情報処理学会論文誌　コンピューティングシステム　Vol. 1　No. 3　1–13 (Dec. 2008)

マルチプロセッサ用の最適な実時間スケジューリング

船岡健司[†1]　加藤真平[†1]　山崎信行[†1]

本論文では，マルチプロセッサ用の実時間スケジューリング手法である T-N Plane Abstraction（TNPA）を拡張することにより，仕事量を保存してプリエンプション数を削減する拡張 TNPA を提案する．さらに，拡張 TNPA はマルチプロセッサ用の最適な実時間スケジューリングを実現する．拡張 TNPA は，時間分配という手法を利用して任意のタスクに優先的にプロセッサ時間を分配することが可能である．これにより，ある程度までは自由に優先して実行するタスクを選択可能である．評価の結果，拡張 TNPA は TNPA と比較して大幅にタスクのプリエンプション数を削減した．

Optimal Real-time Scheduling on Multiprocessors

Kenji Funaoka,[†1] Shinpei Kato[†1]
and Nobuyuki Yamasaki[†1]

Extended T-N Plane Abstraction (E-TNPA) proposed in this paper realizes work-conserving and efficient optimal real-time scheduling on multiprocessors relative to the original T-N Plane Abstraction (TNPA). E-TNPA leverages the idea of time apportionment. Arbitrary tasks can preferentially receive processor time by time apportionment policies in accordance with various system requirements with several restrictions. E-TNPA optimally solves the problem of scheduling periodic tasks on a multiprocessor system. Simulation results show that E-TNPA significantly reduces the number of task preemptions as compared to TNPA.

1. 序　　論

組み込みシステムには，限られた資源および消費電力量という制約のもと，タスクの実時間性やシステム全体の高いスループットが要求される．このような組み込みシステムに対して，マルチプロセッサアーキテクチャである Simultaneous Multithreading（SMT）や Chip Multiprocessing（CMP）が利用されている．また，汎用コンピュータだけでなく組み込みシステムに対しても信頼性が要求されるようになった．本論文では，マルチプロセッサ用のオペレーティングシステムがスケジューラレベルで信頼性を確保しながら低オーバヘッドで実時間性を保証する手法を提案する．

ロボットなどの時間制約を持つシステムでは，時間制約を守ることができなければシステムとしての価値が著しく低下する．実時間性の保証は，実時間スケジューリングアルゴリズムによって行われる．実時間スケジューリングアルゴリズムによって実時間性を保証可能であることを**スケジュール可能**であるという．実時間スケジューリングアルゴリズムに対する重要な指標の 1 つは，実時間性を保証可能なシステム使用率である．システム使用率が 100%であっても実時間性を保証可能であるとき，その実時間スケジューリングアルゴリズムは**最適**であるという．実時間性を保証可能なシステム使用率が高いと，より性能が低いプロセッサでも実時間性を保証できるようになる．そのようなプロセッサは消費電力が低い傾向にあることから，消費電力量の削減という観点からも優れている．

マルチプロセッサ用の実時間スケジューリングは，最適な手法が長い間発見されなかったため，主に非最適な手法が研究されていた．非最適な手法の多くは，シングルプロセッサ用の最適なアルゴリズムである EDF [1] を基礎としており，EDF-FF [2] や EDZL [3] などがあげられる．これらのアルゴリズムが実時間性を保証可能なシステム使用率は約 50%であることが示されている．理論的に最適な場合と比較すると，2 倍のスループットのプロセッサ，もしくは，2 倍の数のプロセッサが必要になる可能性があり，そのようなプロセッサは消費電力が高い傾向にある．また，基礎となる EDF が優先度駆動型[4] のアルゴリズムであるため，その派生アルゴリズムは信頼性がない．**信頼性がある**スケジューリングアルゴリズムとは，いくつかのタスクが何らかの要因により実行が予測どおりに終了しないときでも，他のタスクに影響を与えないアルゴリズムである．信頼性のないアルゴリズムを利用しているシステムは，脆弱性のあるタスクや最悪実行時間解析を誤ったタスクが存在すると，他のタスクは正常であってもシステム全体が異常動作する危険性がある．

その後の研究により，マルチプロセッサ用の最適な実時間スケジューリングアルゴリズムである PD^2 と EKG，LNREF が示された[5)–8)]．理論的には，これらすべてのアルゴリズムが最適であるが，タスクをスケジュールする手法が異なる．PD^2 は，Pfair [9] という厳しい制約を満たすため，時間を固定長に分割してスケジュールを行う．この分割により頻

