-FFDU，RM，RM-US に比べて，最

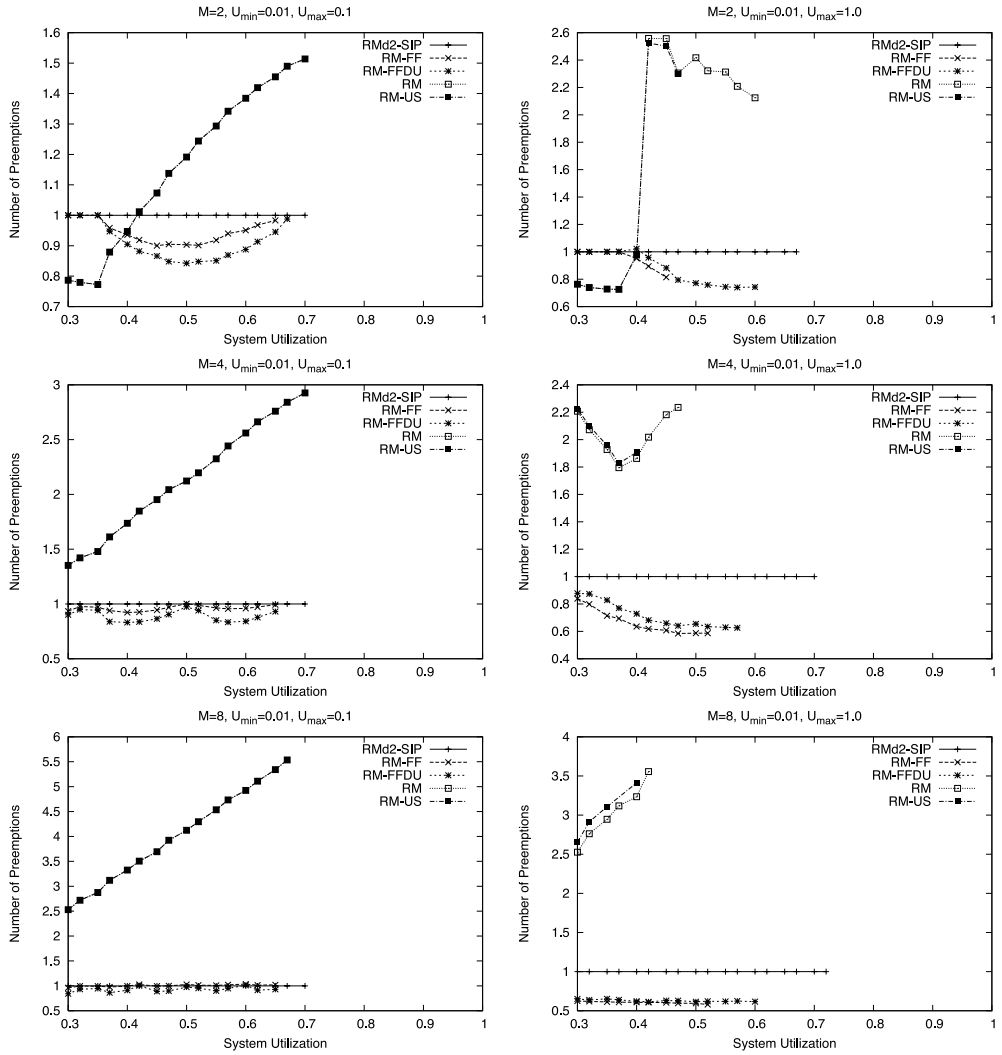

図 11 一般タスクセットにおけるシステム使用率に応じたプリエンプション数
Fig. 11 Number of preemptions as a function of system utilization with generic task sets.

大で各々 45%，30%，65%，60% ほど性能を向上させることができた．

6.3 オーバヘッドに関する評価

図 11 は，一般タスクセットをスケジュールした際の各アルゴリズムの RMd2-SIP に対する相対的なプリエンプション数を示している．RMd2-SIP-INF，RM-FF-INF および RM-FFDU-INF は，$n \to \infty$ によってスケジュール可能性判定式の評価が異なるだけでアルゴリズム自体に違いはないため，本評価では対象外とした．また，ハーモニックタスクセットをスケジュールした際の各アルゴリズムの絶対プリエンプション数は一般タスクセットをスケジュールした際と異なったが，RMd2-SIP に対する相対的なプリエンプション数はあまり変わらなかったため，本評価では省略した．

まず，$U_{max} = 0.1$ の場合について議論する．この場合，RMd2-SIP において分割されたタスクの CPU 使用率も最大で 0.1 なので，division-1 タスクと division-2 タスクのスケジュールが重なる時間も短かった．よって，$U_{max} = 0.1$ の場合においてはタスクを分割することによるプリエンプション数の増加は大きくはなかった．しかしながら，各アルゴリズムのプリエンプション数には違いが確認された．$M = 2$ の場合，システム使用率が 35% までは RMd2-SIP，RM-FF および RM-FFDU もすべてのタスクをプロセッサ P_1 に割り当てることができたので，プリエンプション数は同じであった．システム使用率が 35% を超えると，RM-FF および RM-FFDU はプロセッサ P_2 も利用する機会が多くなり，タスクが 2 つのプロ

