B-017

危険余裕時間に基づいた疑似デッドラインを持つ静的スケジューリングアルゴリズム
Scheduling algorithm with pseudo deadline based on critical laxity time

田中　翼†　　兪　明連†　　横山　孝典†
Tsubasa Tanaka　　Myungryun Yoo　　Takanori Yokoyama

1. 序論

近年，組込みリアルタイムシステムでは，ロボットやユビキタスアプリケーションなど，高性能なプラットホームを必要とする傾向にある[1]．そのため，高い情報処理能力を得るためにマルチプロセッサ技術の利用が一般化しており，マルチプロセッサ環境におけるリアルタイムスケジューリングアルゴリズムが重要となっている．従来のシングルプロセッサ環境における静的優先度方式で最適なスケジューリングアルゴリズムであるRM (Rate Monotonic)[2]はマルチプロセッサ環境で最適ではないことが知られており[3]，マルチプロセッサ環境で最適なアルゴリズムは確立されていない．そのため現在もRMに基づいたマルチプロセッサ向けのアルゴリズムが研究されている．

本研究では，実行時のオーバーヘッド削減のためにスケジューラ起動回数を抑え，かつ高いスケジュール成功率を実現するアルゴリズムを提案し，そのスケジュール可能性を解析することを目的とする．そのため，RMに危険余裕時間ルールを付加したアルゴリズムRMCL(RM until Critical Laxity)[4]をマルチプロセッサ向けに改良したアルゴリズムとそのうえで疑似デッドラインを設定するアルゴリズムを提案する．

2. システムモデル

本研究では，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個のプロセッサを持ち，N個のタスクから構成されるタスクセット$\tau = \tau_1, \tau_2, \ldots, \tau_i, \ldots, \tau_N$が与えられる．各タスク$\tau_i$は最悪実行時間$C_i$と周期$T_i$により$\tau_i = (C_i, T_i)$と構成される．$U_i = C_i/T_i$をタスク$\tau_i$の利用率，$U(\tau)/M$をシステム利用率と呼ぶ．各タスク$\tau_i$は周期$T_i$の間隔でジョブを生成し，タスク$\tau_i$が生成する$k$番目のジョブを$\tau_{ik}$と表す．$\tau_{ik}$は時刻$r_{ik}$でリリースされ，デッドライン$d_{ik}$は次のジョブのリリース時刻とする．ある時刻$t$において，ジョブ$\tau_{ik}$の残り実行時間を$C_{ik}(t)$とするとき，$\tau_{ik}$の余裕時間$L_{ik}$を$L_{ik} = d_{ik} - t - C_{ik}(t)$と定義する．

3. 関連研究

関連研究として，RM と，RM をもとにした RMZL(RM until Zero Laxity)[1][5]，RMCL(RM until Critical Laxity)についてそれぞれ述べる．

3.1 RM(Rate Monotonic)

RM は周期の短いタスクに高優先度を与える静的優先度方式のスケジューリングアルゴリズムであり，シングルプロセッサ環境で最適である．しかし，マルチプロセッサ環境ではスケジュール成功率やスケジュール可能性が低いという問題があり，最適でないことが知られている．また，RM のスケジューラ起動時刻は各タスクの起動時，完了時である．

3.2 RMZL(RM until Zero Laxity)

RMZL は RM にゼロ余裕時間ルールを適用したスケジューリングアルゴリズムであり，ゼロ余裕時間ルールとは，余裕時間 $L_{ik}(t)$ が 0 になったタスクに最高優先度を与えるルールである．RMZL は RM と比べてスケジュール可能性を大きく向上させた．また，余裕時間 0 のタスクが発生しない間は RM と同じ動作をするため，RM でスケジュール可能なタスクセットはすべてスケジュール可能である．そのため，スケジュール成功率も RM から大きく向上している．しかし，余裕時間の計算のためにスケジューラを毎単位時間起動する必要があり，オーバーヘッドが大きいという問題点がある．

3.3 RMCL(RM until Critical Laxity)

RMCL は RM において，危険余裕時間の概念を設定し，危険余裕時間であるタスク(以降クリティカルタスク)に最高優先度を与えるアルゴリズムである．また，RMCL のスケジューラの起動時刻は各タスクの起動時，完了時である．

