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

2007－ＳＬＤＭ－129／2007－ＥＭＢ－4（16）
2007／3／16

マルチプロセッサにおけるスケジュール可能性を向上するための動的優先度スケジューリングアルゴリズム

加藤 真平† 山崎 信行†

† 慶應義塾大学
〒223-8522 横浜市港北区日吉 3-14-1
E-mail: †{shinpei,yamasaki}@ny.ics.keio.ac.jp

あらまし 近年，組込み実時間システムにおいてもマルチプロセッサの利用が主流になりつつある．しかしながら，マルチプロセッサにおける実時間スケジューリングは複雑であることが知られている．一般的に，高いスケジュール可能性を実現できる洗練されたスケジューリングアルゴリズムでは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る実用性の低下が問題となる．一方で，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はできない．そこで，本論文では，高いスケジュール可能性を実現できる実用的な動的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新しいスケジューリング方式に基づいており，従来のスケジューリング方式に基づくあらゆるアルゴリズムのスケジュール可能なシステム利用率の上限が高々50%であるのに対して，その上限を73%に向上できる．評価では，実際のスケジュールにおいても，提案アルゴリズムが既存の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 実時間システム，動的優先度スケジューリング，スケジュール可能性解析，マルチプロセッサ

A Dynamic-Priority Scheduling Algorithm for Improving the Schedulability on Multiprocessors

Shinpei KATO† and Nobuyuki YAMASAKI†

† Keio University
3-14-1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa 223-8522 Japan
E-mail: †{shinpei,yamasaki}@ny.ics.keio.ac.jp

Abstract Recent embedded real-time systems tend to be realized by a multiprocessor system. However it is known to be complicated to realize real-time scheduling on multiprocessors. In general, scheduling algorithms with high schedulability generate a number of context switches and task migrations that incur a significant overhead. Because of the overhead, those algorithms are often considered not to be practical. Meanwhile simple algorithms cannot improve the schedulability. This paper proposes a practical dynamic-priority scheduling algorithm with high schedulability. Our algorithm is based on a new scheduling scheme and achieves 73% of the least upper bound of the schedulable utilization, whereas any algorithm based on the existing scheduling scheme achieves at most 50%. The evaluation shows that our algorithm also outperforms the existing algorithms in the real schedule point of view.
Key words Real-Time Systems, Dynamic-Priority Scheduling, Schedulability Analysis, Multiprocessor Systems

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って，組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てきている．そのため，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，実時間処理の点で，マルチプロセッサにおける実時間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．Rate Monotonic（RM）[6]や Earliest Deadline First（EDF）[6]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もはや最適ではないことが証明されている [4].

Pfair スケジューリングアルゴリズム[2] および LLREF アルゴリズム[3] 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アルゴリズムとして知られている．しかしながら，それらのアルゴリズムでは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る実用性の低下が問題となる．一方で，EDF-US[5] や EDF-FF[7]，EDF-BF[7] 等のアルゴリズムは，実装や計算は簡潔であるが，スケジュール可能なシステム利用率の上限（以下，スケジュール可能上限と略す）は高々50% かそれ以下であることが知られている．Andersson らは，このトレードオフを解決するアルゴリズムとして EKG[1] を提案した．EKG は，EDF-FF や EDF-BF のように各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に基づいているが，いくつかのタスクに関しては 2 つのプロセッサに分割して割り当てることでスケジュール可能性の向上を実現している．EKG のスケジュール可能上限は分割されるタスク数 k に依存しており，$k = 2$ の場合は 66% である．$k = M$（M はシステム内のプロセッサ数）の場合は 100% であり最適なアルゴリズムとなるが，代わりにコンテキストスイッチの回数が増えてしまう．

本論文では，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を必要とせず，且つ高いスケジュール可能性を実現できる実用的な動的優先度スケジューリングアルゴリズムを提案し，そのスケジュール可能上限が 73% であることを証明する．アルゴリズムはパーティショニング方式に基づいているが，EKG と同様にいくつかのタスクに関しては 2 つのプロセッサに分割して割り当てることでスケジュール可能性を向上する．