セッサに分散される傾向にあった．一方で，RMd2-SIP は必ずスケジュール可能上限までプロセッサを利用するので，システム使用率が低いうちはタスクが P_1 に偏る傾向にあった．結果として，RMd2-SIP の方がプリエンプション数は増加してしまった．システム使用率が高くなると P_1 と P_2 のプロセッサ使用率も差がなくなるため，プリエンプション数の差も徐々になくなっていった．RM-FF および RM-FFDU では，最終的にシステム使用率 65% 付近で P_2 のスケジュール可能上限に達し，プリエンプション数は RMd2-SIP とほぼ等しくなった．グローバルスケジューリング方式である RM および RM-US はシステム使用率に関係なくすべてのプロセッサを利用する．よって，システム使用率が低いうちはタスクが 2 つのプロセッサに分散されプリエンプション数は少ないが，システム使用率が高くなるとプリエンプション数は劇的に増加してしまう傾向にあった．$M = 4$ および $M = 8$ の場合にも同じ傾向が確認できた．RM-FF および RM-FFDU では，各プロセッサの使用率がスケジュール可能上限に達するたびに，プリエンプション数は RMd2-SIP とほぼ等しくなった．RMd2-SIP のプリエンプション数は最大で RM-FF の約 1.1 倍および RM-FFDU の約 1.2 倍であったが，RM および RM-US に比べると最大で約 0.2 倍に抑えることができた．

$U_{max} = 1.0$ の場合は，RMd2-SIP のプリエンプション数と RM-FF および RM-FFDU のプリエンプション数の差は $U_{max} = 0.1$ の場合よりも大きくなったことが確認できた．各タスクの CPU 使用率が高いと各タスクの実行時間も長くなる傾向にあり，RMd2-SIP において分割された division-1 タスクと division-2 タスクのスケジュール時刻が重なりやすくなる．結果として，RM-FF および RM-FFDU では発生することない division-1 タスクと division-2 タスクのプリエンプションが多くなったのだと考えられる．また，その差はプロセッサ数が増えるごとに大きくなっていった．$M = 8$ の場合，RMd2-SIP のプリエンプション数は最大で RM-FF の約 1.7 倍および RM-FFDU の約 1.6 倍であった．しかしながら，RM および RM-US に比べると最大で約 0.3 倍に抑えることができた．

6.2 節の評価では，RMd2-SIP はスケジュール可能性の観点でそのほかのアルゴリズムよりも高い性能を発揮することを示した．よって，RMd2-SIP ではプリエンプション数が問題となることが分かる．絶対的なプリエンプション数はしばしばタスク数に依存し，プリエンプションのコストはシステムのプラットフォームに大きく依存する．よって，処理性能の高いプラットフォームにおいて少ないタスク数をスケジュールする場合に RMd2-SIP は特に効果的であるといえる．

7. 結　　論

本論文では，ポーショニング方式に基づいた固定優先度スケジューリングアルゴリズム RMd2-SIP を提案した．スケジュール可能性解析では，RMd2-SIP に対するシステムのスケジュール可能上限が，一般タスクセットに対しては 50% であり，ハーモニックタスクセットに対しては 100% であることを証明した．また，スケジュール可能性判定についても述べた．シミュレーションによる評価では，実際のスケジュール可能性の観点で，RMd2-SIP が既存のアルゴリズムよりも高い性能を発揮することを示した．また，そのトレードオフとして，RMd2-SIP では RM-FF および RM-FFDU よりも 1.6〜1.7 倍のプリエンプションが発生してしまうことを示した．一方で，RM および RM-US に比べると最大で約 0.2 ∼ 0.3 倍に抑制できたことから，プリエンプション数を大きく増加させることなくスケジュール可能性を向上させることができたといえる．

今後の課題を示す．まず，タスク割当て時に，分割するタスクの選択方法について考える．式 (12) より，RMd2-SIP のスケジュール可能上限は分割するタスクの周期や CPU 使用率に大きく依存する．そのため，分割するタスクを効率的に選択することでスケジュール可能性を向上させることができると考えられる．また，本論文では各プロセッサにおいて 1 つのタスクのみを分割するアプローチを採用したが，各プロセッサにおいて複数のタスクを分割する複雑なアプローチに関しても考える．さらに，4.2 節でも述べたように，スケジューリングの際のプリエンプションの数を削減できるアルゴリズムや効率的な実装についても考える．

謝辞　本研究は，科学技術振興機構 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) Anderson, J., Bud, V. and Devi, U.: An EDF-based Scheduling Algorithm for Multiprocessor Soft Real-Time Systems, *Proc. 17th Euromicro Conference on Real-Time Systems*, pp.199–208 (2005).
2) Andersson, B., Baruah, S. and Jansson, J.: Static-priority Scheduling on Multiprocessors, *Proc. 22nd IEEE Real-Time Systems Symposium*, pp.193–202 (2001).