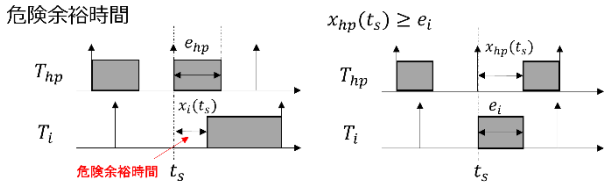

図 1　危険余裕時間の概念図

危険余裕時間について詳しく述べる．任意のスケジューラ起動時刻t_sにおいて，RM でスケジュールした場合に次に実行されるタスクτ_{hp}の残り実行時間をe_{hp}，余裕時間を$x_{hp}(t_s)$とし，それ以外のタスク$\tau_i(i \neq hp)$の残り実行時間をe_i，余裕時間$x_i(t_s)$とする．これらを用いた条件式(1)が成り立つ場合にタスクτ_i の余裕時間$x_i(t_s)$ を危険余裕時間と定義する．

$$e_{hp} > x_i \quad (1)$$

だが条件式(1)のみだとクリティカルタスクに実行権を奪われたタスクτ_{hp} がデッドラインミスを起こす可能性がある．それを防ぐために条件式(2)を追加し，条件式(1)と(2)を満たした場合のみτ_i に最高優先度を与える．

$$x_{hp}(t_s) \geq e_i \quad (2)$$

RMCL は RM と同程度のスケジューラ起動回数でスケジュール可能性を向上させた．また，クリティカルタスクが発生しない間は RM と同じ動作をするため，RMZL と同様の理由でスケジュール成功率も RM より向上している．しかし，問題点としてアルゴリズムがマルチプロセッサに対応していないこと，RMZL と比べるとスケジュール成功率が大きく下がることが挙げられる．

4. マルチプロセッサ向け RMCL

RMCL をマルチプロセッサ向けに拡張したアルゴリズムを提案する(以降 mRMCL)．mRMCL では RM に従って実行されるM個のタスクの中で最小残り実行時間をもつタスクをt_{hp}とし，条件式(1)，(2)を満たしたタスクに最高優先

†東京都市大学　Tokyo City University

度を与える．これを残り実行時間の短い順にt_{hp}を入れ替えながら前述したM個のタスク全てで行う．図 2 に mRMCL のスケジュール例を示す．ただし，実行するタスクセット及びプロセッサ数は$\tau=\{\tau_1=(2,3),\tau_2=(2,4),\tau_3=(4,6)\},M=2$とする．

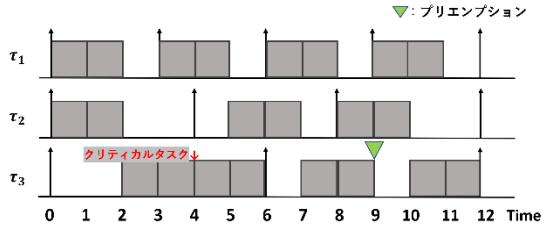

図 2.　mRMCL のスケジュール例

図 2 から時刻 4 でτ_3がクリティカルタスクとなり，最高優先度を獲得することでデッドラインミスを起こすことなくスケジュール出来ていることがわかる．

5. RMCLPD(RMCL add Pseudo Deadline)

mRMCL を基にスケジュール成功率を向上させるアルゴリズムを提案する．RMCLPD は疑似デッドラインと疑似実行時間を使用し，各タスクを分割して実行するアルゴリズムである．各タスクの周期の半分に疑似デッドライン，実行時間の半分に疑似実行時間を設定する．つまり，あるジョブτ_{ik}の疑似デッドラインを$d'_{ik}=r_{ik}+\lceil T_i/2\rceil$，疑似実行時間を$C'_{ik}=\lceil C_i/2\rceil$とする．また，ある時刻$t$において，ジョブ$\tau_{ik}$の残り疑似実行時間を$C'_{ik}(t)$とするとき，$\tau_{ik}$の疑似余裕時間$L'_{ik}$を$L'_{ik}=d'_{ik}-t-C'_{ik}(t)$と定義する．また，スケジューラの起動時刻は各タスクの起動時，完了時，疑似実行時間完了時，疑似デッドライン時とする．残り疑似実行時間と疑似余裕時間を用いて mRMCL と同様にクリティカルタスクの判定を行い，クリティカルタスクに最高優先度を与える．図 3 に RMCLPD のスケジュール例を示す．ただし，実行するタスクセット及びプロセッサ数は$\tau=\{\tau_1=(2,3),\tau_2=(4,6),\tau_3=(6,9)\},M=2$とする．

