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

2008－SLDM－134／2008－EMB－8（33）
2008 / 3 / 28

マルチプロセッサにおけるグローバルRMに基づくリアルタイムスケジューリングアルゴリズム

武田　瑛†　船岡　健司†　加藤　真平†　山崎　信行†

†慶應義塾大学大学院理工学研究科開放環境科学専攻
〒223–8522 横浜市港北区日吉 3–14–1
E-mail: †{takeda,funaoka,shinpei,yamasaki}@ny.ics.keio.ac.jp

あらまし　近年，組込みリアルタイムシステムでもマルチプロセッサ技術の利用が一般的になりつつある．このような背景から，マルチプロセッサにおいてCPUを100%利用できる最適なリアルタイムスケジューリングアルゴリズムが提案されているが，多くのコンテキストスイッチやタスクマイグレーションが発生し，それらのオーバーヘッドにより実用性の面で問題視されている．一方で，従来の単純なアルゴリズムでは高いスケジュール可能性を実現することができない．本論文では，従来の単純なアルゴリズムであるグローバルRate Monotonic (RM)を基にしたスケジューリングアルゴリズムRMZLを提案する．提案するアルゴリズムは，RMの長所を残しつつ，スケジュール可能性を向上させるものである．シミュレーション評価により，提案アルゴリズムは従来のグローバルRMを基にしたスケジューリングアルゴリズムよりも多くのタスクをスケジュール可能であることを示す．
キーワード　リアルタイムシステム，マルチプロセッサシステム，グローバルRM，スケジュール可能性解析

Global-RM based Real-Time Scheduling Algorithm on Multiprocessors

Akira TAKEDA†, Kenji FUNAOKA†, Shinpei KATO†, and Nobuyuki YAMASAKI†

† 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: †{takeda,funaoka,shinpei,yamasaki}@ny.ics.keio.ac.jp

Abstract　In recent embedded systems multiprocessor platforms are commonly used. Due to this background, optimal real-time scheduling algorithms which can use full system utilization have been proposed, but these algorithms generate a number of context switches and task migrations that incur significant overhead and are often considered not to be practical due to the overhead. Meanwhile existing simple algorithms cannot improve the schedulability. This paper propose a new multiprocessor real-time scheduling algorithm based on global Rate Monotonic (RM) which is one of simple conventional algorithms. Our algorithm remains the merit of RM and also improves the schedulability. The simulation evaluation shows that our algorithm outperforms the existing global RM based algorithm in the schedulabiltiy point of view.
Key words　Real-Time Systems, Multiprocessor Systems, Global RM, Schedulability Analysis

1. はじめに

今日の組込みリアルタイムシステムは，ロボットやユビキタスアプリケーションの出現により，高性能かつ省電力なプラットフォームを必要とする傾向にある．そのため，同時細粒度マルチスレッディング(SMT)[12]やチップマルチプロセッサ(CMP)[11]のような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．よって，このようなマルチプロセッサ環境においてリアルタイム性を保証することが重要となる．しかしながら，マルチプロセッサにおけるリアルタイムスケジューリングは非常に複雑であることが知られている．シングルプロセッサにおける最適なリアルタイムスケジューリングアルゴリズムとしてはRate Monotonic (RM)[10]やEarliest Deadline First (EDF)[10]等があるが，これらはマルチプロセッサにおいてはもはや最適でないことが知られている[9].

近年，マルチプロセッサにおける最適なリアルタイムスケジューリングアルゴリズムが提案されてきている[6][7][2]．しかし，これらのアルゴリズムは，CPUの利用率を100%利用できる代わりに，複雑で非常にオーバーヘッドが高いという短所がある．実用性を考えると，従来のアルゴリズムであるRMや

EDF を基にしたアルゴリズムを用いた方が良いと考えられる．特に RM は静的優先度アルゴリズムであり，動的優先度アルゴリズムである EDF と比べると，利用可能な CPU 利用率では劣るが，実装が容易，予測性が高い，タスクのジッタが小さい等の長所がある．そこで本論文では RM に焦点を当てる．