†1 慶應義塾大学
Keio University

繁にスケジュール処理を行う必要があるため，大きなオーバヘッドが発生する可能性がある．EKG は，タスクを特定のプロセッサに割り振るパーティショニングに類似した手法を利用している．そのため，特定のプロセッサに負荷が偏ることになる．このような手法では仕事量を保存することができない．スケジューリングアルゴリズムが**仕事量を保存する（work-conserving）**とは，実行可能なタスクが存在するときにプロセッサをアイドル状態にしないというアルゴリズムの特性である．仕事量を保存するアルゴリズムは，同じタスクを連続して実行できる可能性が高まるため，仕事量を保存しないアルゴリズムと比較して，実行時のオーバヘッドを削減することが可能である．注意すべき点は，本論文で扱う仕事量の保存とは，アイドル状態のプロセッサが存在すれば実行可能なタスクを実行することにより必ず仕事量を保存可能という**実装上の仕事量の保存**とは異なるという点である．前述のように同じタスクを連続して実行するためには，理論的に仕事量を保存しなければならない．また，消費電力は，プロセッサ間の負荷を平均化することにより最小化されるため[10]，EKG は消費電力削減という観点からも不利な点が存在する．さらに，EKG は EDF を利用しているため，信頼性がない．LNREF は，PD^2 よりオーバヘッドを削減できる可能性があり，プロセッサを平均的に利用する．

LNREF は，実時間スケジューリングを抽象化手法する T-N Plane Abstraction（TNPA）に基づいている．LNREF は，TNPA の制約により仕事量を保存しないアルゴリズムである．本論文で提案する拡張 TNPA は，仕事量を保存することにより低オーバヘッドな実時間スケジューリングを実現する．

2. システムモデル

システムには，N 個の周期タスクの集合であるタスクセット $T = \{T_1, \ldots, T_N\}$ が存在している．これらのタスクを M 個のプロセッサ $P = \{P_1, \ldots, P_M\}$ 上で実行する．それぞれのタスクは，同時に複数のプロセッサ上で並列実行できない．それぞれのプロセッサは，同時に最大 1 つのタスクを実行可能である．それぞれのタスクは独立しており，いつでもプリエンプション（実行の一時中断）やマイグレーション（タスクのプロセッサ間の移動）が可能である．タスク T_i は，周期 p_i ごとに最悪実行時間 c_i のプロセッサ時間を要求する．タスク T_i のプロセッサ使用率を $u_i = c_i/p_i$（$0 < u_i \leq 1$）と定義する．タスク T_i の相対デッドラインは周期 p_i と等しく，このデッドラインまでに実行を終えなければならない．タスクセット T のプロセッサ使用率を $U = \sum_{T_i \in T} u_i$，タスクセット T のシステム使用率を $U_s = U/M$ と定義する．

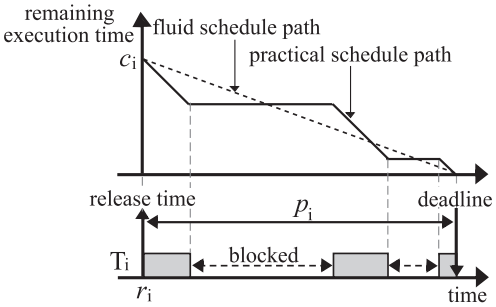

図 1　fluid スケジュールと実際のスケジュール
Fig. 1　Fluid schedule and a practical schedule.

3. T-N Plane Abstraction

LNREF [7] は，最適な実時間スケジューリングアルゴリズムであり，$U_s \leq 1$ を満たすいかなるタスクセットをもスケジュール可能である．理論的に，LNREF でスケジュール不可能なタスクセットは，他のいかなるアルゴリズムをもってしてもスケジュール不可能である．LNREF は，fluid スケジュール[11] の概念を利用して最適なスケジュールを実現する．

図 1 に fluid スケジュールの概念図を示す．下段には実際のシステムでタスク T_i が実行される様子，上段にはタスクの実行によりタスク T_i の残り実行時間が減少する様子が示されている．タスクは到着時刻から実行可能な状態となり，デッドラインまでに残り実行時間をゼロにしなくてはならない．実際のスケジュールでは，ある時刻に 1 つのプロセッサで 1 つのタスクしか実行することができない．そのため，図の下段のようにタスクの実行がブロックされる可能性がある．fluid スケジュールは，つねに一定の割合でタスクの実行を行い，デッドラインにおいて残り実行時間をゼロにする，概念的に最適なスケジュール手法である．図中の fluid スケジュールによって残り実行時間が減少する様子を示す線分を fluid スケジュールパスと呼ぶ．これを実現するには，1 つのプロセッサで複数のタスクを完全に同時実行しなくてはならないことから，実現は不可能である．