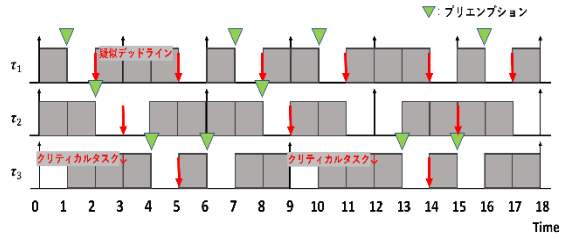

図 3.　RMCLPD のスケジュール例

図 3 から時刻 3，12 でそれぞれτ_3がクリティカルタスクとなり，最高優先度を獲得して実行されていることがわかる．このようにすることで，周期の長いタスクが早い時刻で実行できるため，デッドラインミスを起こさずにスケジュール出来ている．

6. スケジュール可能性解析

本研究では，関連研究である RM, RMZL のスケジュール可能性解析に用いられた反応時間解析(RTA)[6]を応用して mRMCL 及び RMCLPD のスケジュール可能性判定を行う．

6.1 反応時間解析における用語と補題

解析のために二つの用語を定義する．

［定義1］（干渉長） 区間$[a,b)$におけるタスクτ_kへの干渉長$I_k(a,b)$は，τ_kがM個以上の高優先度タスクによってブロックされ実行できない区間の合計長を表す．また，区間$[a,b)$におけるタスクτ_kへ対するタスクτ_iの干渉長$I_k^i(a,b)$は，τ_kがτ_iにブロックされ実行できない区間の合計長を表す．

［定義2］（仕事量）区間$[a,b)$におけるタスクτ_iの仕事量$W_i(a,b)$は与えられたスケジュールにおいてタスクτ_iが区間$[a,b)$で実行しなければならない実行量を表す．

干渉長に関して，以下の補題が示されている．

［補題1］全てのグローバルスケジューリング方式のアルゴリズムに対して，以下の式(3)が成り立つ[7]．

$$I_k(a,b)\geq x \Leftrightarrow \sum_{i\neq k} min\left(I_k^i(a,b),x\right)\geq Mx \qquad (3)$$

それぞれのアルゴリズムにおける解析の流れを以下に示す．各タスクτ_kが最大反応時間を得るときのジョブをJ_k^*とする．まず，ジョブJ_k^*について，J_k^*のリリース時刻から応答までの区間$\left[r_k^*,r_k^*+R_k^{ub}\right)$における$\tau_k$への干渉長$I_k$の上限値$I_k^{ub}$を求める．次に，$I_k^{ub}$と$\tau_k$の実行時間から，$\tau_k$の反応時間の上限値$R_k^{ub}$を求める．そして，$R_k^{ub}$から$\tau_k$のジョブの余裕時間の下限値$L_k^{lb}=T_k-R_k^{ub}$を求める．この余裕時間の下限値$L_k^{lb}$からそれぞれのアルゴリズムにおけるスケジュール可能性判定条件を導く．

ここで，干渉長の上限値I_k^{ub}に関して，以下の補題が示されている．

［補題2］区間$[a,b)$におけるタスクτ_iのτ_kへの干渉長$I_k^i(a,b)$は，区間$[a,b)$におけるτ_iの仕事量$W_i(a,b)$を超えない[6]．よって，干渉するタスクτ_iの仕事量の上限値W_i^{ub}を求めることで，干渉長の上限値I_k^{ub}が得られる．

6.2 RM における仕事量の上限値

RMの各タスクは自分より周期の短いタスクにのみブロックされる．そのため，仕事量は図4のように，リリース時刻が区間より前でデッドラインが区間の中にあるジョブ（以降carry-inジョブ）の実行開始時刻が区間の先頭と一致，リリース時刻が区間の中でデッドラインが区間の外にあるジョブ（以降carry-outジョブ）の実行がそのジョブのリリースと同時に開始されたとき最大となる[6]．