マルチプロセッサにおけるスケジューリング方式は，タスクを特定のプロセッサに割り当てるパーティショニング方式と，動的に割り当てるグローバル方式の 2 つに分けられる．パーティショニング方式は，タスクの割当て後はシングルプロセッサにおけるスケジューリング理論が適用できるという長所があるが，利用可能なシステム利用率は必ず 50% 以下であることが知られている [1]．また，タスクを静的にプロセッサに割り当ててしまうため，プロセッサの負荷が偏りやすい．一方グローバル方式は，タスクのプロセッサ間の移動によるオーバーヘッドが大きい半面，マルチプロセッサの並列性を最大限利用できるためにタスクの平均反応時間が短いという長所がある．しかしながら，グローバル方式の RM アルゴリズム (以下グローバル RM) は，Dhall's Effect [9] により利用可能な CPU 利用率が非常に低い．プロセッサの数を m とすると，グローバル RM の利用可能なシステム利用率は $1/m$ である．そこで Andersson [3] らは，グローバル RM において $m/(3m-2)$ より利用率が高いタスクを最高優先度にすることにより，利用可能なシステム利用率を $m^2/(3m-2)$ まで改善するアルゴリズム RM-US$[m^2/(3m-2)]$ を提案した．しかしながら，RM-US アルゴリズムは RM がスケジュール可能なタスクセットをスケジュールできないことがあり，結果としてスケジュール可能性の低下を招くことがある．

本論文では，グローバル RM を基にして，RM-US よりさらにタスクのスケジュール可能性を向上させるスケジューリングアルゴリズム RMZL を提案する．そして，ハードリアルタイムシステムにおいて不可欠なスケジュール可能性判定式を示す．最後にシミュレーションにより，提案したアルゴリズムをタスクセットのスケジュール成功率の観点から評価する．

2. システムモデル

本論文では，マルチプロセッサリアルタイムスケジューリングの研究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 個のプロセッサまたはコアから構成されるものとする．そして n 個のタスクから構成されるタスクセット $\tau=\{\tau_1,\tau_2,...,\tau_n\}$ がシステムに与えられる．各タスク τ_i は (C_i,T_i) というタプルで定義される．C_i は最悪実行時間であり，T_i は周期である．タスクは周期の短い順，つまり RM において優先度の高い順にソートされているものとする．つまり，$T_1 \leq T_2 \leq ... \leq T_n$ が成り立つ．τ_i の利用率を $U_i=C_i/T_i$ で表す．また，タスクセット τ に含まれるタスクの合計利用率を $U(\tau)=\sum_i U_i$ で表す．すなわち，$U(\tau)$ はシステム全体の負荷を表す．ここで，$U(\tau)/m$ をシステム利用率と呼ぶことにする．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 $\tau_{i,k}$ と表す．$\tau_{i,k}$ 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，$d_{i,k}=r_{i,k+1}=r_{i,k}+T_k$ とする．時間 t において，ジョブ $\tau_{i,k}$ の残り実行時間を $C_{i,k}(t)$ とする時，$\tau_{i,k}$ の余裕時間 (laxity) を $L_{i,k}(t)=d_{i,k}-t-C_{i,k}(t)$ と定義する．余裕時間はそのジョブの緊急度を表し，緊急度は余裕時間が少なくなる程増す．余裕時間が負であることはデッドラインミスを意味する．ジョブ $\tau_{i,j}$ がレディ状態であるが，すべてのプロセッサが他の優先度の高いジョブによって使用されているとき，ジョブ $\tau_{i,j}$ はそれらのジョブにブロックされているという．タスクはプリエンプト可能で互いに独立しているものとし，如何なるタスクも複数のプロセッサで実行することはできないものとする．また，一度システムの実行が開始したら，新たなタスクが到着したり，既にシステムに存在するタスクが消滅することはないものとする．

3. スケジューリングアルゴリズム

提案アルゴリズム RMZL (Rate Monotonic until Zero Laxity) はゼロ余裕時間ルール (zero laxity rule) [8] [5] を RM に適用したものである．つまり，通常は RM によりジョブをスケジュールし，ジョブの余裕時間が 0 になったらそのジョブを最高優先度にする．RMZL アルゴリズムを図 1 に示す．まず，余裕時間が負であるジョブが存在すると，そのジョブはデッドラインミスを起こしているので，タスクをレディーキューから外す (2-3 行目)．次に，余裕時間が 0 である各ジョブについて，それぞれ以下の処理を行う (5-15 行目)．アイドル状態のプロセッサが存在するならば，そこでタスクを実行する (6-7 行目)．アイドル状態のプロセッサがない場合は，余裕時間が正のジョブがあればそのジョブをプリエンプトする (8-10 行目)．なければ，余裕時間が 0 でかつ自ジョブよりも周期の長いジョブをプリエンプトする (11-13 行目)．これらのいずれにも一致しない場合は，何もしない．結果として，そのジョブは余裕時間が負となり，上で述べたようにレディーキューから外される．すべてのジョブの余裕時間が正ならば，それらは通常の RM によってスケジュールされる (16-18 行目)．