2. システムモデル

本論文ではマルチプロセッサ実時間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 $P_1, P_2, ..., P_M$ 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 の利用率を $U_i = C_i/T_i$ で表す．また，あるタスクサブセット Λ に含まれるタスクの利用率の合計を $U(\Lambda) = \sum_{\tau_i \in \Lambda} U_i$ で表す．すなわち，$U(\Gamma)$ はシステム全体の負荷を意味する．ここで，$U(\Gamma)/M$ をシステム利用率と定義する．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．全ての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上で並列に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，すで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

提案アルゴリズムは，タスクを特定のプロセッサへ割り当てるフェーズとタスクを実行するフェーズから構成される．本節では，各フェーズにおけるアルゴリズムについて述べる．

Preliminary:
task set Γ is sorted so that $T_1 \leqq T_2 \leqq ... \leqq T_N$.
all the per-processor task sets are empty: $\forall m, \Lambda_m = \emptyset$.

1. $i \leftarrow 1$, $m \leftarrow 1$, $U_s \leftarrow 0$ and $U''_s \leftarrow 0$;
2. $U_{lub} = \min\left\{U''_s + \frac{1-U''_s}{1+U''_s},\ U''_s + \frac{2-U''_s-U_s}{2U''_s-U_s+2}\right\}$
3. **if** $U(\Lambda_m \cup \tau_i) \leqq U_{lub}$
4. $\quad \Lambda_m \leftarrow \Lambda_m \cup \tau_i$;
5. **else if** $m = M$
6. $\quad$ allocation fails;
7. **else**
8. $\quad$ split τ_i into $\tau'_i(T_i, C'_i)$ and $\tau''_i(T_i, C''_i)$ where $C'_i = T_i\{U_{lub} - U(\Lambda_m)\}$ and $C''_i = C_i - C'_i$;
9. $\quad \Lambda_m \leftarrow \Lambda_m \cup \tau'_i$ and $\Lambda_{m+1} \leftarrow \emptyset \cup \tau''_i$;
10. $\quad U''_s \leftarrow \frac{C''_i}{T_i}$ and $U_s \leftarrow \frac{C_i}{T_i}$;
11. $\quad m \leftarrow m + 1$;
12. **if** $i = N$
13. $\quad$ allocation exits;
14. $i \leftarrow i + 1$;
15. go back to step 2.;

図 1 SIP アルゴリズム

3.1 タスク割当てフェーズ

まず，各プロセッサにタスクを割り当てるアルゴリズムとして Sequential allocation in Increasing Period（SIP）を提案する．アルゴリズムは図 1 に示されるとおりである．各タスクは $T_1 \leqq T_2 \leqq \cdots \leqq T_N$ 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_m 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_m と定義する．

まず，タスク τ_i をプロセッサ P_m に割り当てる際に，スケジュール可能な P_m の利用率の上限（U_{lub}）を計算する（2 行目）．計算式に関しては 4. 節で詳しく述べる．そして，τ_i を P_m に割り当てた場合の利用率 $U(\Lambda)$ が U_{lub} 以下であれば τ_i を P_m に割り当てる（4 行目）．$U(\Lambda) > U_{lub}$ となってしまう場合には，P_m へのタスク割当てを終了する．この時，これ以上のプロセッサが存在しなければアルゴリズムは失敗する（6 行目）．そうでなければ，τ_i を τ'_i と τ''_i に分割する（8 行目）．τ'_i は P_m に，τ''_i は P_{m+1} に各々割り当てられる（9 行目）．本論文では，τ'_i を τ_i の **portion-1**，τ''_i を τ_i の **portion-2** と定義する．ここで，「分割」ということが実際にタスクを 2 つの部分に分けることではないことに注意されたい．τ'_i と τ''_i は，プロセッサ P_m および P_{m+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_m と P_{m+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．

$$C'_i = T_i\{U_{lub} - U(\Lambda_m)\},\ C''_i = C_i - C'_i$$

次に，これ以上タスクがなければアルゴリズムは終了する（13 行目）．そうでなければ，同じ処理を繰り返す（15 行目）．

3.2 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

Preliminary:
the following algorithm schedules the tasks on P_m.
$\tau_i'', \tau_{i+1}, \tau_{i+2}, \ldots, \tau_j'$ are assigned to P_m.