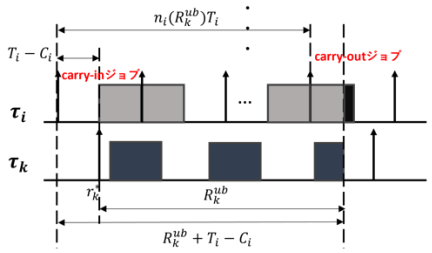

図 4　RM における仕事量

このときの区間$\left[r_k^*,r_k^*+R_k^{ub}\right)$における$\tau_i$の仕事量の上限値$W_i^{ub}\left(r_k^*,r_k^*+R_k^{ub}\right)$を計算する．まず，$n_i\left(R_k^{ub}\right)$を区間内で$C_i$全て実行できるジョブの数とすると，$n_i\left(R_k^{ub}\right)$は以下の式(4)で計算できる．

$$n_i\left(R_k^{ub}\right)=\left\lfloor\frac{R_k^{ub}+T_i-C_i}{T_i}\right\rfloor \qquad (4)$$

また，carry-outジョブの仕事量は，$min(C_i,R_k^{ub}+T_i-C_i-n_i\left(R_k^{ub}\right)T_i$である．よって，RMにおける仕事量の上限値$W_i^{ub}\left(r_k^*,r_k^*+R_k^{ub}\right)=W_i^{ub}(R_k^{ub})$は，以下の式(5)で与えられる．

$$W_i^{ub}(R_k^{ub})=n_i\left(R_k^{ub}\right)C_i+min(C_i,R_k^{ub}+T_i-C_i-n_i\left(R_k^{ub}\right)T_i \qquad (5)$$

6.3 RMZL における仕事量の上限値

RMZLでは，ゼロ余裕時間ルールにより自分より周期の長

いタスクにもブロックされる．そのため，$i > k(T_i \geq T_k)$ と$i < k(T_i \leq T_k)$に場合分けして考える．
$i > k$ の場合を図5に示す．$i > k$ よりτ_kの優先度が τ_i より高いため，τ_i が区間内で仕事をするためにはτ_i のジョブがゼロ余裕時間となる必要がある．またその仕事量の上限値は$T_i \geq T_k$ より，たかだかC_i である．よって$W_i^{ub}(R_k^{ub}) = C_i, (i > k)$ である．

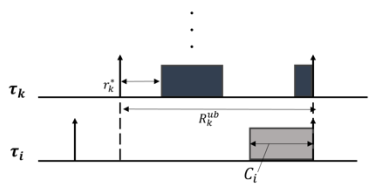

図 5　RMZL における仕事量 $(i > k)$

次に，$i < k(T_i \leq T_k)$ の場合を考える．この場合の仕事量はRMと同様の状況で最大となる．ただし，干渉するタスクτ_i の反応時間が求まっているのならば，τ_i の余裕時間L_k^{lb}を考慮することで干渉長を少なく見積もることができる．よって，$W_i^{ub}(R_k^{ub}) = n_i(R_k^{ub})C_i + min(C_i, R_k^{ub} + T_i - C_i - L_i^{lb} - n_i(R_k^{ub})T_i, (i < k)$である．ただし，$n_i$ は以下の式(6)で計算される．

$$n_i(R_k^{ub}) = \left\lfloor \frac{R_k^{ub} + T_i - C_i - L_i^{lb}}{T_i} \right\rfloor \quad (6)$$

RMZL の仕事量の上限値は以下の式(7)で表される

$$W_i^{ub}(R_k^{ub}) = \begin{cases} n_i(R_k^{ub})C_i + min(C_i, R_k^{ub} + T_i - C_i - L_i^{lb} \\ \quad -n_i(R_k^{ub})T_i, (i < k) \\ C_i, (i > k) \end{cases} \quad (7)$$

以上より，それぞれのアルゴリズムにおける反応時間の上限値が求められる．

［定理1］RM，RMZLでスケジュールされたマルチプロセッサシステムのタスクτ_k の反応時間の上限値は，以下の式(8)のR_k^{ub} に関して$R_k^{ub} = C_k$ から始まる不動点反復法を解くことによって求められる[6]．

$$R_k^{ub} \leftarrow C_k + \left\lfloor \frac{1}{M} \sum_{i \neq k} \hat{I}_k^i(R_k^{ub}) \right\rfloor \quad (8)$$