図 2 は 2 プロセッサ上でタスクセット $\tau=\{\tau_1=(2,3),\tau_2=(2,3),\tau_3=(2,3)\}$ を RM と RMZL でスケジュールした例である．すべてのタスクの周期は等しいので，RM スケジュールと RMZL スケジュール共に τ_1，τ_2 が実行を開始する．RM スケジュールでは，$t=2$ まで τ_1 と τ_2 が実行してしまうため，τ_3 はデッドラインミスを起こしてしまう．しかしながら RMZL スケジュールでは，$t=1$ において τ_3 の余裕時間が 0 となり，最高優先度となる．よって τ_2 はプリエンプトされ，τ_3 が実行を開始する．$t=2$ では τ_1 が実行を終了するので，τ_2 が実行を再開する．結果として，RMZL スケジュール例では全てのタスクがデッドラインミスすることなく実行できていることがわかる．

RMZL は，RM と同様に work conserving である．すなわち，実行されていないレディージョブがある限りプロセッサはアイドル状態にならない．また，ゼロ余裕時間ルールは安全 (safety) ルールである．つまり，RM と RMZL のスケジュールが異なってくるのは RM がデッドラインミスを起こす場合のみであり，RM でスケジュール可能なタスクセットは RM と RMZL では全く同じようにスケジュールされる．この性質から，RMZL は，RM でスケジュールできない場合を除いては，予測性が高い，

```
1.  while TRUE do
2.    if there is any job with the negative laxity then
3.      remove it from the ready queue;
4.    end if
5.    while there is any job τ_{i,j} with zero-laxity do
6.      if there is an idle processor then
7.        execute the zero-laxity job on it;
8.      else if there is any job with a positive laxity then
9.        preempt the job with a positive laxity
10.         and the largest period;
11.     else if there is any other job τ_{k,l} with zero laxity
12.      and larger period than τ_{i,j} then
13.       preempt τ_{k,l}
14.     end if
15.   end while
16.   if every job has a positive laxity then
17.     perform RM
18.   end if
19. end while
```

図1 RMZL アルゴリズム

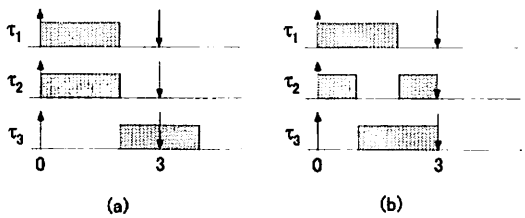

図2 (a) RM スケジュールと (b) RMZL スケジュールの例

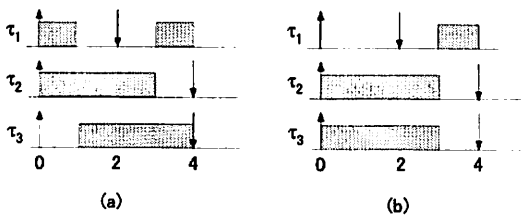

図3 (a) RM スケジュールと (b) RM-US スケジュールの例

タスクのジッタが小さいなどの RM の長所を併せ持つアルゴリズムである．一方，従来のグローバル RM アルゴリズムである RM-US は，RM でスケジュール可能なタスクセットがスケジュール不能になる場合がある．この例として RM 及び RM-US によるタスクセット $\tau=\{\tau_1=(1,2),\tau_2=(3,4),\tau_3=(3,4)\}$ のスケジュールを図 3 に示す．RM スケジュールでは周期の短い τ_1 が先にスケジュールされ，結果としてデッドラインをミスすることなく実行できるが，RM-US スケジュールでは利用率の高いタスクである τ_2 と τ_3 が先に実行されてしまい，結果 τ_1 がデッドラインミスを起こしていることが分かる．

4. スケジュール可能性解析

本章では，RMZL のスケジュール可能性判定条件について述べる．以下の解析は DM (Deadline Monotonic) のスケジュール可能性解析 [4] 及び EDZL (Earliest Deadline until Zero Laxity) のスケジュール可能性解析 [5] を基にしている．