1. when any task is released or is completed on P_m
2. call *schedule_on_*P_m;
3. if the task is τ_j'
4. call *schedule_on_*P_{m+1};

5. **subroutine** *schedule_on_*P_m
6. if τ_i'' is ready
7. if $m > 1$ and τ_i' is being executed on P_{m-1}
8. execute a task with the earliest deadline but τ_i'';
9. else
10. execute τ_i'';
11. else
12. execute a task with the earliest deadline;
13. if the executed task is τ_j' and τ_j'' is being executed on P_{m+1}
14. call *schedule_on_*P_{m+1};

図 2 Ehd2 アルゴリズム

サ P_m でスケジュールするためのアルゴリズムとして Earliest Deadline First with the highest-priority-assigned deferrable portion-2 (Ehd2) を提案する．アルゴリズムを図 2 に示す．

アルゴリズムはプロセッサ P_m に割り当てられたタスクをスケジュールするものとする．タスク τ_i は SIP によるタスク割当て時に τ_i' および τ_i'' に分割され，各々 P_{m-1} および P_m に割り当てられたものとする．また，タスク τ_j も SIP によって τ_j' および τ_j'' に分割され，各々 P_m および P_{m+1} に割り当てられたものとする．まず，あるタスクがリリースされるか実行が完了した場合に，P_m に割り当てられたタスクの再スケジュールを行なう (2 行目)．スケジュール関数は図 2 の 5 行目以降に示す．portion-2 である τ_i'' には常に最高優先度が与えられるが，例外として τ_i'' が実行可能であっても，この時刻に P_{m-1} 上でその portion-1 である τ_i' が実行されていれば，その他のタスクの中で EDF に従ってスケジュールする (6〜10 行目)．τ_i'' が実行可能でない場合も EDF に従ってスケジュールする (11〜12 行目)．ここで，P_m 上で実行を開始したタスクが portion-1 である τ_j' であり，且つその portion-2 である τ_j'' が P_{m+1} 上で実行中の場合には，τ_j'' の実行を中断させるために P_{m+1} 用のスケジュール関数を呼び出す (13〜14 行目)．また，その後 τ_j' の実行が完了した時に，τ_j'' の実行を再開させるために P_{m+1} 用のスケジュール関数を呼び出す (3〜4 行目)．

Ehd2 の特徴は，あらゆるコンテキストスイッチやマイグレーションのタイミングが EDF のスケジュールによって生成できることである．一方で，EKG では，**timea** と **timeb** と呼ばれる特別なタイミングが分割した各タスクに必要になる．その代わり，Ehd2 では P_{m-1} と P_m に分割されたタスクをスケジュールする際に，そのタスクに関して，P_{m-1} 上での portion-1 の実行のために portion-2 の実行を中断・再開する場合があるため，プロセッサ間で同期をとる必要がある．

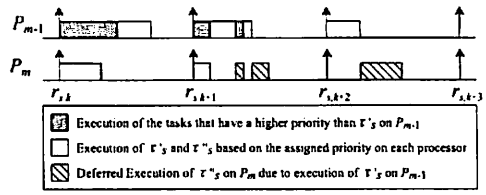

図 3 portion-2 の実行の延期

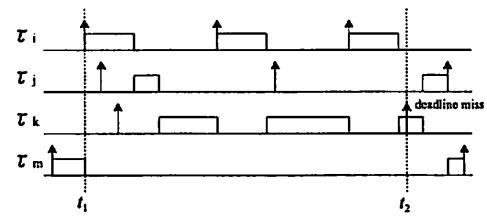

図 4 τ_i に最高優先度を与えた場合の EDF スケジュールの失敗例

4. スケジュール可能性解析

本節では，SIP と Ehd2 を組み合わせたアルゴリズムである Ehd2-SIP のスケジュール可能性判定条件について述べ，スケジュール可能なシステム利用率の上限 (Least Upper Bound) を求める．以降，あるタスク τ_s が τ_s' および τ_s'' に分割され，各々プロセッサ P_{m-1} および P_m に割り当てられたとする．