ただし$\hat{I}_k^i(R_k^{ub}) = \min(W_i^{ub}(R_k^{ub}), R_k^{ub} - C_k + 1)$ であり，$W_i^{ub}(R_k^{ub})$ は式(5)，(7)それぞれの値である．

各アルゴリズムにおける反応時間の上限値が求められることで，それぞれのスケジュール可能性判定が得られる．

［定理2］タスクセット$\tau = \tau_1, \tau_2, \ldots, \tau_N$ は，全てのタスクが以下の式(9)を満たせばRM によりスケジュール可能である．ただし，R_k^{ub} は定理1を式(5)で計算して求められる値である．

$$L_k^{lb} = T_k - R_k^{ub} \geq 0 \quad (9)$$

［定理3］タスクセット$\tau = \tau_1, \tau_2, \ldots, \tau_N$ は，「少なくとも$M + 1$ 個のタスクが以下の式(10) を満たし，かつそのうちの一つのタスクが式(10)の不等号< を満たす」ことがなければ，M 個のプロセッサシステムにおいてRMZLによりスケジュール可能である．ただし，R_k^{ub} は定理1を式(7)で計算して求められる値である．

$$L_k^{lb} = T_k - R_k^{ub} \leq 0 \quad (10)$$

6.4 mRMCL 及び RMCLPD のスケジュール可能性解析

mRMCL 及び RMCLPD(以降提案アルゴリズム)のスケジュール可能性解析では，RM の解析を応用し，余裕時間が正か負かで場合分けを行う．

提案アルゴリズムの仕事量の上限値は RM と同様の状況である．そのため，余裕時間が正の場合は，タスクセット$\tau = \tau_1, \tau_2, \ldots, \tau_N$の全てのタスクが式(9)を満たせば提案アルゴリズムによりスケジュール可能である．このとき，R_k^{ub}は定理 1 を式 (5)で計算して求められる値である．

次に，余裕時間が負の場合を考える．提案アルゴリズムでは余裕時間が負，つまりあるタスクτ_k が$L_k^{lb} = T_k - R_k^{ub} < 0$ となった場合でも危険余裕時間によってτ_kのデッドラインミスを回避することが可能である．デッドラインミスが発生するのは，τ_k に最高優先度を与えたことでほかのタスクがクリティカルタスクになってしまう場合に限られる．また，τ_k がクリティカルタスクになって最高優先度が与えられた場合，τ_k の実行は早まることがあっても遅れることはない．よって，影響を受けるのはτ_k よりも周期の短いタスクのみである．この性質を利用して余裕時間が負の場合の可能性解析を行う．$L_k^{lb} = T_k - R_k^{ub} < 0$ のとき，危険余裕時間による最高優先度を考えない場合にτ_k が周期T_k 以内に消費しきれない最悪実行時間E_kは式(11)で表せる．

$$E_k = max\{R_k - T_k, C_k\} \quad (11)$$

危険余裕時間によってτ_k に最高優先度が割り当てられると，すべてのτ_i $(i < k)$ の実行がE_k 時間遅れる可能性を持つ．その結果，$L_i^{lb} = T_i - R_i^{ub} < 0$となる$\tau_i$ が存在すればタスクセットはデッドラインミスを発生する可能性がある．いいかえると，すべてのτ_i が式(12)を満たしていればタスクセットはスケジュール可能である．

$$L_k^{lb} = T_k - (R_j^{ub} + W_k) \geq 0 \quad (i = 1,2, \ldots k-1) \quad (12)$$

7. 評価

シミュレーションによって評価を行う．シミュレーションではシステム利用率 30%から 100%の範囲で無作為に生成した各 1000 個のタスクセットを投入し，各アルゴリズムでシミュレーションを行う．また，プロセッサ数は4とし，シミュレーション範囲は[0, 1000000]とする．比較する項目は平均スケジューラ起動回数(全スケジューラ起動回数/タスクセット数)，平均プリエンプション回数(全スケジューラ起動回数/タスクセット数)，スケジュール成功率(スケジュール成功したタスク数/タスクセット数)，スケジュール可能性判定(スケジュール可能であると判定されたタスク数/タスクセット数)の 4 項目とする．