ジョブがデッドラインミスする必要条件を解析するために，与えられたスケジュールで何らかのジョブがデッドラインミスする最初の点に注目する．そのジョブを問題ジョブ (problem job)，そのタスクを問題タスク (problem task) と呼ぶ．問題タスクを τ_k，問題ジョブを $\tau_{k,j}$ で表す．$\bar{t}$ をデッドラインミス時の $\tau_{k,j}$ のデッドライン，$[\bar{t}-T_k,\bar{t})$ を問題ウィンドウ (problem window) とする．

解析のために以下の用語を定義する．

定義 1 (競合量)．区間 $[t,t+\Delta)$ でのタスク $\tau_i(i\neq k)$ の競合量 $W_i^k(t,t+\Delta)$ とは，その区間においてタスク τ_i のジョブがジョブ $\tau_{k,j}$ をブロックしている時間の合計を表す．

定義 2 (競合負荷)．区間 $[t,t+\Delta)$ での競合負荷は，その区間での競合量の合計を W^k とすると，W^k/Δ と定義される．

解析の流れを以下に示す．まず問題ジョブの余裕時間が 0 または負となり得るための問題ウィンドウの競合負荷の下限値を計算する．次に問題タスクをブロックするタスクの競合量の上限値を計算する．すべてのタスクにおける競合量の上限値から求めた競合負荷が最初に求めた下限値より小さいならば，ジョブはデッドラインミスを起こさない．この条件がスケジュール可能性判定条件となる．

補題 1. m 個のプロセッサ上で，タスクセット $\tau=\{\tau_1,\ldots,\tau_n\}$ を *RMZL* でスケジュールすると，デッドライン $\bar{t}$ を持つ問題ジョブ $\tau_{k,j}$ の余裕時間は，以下の条件式を満たせば 0 となり，そのうち不等号 $>$ を満たせば負となる可能性がある．

$$\sum_{i\neq k} W_i^k(\bar{t}-T_k,\bar{t})/T_k \geq m(1-U_k) \qquad (1)$$

ただし，$W_i^k(\bar{t}-T_k,\bar{t})$ は区間 $[\bar{t}-T_k,\bar{t})$ におけるタスク τ_i の競合量である．

証明. 図 4 から分かるように，問題ジョブ $\tau_{k,j}$ の余裕時間が 0 となるのは，それより優先度の高いまたは等しいタスクがすべてのプロセッサを少なくとも (T_k-C_k) 時間占有した場合のみである (斜線部分は τ_k が実行されている部分であり，点線の長方形は τ_k をブロックしているタスクが実行されている部分)．よって，下式が成り立つ時，$\tau_{k,j}$ はゼロ余裕時間となり得る．

$$\sum_{i\neq k} W_i^k(\bar{t}-T_k,\bar{t}) \geq m(T_k-C_k) \qquad (2)$$

上の式は両辺を T_k で割ることにより下式に変形できる．

$$\sum_{i\neq k} W_i^k(\bar{t}-T_k,\bar{t})/T_k \geq m(1-U_k) \qquad (3)$$

よって，補題が成り立つ． □

補題 2. m 個のプロセッサ上で，タスクセット $\tau=\{\tau_1,\ldots,\tau_n\}$ を *RMZL* でスケジュールすると，区間 $[\bar{t}-T_k,\bar{t})$ におけるタスク τ_i

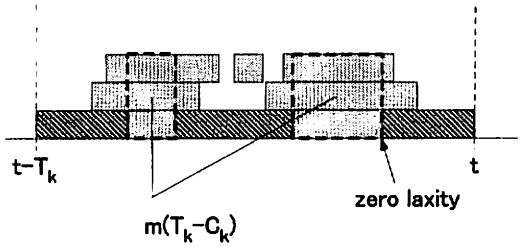

図4 問題ウィンドウとなるスケジュール

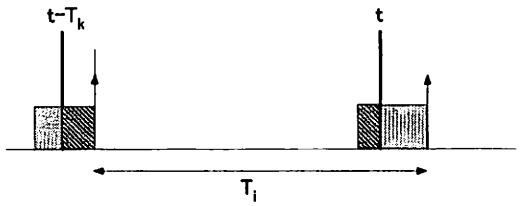

図5 $i>k$ の場合

の競合量 $W_i^k(\bar{t}-T_k,\bar{t})$ は下式を満たす.

$$W_i^k(\bar{t}-T_k,\bar{t}) \leqq \begin{cases} n_iC_i+\min(C_i,\max(0,T_k-((n_i-1)T_i+C_i))) & (i<k) \\ C_i & (i>k) \end{cases} \quad (4)$$

ただし,

$$n_i=\left\lfloor \frac{T_k-C_i}{T_i} \right\rfloor+1 \quad (5)$$

とする.