Ehd2-SIP のスケジュール可能性を解析するためには，τ_s'' がどのようにスケジュールされるかを理解する必要がある．その実行例を図 3 に示す．この例では，τ_s の k 番目のジョブ ($\tau_{s,k}$) がリリースされた際に，P_{m-1} 上では portion-1 ($\tau_{s,k}'$) よりも高い優先度を持ったジョブが実行されているので $\tau_{s,k}'$ の実行は即座には開始されない．一方，portion-2 ($\tau_{s,k}''$) は，P_{m-1} 上で $\tau_{s,k}'$ が実行されてない限り，P_m 上で常に最高優先度が与えられるので即座に実行を開始することができる．そして，図に示すように $\tau_{s,k}'$ と $\tau_{s,k}''$ の実行は重ならないので，$\tau_{s,k}''$ の実行が延期されることはない．次に，ジョブ $\tau_{s,k+1}$ に関しては，$\tau_{s,k+1}$ がリリースされた際に，P_{m-1} 上で $\tau_{s,k+1}'$ よりも高い優先度を持ったジョブの実行が短いために，$\tau_{s,k+1}'$ と $\tau_{s,k+1}''$ の実行が重なる時間が発生する．Ehd2-SIP では，τ_s' と τ_s'' のスケジュールが重なった場合には τ_s' を優先してスケジュールするので，$\tau_{s,k+1}'$ の実行が完了または中断されるまで $\tau_{s,k+1}''$ の実行は延期される．Ehd2-SIP のスケジュール可能上限を求めるためには，τ_s'' の実行が最も長く延期される状況を考慮する必要がある．この状況は図 3 におけるジョブ $\tau_{s,k+2}$ のようにスケジュールされる場合に発生する．すなわち，$\tau_{s,k+2}$ のリリースされた際に，P_{m-1} 上で $\tau_{s,k+2}'$ よりも高い優先度を持ったジョブが存在せず，$\tau_{s,k+2}'$ の実行が即座に開始され，プリエンプトされることなく完了する場合である．言い替えると，$\tau_{s,k+2}''$ の実行が時刻 $r_{s,k+2} + C_s'$ まで延期される場合である．

上述した Ehd2-SIP の性質を考慮に入れ，各プロセッサ上で割り当てられた全てのタスクがデッドラインミスを起こさない

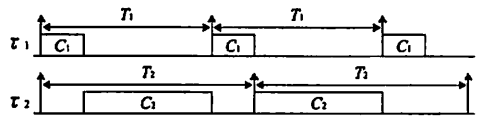

図 5 RM による最悪時の 2 つのタスクのスケジュール

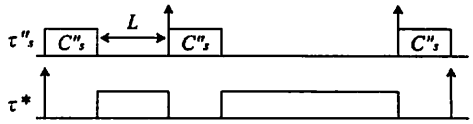

図 6 RM よりスケジュール可能上限が低くなるスケジュール

条件を考える．まず，以下の補題を与える．

補題 1. 任意のプロセッサ P_m において，タスクセット Λ_m の中で最も周期の短いタスク τ_i に静的に最高優先度を与え，その他のタスクを EDF に従ってスケジュールした場合のスケジュール可能上限は，任意の 2 つのタスクを RM に従ってスケジュールした場合のスケジュール可能上限と同一である．