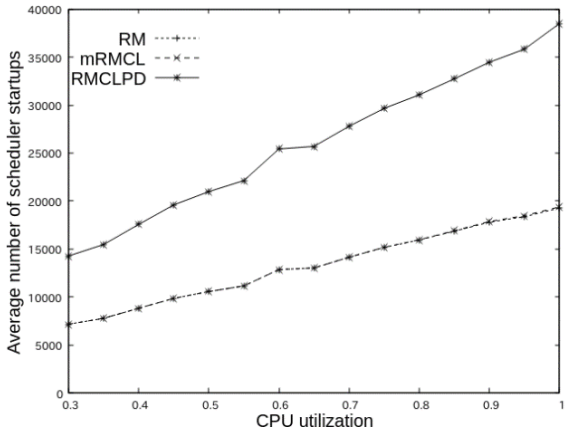

図 6　平均スケジューラ起動回数

図 6 に平均スケジューラ起動回数のグラフを示す．ただし，RMZL は全てのタスクセットで 1000000 回スケジュ

ーラを起動するため省略している．RM と比べると mRMCL は同程度，RMCLPD は平均 2.01 倍増加しているが，どちらのアルゴリズムも RMZL の 1000000 回と比べると，システム利用率 100%で mRMCL が$\frac{1}{51}$，RMCLPD は$\frac{1}{26}$と減少している．

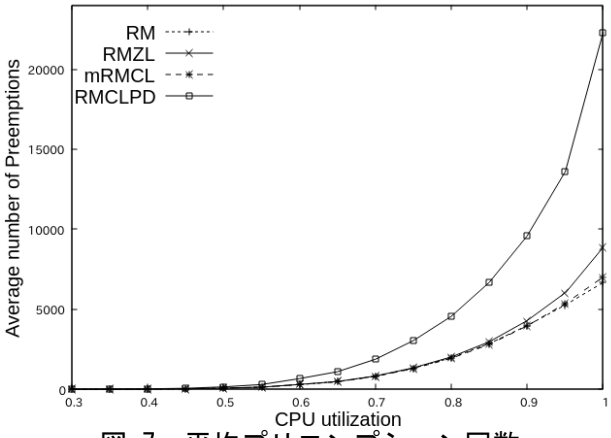

図 7　平均プリエンプション回数

図 7 に平均プリエンプション回数を示す，平均プリエンプション回数は，mRMCL では RM と同程度かつ RMZL より減少，RMCLPD は RMZL と比べて平均 2.32 倍増加している．

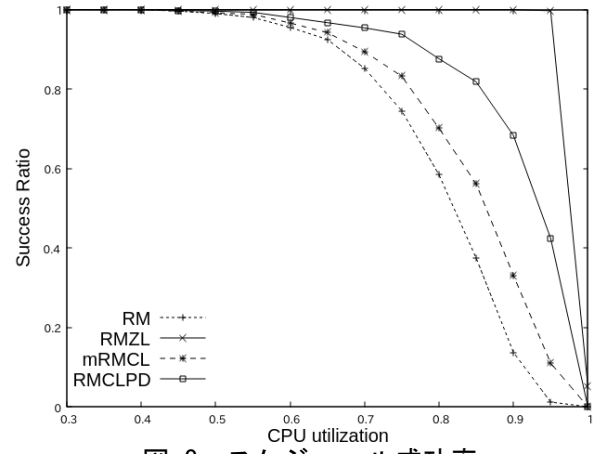

図 8　スケジュール成功率

図 8 にスケジュール成功率のグラフを示す，スケジュール成功率は，システム利用率 90%で mRMCL は RM と比べて 19.4%，RMCLPD は mRMCL と比べて 35.2%スケジュール成功率を向上させている．