証明. 最初に, タスク τ_k より周期の長いタスク $\tau_i(i>k)$ について見ていく. τ_i は, 余裕時間が正の時は τ_k より優先度が低いため, τ_i が τ_k をブロックするためには, 余裕時間が 0 になり, ジョブが最高優先度になる必要がある. すなわち, ジョブが周期の最後に C_i だけ実行される必要がある. このとき, 図 5 から分かるように, 区間 $[\bar{t}-T_k,\bar{t}]$ における τ_i の競合量 (図 5 の斜線部分) は, $T_i \geqq T_k$ から, 必ず C_i 以下である. よって, 競合量 $W_i^k(\bar{t}-T_k,\bar{t})$ は下式を満たす.

$$W_i^k(\bar{t}-T_k,\bar{t}) \leqq C_i \quad (6)$$

次に, タスク τ_k より周期の短いタスク $\tau_i(i<k)$ について見ていく. リリース時刻が区間より前で, デッドラインが区間内にあるジョブの競合量の合計は, それらのジョブが周期の最後に実行された時が最も大きい. また, リリース時刻が区間内で, デッドラインが区間より後にあるジョブの競合量は, 周期の最初に実行されたときが最も大きい. このとき, タスクの競合量が最大となるのは, 図 6 のように問題ジョブのデッドライン $\bar{t}$ が周期の最初に実行されたジョブの終了時刻と一致する場合である.

このとき, n_i を区間 $[\bar{t}-T_k,\bar{t}]$ で C_i すべて実行されるジョブ

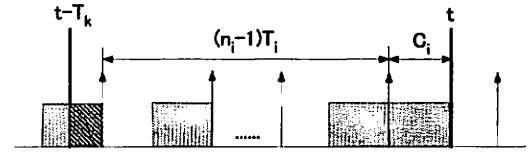

図6 $i<k$ の場合

の数とすると, n_i は以下で計算できる.

$$n_i=\left\lfloor \frac{T_k-C_i}{T_i} \right\rfloor+1 \quad (7)$$

一方, 区間の最初のジョブの競合量 (図 6 の斜線部分) は $T_k-((n_i-1)T_i+C_i)$ である. よって, タスク τ_i の競合量は, 下式を満たす.

$$W_i^k(\bar{t}-T_k,\bar{t}) \leqq n_iC_i+\min(C_i,\max(0,T_k-((n_i-1)T_i+C_i))) \quad (8)$$

以上, 式 (6)(8) より, 補題が示された.

□

簡単のため, タスク τ_i の区間 $[\bar{t}-T_k,\bar{t}]$ における競合負荷の上限値を β_i^k とすると, 補題 2 より, β_i^k は以下で表される.

$$\beta_i^k=\begin{cases} \frac{n_iC_i+\min(C_i,\max(0,T_k-((n_i-1)T_i+C_i)))}{T_k} & (i<k) \\ \frac{C_i}{T_i} & (i>k) \end{cases} \quad (9)$$

RMZL では, 余裕時間が 0 になるとジョブは最高優先度となり, デッドラインミスを防ごうとする. よって, 問題ジョブ $\tau_{k,j}$ がデッドラインミスを起こすためには, $\tau_{k,j}$ をブロックする少なくとも m 個のジョブが余裕時間 0 となり最高優先度で実行される必要がある. このことと, 補題 1, 2 より, RMZL のスケジュール可能性判定条件が導ける.

定理 1 (RMZL test). タスクセット $\tau=\{\tau_1,\ldots,\tau_n\}$ は, *(*少なくとも $m+1$ 個のタスクが下式を満たし, かつそのうちの一つが下式の不等号 $>$ を満たす*)* ことがなければ, m 個のプロセッサ上で *RMZL* によりスケジュール可能である.

$$\sum_{i\neq k}\beta_i^k \geqq m(1-U_k) \quad (10)$$

ただし β_i^k は式 *(9)* で定義される値である.