証明． 2 つのタスクに関する RM のスケジュール可能上限を U_{RM} とする．まず，タスク τ_i は最高優先度が与えられているため，$U_i \leqq 1$ である限り τ_i はデッドラインミスを起こすことはない．次に，$U(\Lambda_m) \leqq U_{RM}$ であるにもかかわらず，τ_i 以外の任意のタスクのジョブ $\tau_{k,c}$ がデッドラインミスを起こしたと仮定する．t_1 をプロセッサがアイドル状態もしくは $\tau_{k,c}$ のデッドラインよりも遅いデッドラインを持つジョブを実行していた最後の時刻とする．t_2 を $\tau_{k,c}$ のデッドラインとする．図 4 に例を示す．ここで，時刻 t_2 に $\tau_{k,c}$ がデッドラインミスを起こしたということは，時刻 t_2 よりも遅いデッドラインを持つあらゆるジョブは時間 $[t_1, t_2]$ において実行されないことに注意されたい．時間 $[t_1, t_2]$ において，τ_i 以外の各タスク τ_j がスケジュールされる合計時間 $C_j(t_1, t_2)$ は，

$$C_j(t_1,t_2) = \left\lfloor \frac{t_2-t_1}{T_j} \right\rfloor C_j \leqq \frac{t_2-t_1}{T_j} C_j = (t_2-t_1)U_j$$

と表せる．よって，時間 $[t_1, t_2]$ において，τ_i がスケジュールされる合計時間を $C_i(t_1, t_2)$ とすると，$\tau_{k,c}$ がデッドラインミスを起こす条件は，

$$t_2 - t_1 < C_i(t_1,t_2) + \sum_{\tau_j \in \Lambda_m \setminus \tau_i} (t_2-t_1)U_j$$

となる．この条件式は以下のように書き直すことができる．

$$1 - \frac{C_i(t_1,t_2)}{t_2-t_1} < \sum_{\tau_j \in \Lambda_m \setminus \tau_i} U_j$$

左辺に関して，$C_i \leqq T_i$ および図 4 より，

$$1 - \frac{C_i(t_1,t_2)}{t_2-t_1} \geqq 1 - \frac{2C_i}{T_i+C_i} = \frac{T_i-C_i}{T_i+C_i}$$

と求められるので，$\tau_{k,c}$ がデッドラインミスを起こす条件は式 (1) で表される．

$$\frac{T_i-C_i}{T_i+C_i} < \sum_{\tau_j \in \Lambda_m \setminus \tau_i} U_j \qquad (1)$$

ここで，任意の 2 つのタスクに関する RM のスケジュール可能上限は，2 つのタスクが図 5 の関係にある場合に得られる [6]．すなわち，τ_i 以外のタスクに与えられる最小の実行時間は図 5 における τ_2 の実行時間と等しくなる．仮定より $U(\Lambda_m) \leqq U_{RM}$ であり，図 5 より $C_i/T_i = C_1/T_1$ とおけるので，

$$\sum_{\tau_j \in \Lambda_m \setminus \tau_i} U_j \leqq \frac{C_2}{T_2} = \frac{T_1-C_1}{T_1+C_1}$$

となる．これは，条件式 (1) と矛盾する．よって，$U(\Lambda_m) \leqq U_{RM}$ の条件下では $\tau_{k,c}$ はデッドラインミスを起こさないことになるので，補題は真である． □

プロセッサ P_m 上で τ''_s の実行が延期されることはないと仮定すると，補題 1 より，P_m 上のその他の全てのタスク $\{\Lambda_m \setminus \tau''_s\}$ は RM における 1 つのタスクであるとみなすことができる．よって，この仮定の下では，P_m に割り当てられたタスクセットのスケジュール可能上限はタスク数が 2 であり，且つそのうちの 1 つのタスクの利用率が U''_s である RM のスケジュール可能上限と同一である．これ以降，τ''_s 以外の全タスク，すなわち，サブセット $\Lambda_m \setminus \tau''_s$ を 1 つのタスク τ^* であるとみなし，τ^* の周期を T^*，実行時間を C^*，利用率を U^* と表記する．