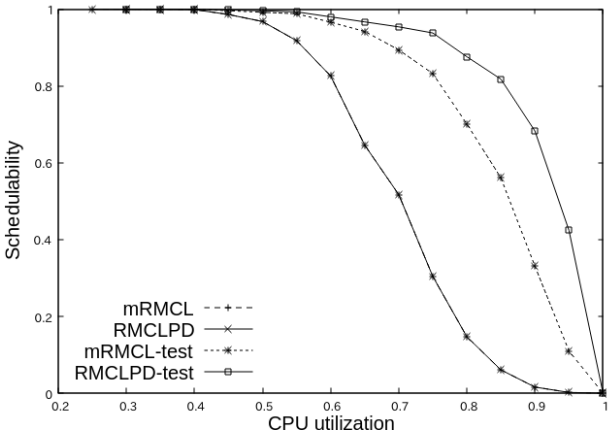

図 9　スケジュール可能性判定

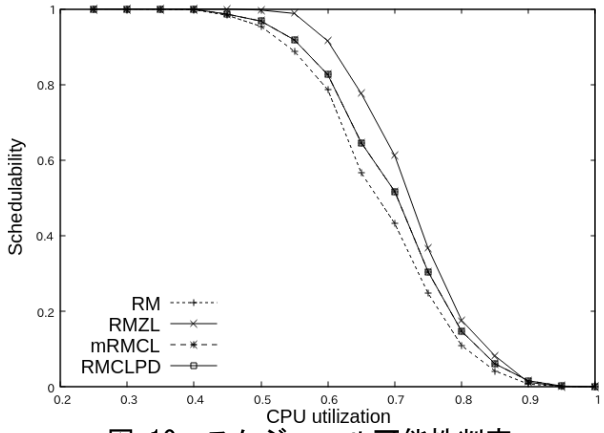

図 10　スケジュール可能性判定

図 9，10 にスケジュール可能性判定のグラフを示す．図 9 のグラフの mRMCL-test，RMCLPD-test はそれぞれのアルゴリズムで実際にスケジュールを行った成功率である．実際にスケジュールを行った結果と比べるとスケジュール可能性は mRMCL，RMCLPD ともに低くなっているが，元の成功率が低い分 mRMCL の方がより正確な判定結果になったといえる．また，図 10 の判定結果では，RM 以上 RMZL 以下であり，実際の成功率と同様であることがわかる．

8. 結論

本研究では，RMCL をマルチプロセッサ向けに拡張した mRMCL と mRMCL に疑似デッドラインを設定した RMCLPD を提案し，シミュレーションによりスケジューラの起動回数を抑え，かつスケジュール成功率を向上させていることを示した．しかし，RMCLPD のプリエンプション回数は増加してしまうため，スケジューラ起動回数とプリエンプション回数のオーバーヘッドとの関係について今後も研究する．また，スケジュール可能性解析について，mRMCL と RMCLPD の相違点を示せていないため，スケジュール可能性解析の精度向上とともに今後の課題として研究する予定である．

謝辞

本研究は JSPS 科研費 20K11755 の助成を受けたものです．

参考文献

[1] 武田 瑛，加藤 真平，山崎 信行,“Rate monotonic に基づくマルチプロセッサ用リアルタイムスケジューリング”，情報処理学会論文誌，コンピューティングシステム，vol.2, No.1, pp.64-74, (2009).
[2] C. L. Liu, J. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment", Journal of the ACM, Vol. 20, No. 1, pp. 46-61 (1973)
[3] S.K Dhall, C.L.Liu, “On a real-time scheduling problem”, Operations Research, vol.26, No.1, pp.127-140, (1978)
[4] 加藤 真平,山崎 信行,“Linux カーネル用リアルタイムスケジューリングモジュール”,情報処理学会論文誌，コンピューティングシステム, vol.2, No.1, pp.75-86, (2009)
[5] R.I. Davis, A. Burns, “Fpzl schedulability analysis,”17th IEEE Real-Time and Embedded Technology and Applications Symposium, pp.245–256, (2011)
[6] M. Bertogna, M. Cirinei,”Response-time analysis for globally scheduled symmetric multiprocessor platforms,” 28th IEEE International Real-Time Systems Symposium, pp.149–160, (2007)
[7] M. Bertogna, M. Cirinei, G. Lipari,“Improved schedulability analysis of edf on multiprocessor platforms,”Proc. 17th Euromicro Conference on Real-Time Systems, pp.209–218, (2005)