証明. 補題 1 より, 問題ジョブ $\tau_{k,j}$ は問題ウィンドウにおける他タスクの競合量の合計が $m(1-U_k)$ より大きいかまたは等しい場合にのみ余裕時間が 0 になり得る. 一旦 $\tau_{k,j}$ がゼロ余裕時間となると, 他の m 個のタスクがゼロ余裕時間になった時のみ $\tau_{k,j}$ はデッドラインミスを起こし得る. これは少なくとも $m+1$ 個のタスクが式 (10) を満たすときのみ起きる. よって定理が示される. □

ところで，補題1でジョブがゼロ余裕時間となり得る条件を示したが，この条件はより改善することが可能である．図4より，1つのブロックジョブの要求量が $(T_k - C_k)$ より大きくなった場合は，要求量を $(T_k - C_k)$ として考えれば十分であることは明らかである．このことから，補題3が成り立つ．

補題3. m 個のプロセッサ上で，タスクセット $\tau = \{\tau_1, ..., \tau_n\}$ を *RMZL* でスケジュールすると，デッドライン $\bar{t}$ を持つ問題ジョブ $\tau_{k,j}$ は以下の条件式を満たせば余裕時間は0となり，そのうち不等号 $>$ を満たせば余裕時間は負となる可能性がある．

$$\sum_{i \neq k} \min\left(W_i^k(\bar{t} - T_k, \bar{t})/T_k, 1 - U_k\right) \geqq m(1 - U_k) \qquad (11)$$

証明. 上の議論から，問題ジョブ $\tau_{k,j}$ は，下式を満たす時ゼロ余裕時間となり得る．

$$\sum_{i \neq k} \min\left(W_i^k(\bar{t} - T_k, \bar{t}), T_k - C_k\right) \geqq m(T_k - C_k) \qquad (12)$$

上の式は下式に変形できる．

$$\sum_{i \neq k} \min\left(W_i^k(\bar{t} - T_k, \bar{t})/T_k, 1 - U_k\right) \geqq m(1 - U_k) \qquad (13)$$

よって，補題が成り立つ． □

補題3，補題2から，よりよいスケジュール可能性判定である定理2が導ける．

定理2 (Refined RMZL test). タスクセット $\tau = \{\tau_1, ..., \tau_n\}$ は，*($m+1$ 個のタスクが下式を満たし，かつそのうちの一つが下式の不等号 $>$ を満たす)* ことがなければ，m 個のプロセッサ上で *RMZL* によりスケジュール可能である．

$$\sum_{i \neq k} \min(\beta_i^k, 1 - U_k) \geqq m(1 - U_k) \qquad (14)$$

ただし β_i^k は式 *(9)* で定義される値である．

証明. 補題3より，問題ジョブ $\tau_{k,j}$ は問題ウィンドウにおける他タスクの競合量の合計が $m(1-U_k)$ より大きいかまたは等しい場合にのみ余裕時間が0になり得る．一旦 $\tau_{k,j}$ がゼロ余裕時間となると，他の m 個のタスクがゼロ余裕時間になった時のみ $\tau_{k,j}$ はデッドラインミスを起こし得る．これは少なくとも $m+1$ 個のタスクが式 (14) を満たすときのみ起きる．よって定理が示される． □

5. 評　　価

本章では，一様乱数で生成したタスクを従来のグローバルRMアルゴリズムであるRM及びRM-US，そしてRMZLでスケジュールし，RMZLがスケジュール可能性において最も優れていることを示す．評価指標は下式で表されるスケジュール成功率 (Success Ratio) とする．

$$\text{Success Ratio} = \frac{\text{スケジュールが成功したタスクセットの数}}{\text{全体のタスクセット数}} \qquad (15)$$