補題 2. プロセッサ P_m 上で τ''_s の実行が延期されることはないと仮定すると，P_m に割り当てられたタスクセットの Ehd2-SIP によるスケジュール可能上限は式 (2) である．

$$U_{lub} = U''_s + \frac{1-U''_s}{1+U''_s} \qquad (2)$$

証明． 補題 1 および図 5 より，P_m でのスケジュール可能なプロセッサ利用率の上限を最も低下させる C''_s および C^* は，

$$C''_s = T^* - T_s,\ C^* = 2T_s - T^*$$

と表せる．ここで，$U''_s = \frac{C''_s}{T''_s} = \frac{T^*}{T_s} - 1$ より，

$$U^* = \frac{C^*}{T^*} = \frac{2T_s - T^*}{T^*} = \frac{1-U''_s}{1+U''_s}$$

と書ける．よって，プロセッサ利用率の上限 U_{lub} は，

$$U_{lub} = U''_s + U^* = U''_s + \frac{1-U''_s}{1+U''_s}$$

と表せる． □

次に，τ''_s の実行が延期されないという仮定を取り除くことを考える．τ''_s の実行が延期されるということは，τ''_s は Deferrable Server （DS）[8] のように振る舞うということである．Strosnider らによる DS のスケジュール可能性解析を応用すると，プロセッサ P_m における Ehd2-SIP のスケジュール可能上限が RM のスケジュール可能上限より低くなる可能性があるのは，各タスクが図 6 のようにスケジュールされる場合である．この場合に関して，以下の補題を与える．

補題 3. 図 6 におけるスケジュール可能上限は式 (3) である.

$$U_{lub} = U_s'' + \frac{2 - U_s'' - U_s}{2U_s'' - U_s + 2} \quad (3)$$

証明. まず, 図 6 より, τ^* の実行時間は $C^* = T^* - 3C_s''$ と表せるので, $R_s = T^*/T_s$ とおくとプロセッサ利用率は,

$$U = \frac{C_s''}{T_s} + \frac{C^*}{T^*} = U_s'' + 1 - \frac{3C_s''}{T^*} = U_s'' + 1 - \frac{3U_s''}{R_s}$$

と書ける. 図 6 より T^* は,

$$T^* = T_s + 2C_s'' + L$$

となり, さらに図 3 より $L \geqq T_s - C_s$ であることから,

$$T^* \geqq 2T_s + 2C_s'' - C_s$$

と書ける. ここで, 両辺を T_s で割ると,

$$R_s \geqq 2U_s'' - U_s + 2$$

となるので, U を最小にするのは $R_s = 2U_s'' - U_s + 2$ の場合であることがわかる. よって, スケジュール可能上限 U_{lub} は U_s'' および U_s の関数として,

$$U_{lub} = U_s'' + 1 - \frac{3U_s''}{2U_s'' - U_s + 2} = U_s'' + \frac{2 - U_s'' - U_s}{2U_s'' - U_s + 2}$$

と表せる. □

以上の議論より, 最終的に, Ehd2-SIP のスケジュール可能性に関する以下の定理を導びくことができる.

定理 1. プロセッサ P_m に割り当てられたタスクセットの Ehd2-SIP によるスケジュール可能上限は式 (4) である.

$$U_{lub} = \min\left\{U_s'' + \frac{1 - U_s''}{1 + U_s''},\ U_s'' + \frac{2 - U_s'' - U_s}{2U_s'' - U_s + 2}\right\} \quad (4)$$

証明. 補題 2 および補題 3 より自明である. □

定理 2. M 個のプロセッサが存在するシステムにおいて, 与えられたタスクセット Γ が Ehd2-SIP によってスケジュール可能であるための必要十分条件は式 (5) である.

$$\sum_{\tau_i \in \Gamma} U_i \leqq \left(\sqrt{3} - 1\right) M \quad (5)$$

すなわち, Ehd2-SIP のスケジュール可能なシステム利用率の上限は 73% である.

証明. 式 (4) の最小値を求める. $U_{lub1} = U_s'' + (1 - U_s'')/(1 + U_s'')$, $U_{lub2} = U_s'' + (2 - U_s'' - U_s)/(2U_s'' - U_s + 2)$ とする. RM のスケジュール可能性解析 [6] より, U_{lub1} の最小値は $U_s'' = \sqrt{2} - 1$ の時の $2(\sqrt{2} - 1)$ である. 次に, U_{lub2} の最小値を求める. U_{lub2} を U_s'' に関して偏微分すると,

$$\frac{\partial U_{lub2}}{\partial U_s''} = 1 + \frac{3U_s - 6}{(2U_s'' - U_s + 2)^2}$$