3) Andersson, B. and Tovar, E.: Multiprocessor Scheduling with Few Preemptions, *Proc. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.322–334 (2006).
4) Baruah, S., Cohen, N., Plaxton, C. and Varvel, D.: Proportionate Progress: A Notion of Fairness in Resource Allocation, *Algorithmica*, Vol.15, pp.600–625 (1996).
5) Bini, E., Buttazzo, G. and Buttazzo, G.: A Hyperbolic Bound for the Rate Monotonic Algorithm, *Proc. 14th Euromicro Conference on Real-Time Systems*, pp.59–66 (2002).
6) Calandrino, J., Leontyev, H., Block, A., Devi, U. and Anderson, J.: LITMUS^{RT}: A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers, *Proc. 27th IEEE Real-Time Systems Symposium*, pp.111–123 (2006).
7) Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J. and Baruah, S.: A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms, *Handbook of SCHEDULING Algorithms, Models and Performance Analysis*, pp.30.1–30.19, CHAPMAN & HALL/CRC (2004).
8) Cho, H., Ravindran, B. and Jensen, E.: An Optimal Real-Time Scheduling Algorithm for Multiprocessors, *Proc. 27th IEEE Real-Time Systems Symposium*, pp.101–110 (2006).
9) Dhall, S.K. and Liu, C.L.: On a Real-Time Scheduling Problem, *Operations Research*, Vol.26, No.1, pp.127–140 (1978).
10) Goosens, J., Funk, S. and Baruah, S.: Priority-driven Scheduling of Periodic Task Systems on Multiprocessors, *Real-Time Systems*, Vol.25, pp.187–205 (2003).
11) Kuo, T., Chang, L., Liu, Y. and Lin, K.: Efficient On-Line Schedulability Tests for Real-Time Systems, *IEEE Trans. Softw. Eng.*, Vol.29, No.8, pp.734–751 (2003).
12) Kuo, T. and Mok, A.: Load Adjustment in Adaptive Real-Time Systems, *Proc. 12th IEEE Real-Time Systems Symposium*, pp.160–171 (1991).
13) Lauzac, S., Melhem, R. and Mosses, D.: An Improved Rate Monotonic Admission Control and Its Applications, *IEEE Trans. Comput.*, Vol.52, No.3, pp.337–350 (2003).
14) Liu, C.L. and Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *J. ACM*, Vol.20, No.1, pp.46–61 (1973).
15) Lopez, J., Diaz, J. and Garcia, D.: Minimum and Maximum Utilization Bounds for Multiprocessor Rate-Monotonic Scheduling, *IEEE Trans. Parallel and Distributed Systems*, Vol.15, No.7, pp.642–653 (2004).
16) Lopez, J., Diaz, J. and Garcia, D.: Utlization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems, *Real-Time Systems*, Vol.28, pp.39–68 (2004).
17) Lu, W., Wei, H. and Lin, K.: Rate Monotonic Schedulability Conditions Using Relative Period Ratios, *Proc. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.3–9 (2006).
18) Oh, D. and Baker, T.: Utlization Bounds for N-Processor Rate Monotonic Scheduling with Static Processor Assignment, *Real-Time Systems*, Vol.15, No.2, pp.183–192 (1998).
19) Oh, Y. and Son, S.: Allocating Fixed-Priority Periodic Tasks on Multiprocessor Systems, *Real-Time Systems*, Vol.9, No.3, pp.207–239 (1995).
20) Olukotun, K., Nayfe, B., Hammond, L., Wilson, K. and Chang, K.: The Case for a Single-Chip Multiprocessor, *Proc. 7th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.2–11 (1996).
21) Spracklen, L. and Abraham, S.: Chip Multithreading: Opportunities and Challenges, *Proc. 11th IEEE International Symposium on High-Performance Computer Architecture*, pp.248–252 (2005).
22) Strosnider, J., Lehoczky, J. and Sha, L.: The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments, *IEEE Trans. Comput.*, Vol.44, No.1, pp.73–91 (1995).
23) 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).
24) Tullsen, D., Eggers, S., Levy, H., Lo, J. and Stamm, R.: Exploiting Choice: Instruction Fetch and Number on an Implementable Simultaneous Multithreading Processor, *Proc. 23rd Annual International Symposium on Computer Architecture*, pp.191–202 (1996).

(平成 19 年 1 月 22 日受付)
(平成 19 年 6 月 6 日採録)

加藤　真平（学生会員）

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．現在同博士課程に在籍．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．