5.1 評価環境

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決定する．m はプロセッサ数，U_{max} と U_{min} は各々与えられるタスクセットに含まれるタスクの利用率の最大値と最小値である．U_{total} はタスクセットに含まれるタスクの利用率の合計である．システム利用率 (System Utilization) は U_{total}/m と定義する．1つの m, U_{max}, U_{min} の組み合わせに対して，システム利用率30%から100%まで各々1000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する．これらパラメータに関しては様々な組み合わせが考えられるが，既存の組込み用のプラットフォームの規模を考慮して，プロセッサ数は2，4，8の3通りとする．各タスクの利用率の範囲は，$(U_{max}, U_{min}) = (1.0, 0.01)$ とする．タスクセット τ は以下のように生成する．$U(\tau) \leqq U_{total}$ である限り，新しいタスクを τ に追加していく．本論文では，特定のアプリケーションを対象としているわけではないため，各タスク τ_i の利用率 U_i は $[U_{max}, U_{min}]$ の範囲で一様分布で生成する．最後に生成されるタスクの利用率のみ，$U(\tau) = U_{total}$ となるように調節する．タスクはハーモニックであるとし，周期 T_i を $[100, 200, 400, 800, 1600, 3200]$ からランダムに選択する．タスクの実行時間 C_i は $C_i = U_i T_i$ として得られる．評価では，RMとRM-US，RMZLのそれぞれについて，スケジュール可能性判定と実際のスケジュールの両方を行った．実際のスケジュール時間はタスクセットのハイパーピリオド，つまり $lcm(\{T_i \| \tau_i \in \tau\})$ とする．RMとRM-USのスケジュール可能性判定は [4] のものを使う．すなわち，タスクセットは $\sum_i U_i \leqq (m/2)(1 - U_{max}) + U_{max}$ を満たせば m 個のプロセッサ上でRMによりスケジュール可能である．また，タスクセットは m 個のプロセッサにおいて，利用率が λ より高いタスクが k 個あり，それら以外のタスクの合計利用率が $((m-k)/2)(1-\lambda) + \lambda$ 以下であれば，RM-US[λ] でスケジュール可能である．本評価では，RM-US[$m/(3m-2)$] を用いる．また，RMZLのスケジュール可能性判定は定理2を用いる．

5.2 評価結果

シミュレーション結果を図7に示す．RMZL-sim，RM-US-sim，RM-simはそれぞれRMZL，RM-US，RMで実際にスケジュールしたときのスケジュール成功率，RMZL-test，RM-US-test，RM-testはそれぞれRMZL，RM-US，RMのスケジュール可能性判定を行ったときのスケジュール成功率を表す．まず，実際にスケジュールした時の結果について見ると，RMZLはどのプロセッサ数においても最もスケジュール成功率が高かった．また，RMやRM-USはプロセッサ数が増えるにつれてスケジュール成功率が低下する傾向があるのに対して，RMZLはスケジュール成功率は逆にやや増加した．これは，プロセッサ数が増えるにつれて，許されるゼロ余裕時間のタスク数が増えるからであると考えられる．一方，RM-USは，$m = 2$ の場合で，RMよりスケジュール成功率が低下していた．これはRM-USが，RM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にする半面，RMでスケジュールできるはずの多くのタスクをスケジュール不能にしてしまうことを示している．この点で，RMZLがRM-USよりも優れていることがわかる．