と求められる. $\partial U_{lub2}/\partial U_s'' = 0$ として, U_{lub2} を最小にする U_s'' の極小値を求めると,

$$U_s'' = \frac{U_s - 2 + \sqrt{3(2 - U_s)}}{2} \quad (6)$$

となる. また, U_s に関して, 図 6 より, $L = 0$, すなわち, $U_s = 1$ の時に U_{lub2} が最小になるのは明白である. よって, 上記の式に $U_s = 1$ を代入すると,

$$U_s'' = \frac{\sqrt{3} - 1}{2}$$

となる. よって, U_{lub2} は $U_s = 1$ および $U_s'' = (\sqrt{3} - 1)/2$ の時に最小値 $\sqrt{3} - 1$ となる. $\sqrt{3} - 1 < 2(\sqrt{2} - 1)$ より, あるプロセッサの利用率が $\sqrt{3} - 1$ 以下であれば, そのプロセッサではスケジュール可能であることが保証される. よって, システム利用率が $\sum_{\tau_i \in \Gamma} U_i \leqq (\sqrt{3} - 1)M$ であれば全てのタスクはスケジュール可能であることが保証される. □

5. 評　　価

前節では, Ehd2-SIP の理論的なスケジュール可能性の解析を行ない, スケジュール可能なシステム利用率の上限 (Least Upper Bound) が 73% であることを示した. このスケジュール可能性は, 最適なアルゴリズムを除く既存のあらゆるアルゴリズムのスケジュール可能性よりも高い. 本節では, それら既存のアルゴリズムに対して Ehd2-SIP が, 理論的なスケジュール可能性だけでなく実際のスケジュール可能性の面でも優れていることを示す. 評価指標は以下のスケジュール成功率 (Success Ratio) とする.

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

5.1 シミュレーション環境

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 はプロセッサ数で, 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. システム利用率は U_{total}/M と定義する. 1 つの M, U_{max}, U_{min} の組み合わせ対して, 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合せが考えられるが, 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 $M = 2$, $M = 4$, $M = 8$ 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 Γ は以下のように生成する. $U(\Gamma) \leqq U_{total}$ である限り, 新しいタスクを Γ 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 $[U_{max}, U_{min}]$ の範囲で一様分布によって決定する. 最後に生成されるタスクの利用率のみ, $U(\Gamma) = U_{total}$ となるように調節する. 周期 T_i に関しては, リアルタイムアプリケーションの周期が $1ms$ から $30ms$ 程度であることが多いことを考慮して, [100, 3000] の範囲で無作為に選択する. 最終的に実行時間は $C_i = U_i T_i$ として得られる. 各シミュレーションの時間は $[0, \max\{lcm(\{T_i \mid \tau_i \in \Gamma\}), 2^3 2\})$ とする.

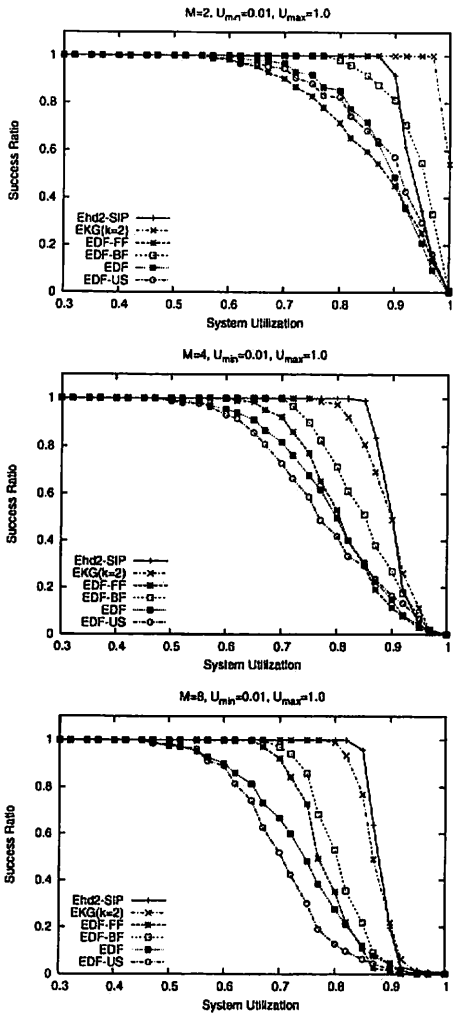

図 7 システム利用率に応じたスケジュール成功率

5.2 シミュレーション結果

シミュレーション結果を図 7 に示す．まず，$M = 2$ の場合には EKG が最も良い性能を示した．シミュレーションでは，EKG の分割タスク数は $k = 2$ とした．そのため，1. 節でも述べた通り $k = M$ の時，EKG は最適なアルゴリズムとなるのでスケジュール成功率は最も高かった．しかしながら，合計のコンテキストスイッチ数は Ehd2-SIP よりも多かった．システム利用率が 100% の時にスケジュール成功率が 100% でなくなるのは，各タスクの周期と実行時間が整数であるために，タスクを分割する際に各プロセッサの利用率が丁度 100% になるように分割できない場合があるからである．Ehd2-SIP は，システム利用率が 87% まではスケジュール成功率 100% を達成できた．一方で，EDF-FF，EDF-BF，EDF，EDF-US はシステム利用率が各々55%，77%，60%，55% までしかスケジュール成功率 100% を維持できなかった．

次に，$M = 4$ の場合には Ehd2-SIP が最も良い性能を示した．Ehd2-SIP は，システム利用率が 82% まではスケジュール成功率 100% を達成できたのに対して，EKG，EDF-FF，EDF-BF，EDF，EDF-US はシステム利用率が各々75%，60%，70%，47%，47% までしかスケジュール成功率 100% を維持できなかった．また，$M = 8$ の場合にも Ehd2-SIP が最も良い性能を示し，システム利用率が 82% まではスケジュール成功率 100% を達成した．一方，EKG，EDF-FF，EDF-BF，EDF，EDF-US はシステム利用率が各々77%，65%，67%，45%，45% までしかスケジュール成功率 100% を維持できなかった．

この結果から，Ehd2-SIP と EKG はプロセッサ数に関係なく高いスケジュール可能性を実現できることがわかった．そして，プロセッサ数が増加すると Ehd2-SIP の方が高い性能を発揮することを確認した．一方で，その他のアルゴリズムはプロセッサ数が増加するとスケジュール可能性が低下していくことがわかった．

6. 結　論

本論文では，高いスケジュール可能性を実現できる実用的な実時間スケジューリングアルゴリズム Ehd2-SIP を提案した．スケジュール可能性解析では，そのスケジュール可能上限が 73% であることを証明した．このスケジュール可能上限は，最適なアルゴリズムを除くと既存のどのアルゴリズムよりも高い．また，シミュレーションによる評価では，実際のスケジュール成功率の面でも Ehd2-SIP は既存のアルゴリズムよりも 5%〜37% 上回っていたことを確認した．

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

文　献

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

[2] S. Baruah, J. Gehrke, and C.G. Plaxton. Fast Scheduling of Periodic Tasks on Multiple Resources. In *Proceedings of the International Parallel Processing Symposium*, pages 280–288, 1995.

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

[4] S. K. Dhall and C. L. Liu. On a Real-Time Scheduling Problem. *Operations Research*, 26:127–140, 1978.

[5] J. Goosens, S. Funk, and S. Baruah. Priority-driven Scheduling of Periodic Task Systems on Multiprocessors. *Real-Time Systems*, 25:187–205, 2003.

[6] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *Journal of the ACM*, 20:46–61, 1973.

[7] J.M. Lopez, J.L. Diaz, and D.F. Garcia. Utlization Bounds for EDF Scheduling on Real-Time Multiprocessor Systems. *Real-Time Systems*, 28:39–68, 2004.

[8] J.K. Strosnider, J.P. Lehoczky, and L. Sha. The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments. *IEEE TRANSACTIONS ON COMPUTERS*, 44:73–91, 1995.