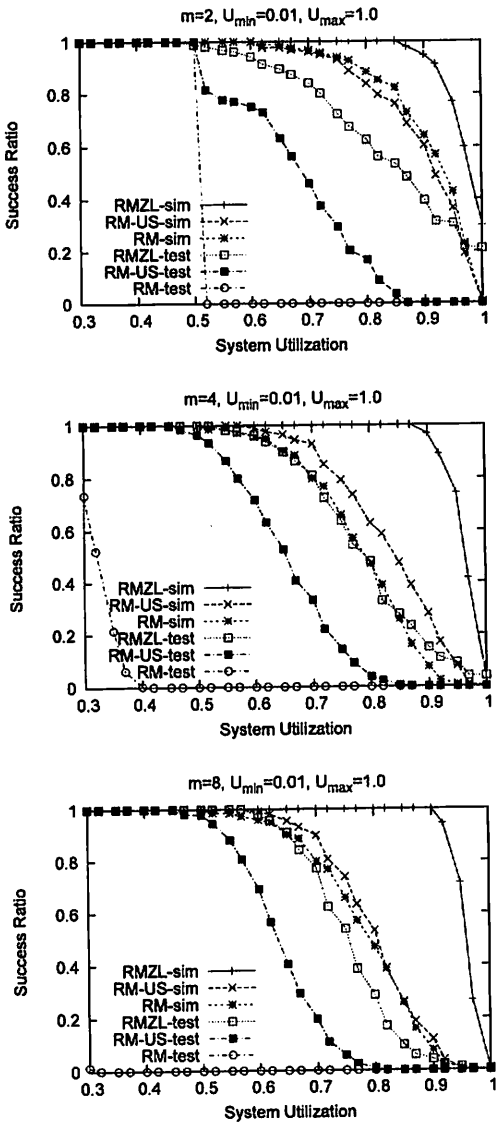

図7 システム利用率に応じたスケジュール成功率

次に，スケジュール可能性判定の結果を見ると，ほとんどの場合で RMZL が RM-US 及び RM よりもよい性能を示しているが，$m = 2$，System Utilization = 0.5 付近では RMZL の性能が低下している．RMZL は常に RM と等しいか良い性能を示すはずであるので，この条件でのスケジュール可能性判定が適当でない可能性がある．また，RMZL のスケジュール可能性判定では，プロセッサ数が増えるにつれ他のアルゴリズムと同様にスケジュール成功率が低下しており，このような状況では判定がより厳密でない．さらに，一般的にグローバル方式の RM 及び EDF アルゴリズムのスケジュール可能性判定は非常に悲観的なものであり，それが評価結果にも表れている．これらのスケジュール可能性判定の改善は今後の課題とする．

6. 結論及び今後の課題

本論文では，グローバル RM の長所を残しつつ高いスケジュール可能性を実現できる実用的なリアルタイムスケジューリングアルゴリズム RMZL を提案した．また，ハードリアルタイムシステムには不可欠な RMZL のスケジュール可能性判定を解析した．シミュレーションによる評価では，グローバル RM に基づく既存のどのアルゴリズムよりも高いスケジュール可能性を確認できた．

評価で見たように，本論文で示したスケジュール可能性判定は厳密でない部分があり，また非常に悲観的なものである．また，本論文では RMZL で利用可能なシステム利用率の上限を解析できていない．アルゴリズムの比較は主にこの利用率の上限によって行われるため，この解析は重要である．また，RMZL は固定優先度アルゴリズムではなくなるため，オーバーヘッドが高くなる可能性があり，またゼロ余裕時間の検出など実装面の問題がある可能性がある．よって，ITRON 仕様 OS などに RMZL を実装し，評価をとる必要があると考えられる．これらは今後の課題とする．

謝辞 本研究は，科学技術振興機構 CREST の支援による．

文 献

[1] B. Andersson and J. Jonsson. The Utilization Bounds of Partitioned and Pfair Static-Priority Scheduling on Multiprocessors are 50%. In *Proc. of the 15th Euromicro Conference on Real-Time Systems*, pages 33–40, 2003.

[2] B. Andersson and E. Tovar. Multiprocessor Scheduling with Few Preemptions. In *Proc. of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 322–334, 2006.

[3] J. Jonsson B. Andersson, S. Baruah. Static-priority Scheduling on Mutiprocessors. In *Proc. of the 22nd IEEE Real-Time Systems Symposium*, pages 193–202, 2001.

[4] T. P. Baker. Multiprocessor EDF and Deadline Monotonic Schedulability Analysis. In *Proc. of the 24th IEEE Real-Time Systems Symposium*, pages 120–129, 2003.

[5] T. P. Baker. M. Cirinei and T. P. Baker. In *Proc. of the 19th Euromicro Conference on Real-Time Systems*, pages 9–18, 2007.

[6] S. Baruah, N. Cohen, C. G. Plaxton, and D. Varvel. Proportionate progress: A notion of fairness in resource allocation. *Algorithmica*, 15:600–625, 1996.

[7] H. Cho, B. Ravindran, and E. D. Jensen. An Optimal Real-Time Scheduling Algorithm for Multiprocessors. In *Proc. of the 27th IEEE Real-Time Systems Symposium*, pages 101–110, 2006.

[8] S. Cho, S. K. Lee, A. Han, and K. J. Lin. Efficient Real-Time Scheduling Algorithms for Multiprocessor Systems. E85-B:2859–2867, 2002.

[9] S. K. Dhall and C. L. Liu. On a real-time scheduling proglem. *Operations Research*, 26:127–140, 1978.

[10] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20:46–61, 1973.

[11] K. Olukotun, B. Nayfe, L. Hammond, K. Wilson, and K.Chang. The Case for a Single-Chip Multiprocessor. In *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 2–11, 1996.

[12] D. M. Tullsen, S. J. Eggers, and H. M. Levy. Simutaneos Multithreading: Maximizing On-Chip Parallelism. In *Proc. of the 22nd Annual International Symplsium on Computer Architecture*, pages 392–403, 1995.