LNREF は，全タスクのデッドラインごとに fluid スケジュールパスを追跡する．**図 2** 上段は，図 1 上段を複数タスクかつ複数周期にまで広げて示したものである．図 2 の垂直方向の破線で示したように，全タスクのデッドラインごとに時間を分割する．このデッドラインで分割された各時間区間を**ノード**と呼ぶ．重要な点は，ノードの右端において，全タスクの

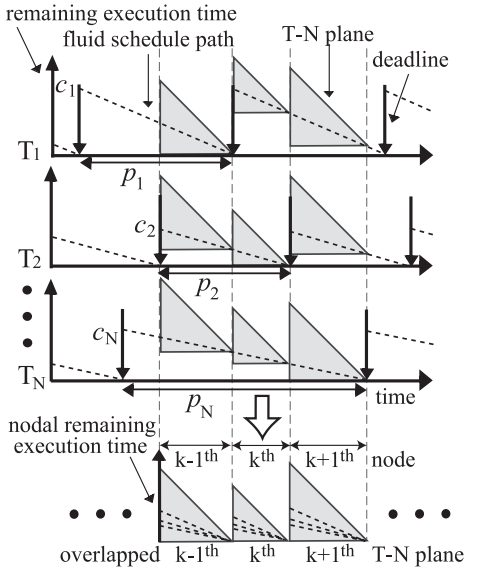

図 2　T-N Plane による実時間スケジューリングの抽象化
Fig. 2　T-N Plane Abstraction for real-time scheduling.

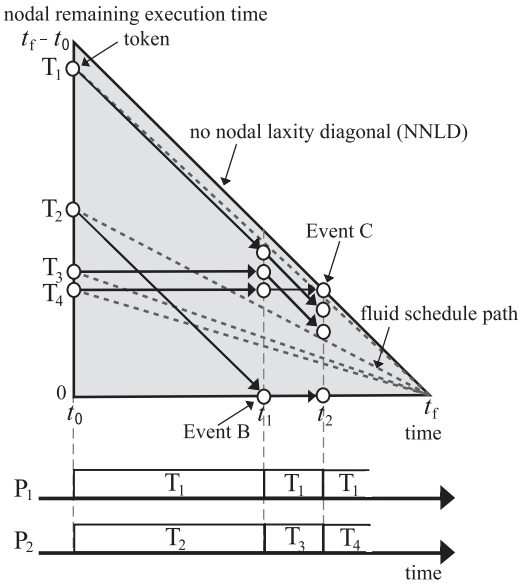

図 3　LNREF によるスケジューリング
Fig. 3　LNREF scheduling.

残り実行時間を fluid スケジュールの値以下にすることにより，全デッドラインを守ることが可能であるという点である．LNREF では，この考え方を T-N Plane（Time and Nodal remaining execution time domain Plane）と呼ばれる直角二等辺三角形によって実現する．図 2 上段に示したように，T-N Plane の二等辺は各軸に平行であり斜辺の傾きは -1 となるように，かつ，T-N Plane の右頂点が fluid スケジュールパスと重なるように T-N Plane を配置する．このとき，この T-N Plane の右頂点を目指してタスクを実行することにより，タスクの残り実行時間を fluid スケジュールの値に合わせる．同じノード内の T-N Plane は合同であることから，図 2 下段のように同じノード内の T-N Plane を重ね合わせることにより，あるノードでは単一の T-N Plane の中でのタスクのスケジュールを考えればよい．この重ね合わせた T-N Plane の中で，T_i を実行しなければならない残り時間をタスク T_i のノード内の残り実行時間（nodal remaining execution time）という．重ね合わせた T-N Plane は，横軸に時刻，縦軸にタスクのノード内の残り実行時間を示している．

図 3 上段に，図 2 下段で示した時刻 t_0 から時刻 t_f の間に存在する 1 つの重ね合わせた T-N Plane を示す．それぞれのタスクは，T-N Plane の中で現在の時刻とそのタスクのノード内の残り実行時間を表すトークンとして表される．時刻 t_0 において，各トークン T_i は対応するタスクの fluid スケジュールパスと T-N Plane の左辺の交点 $(t_0, u_i(t_f - t_0))$ に存在する．ノード内の残り実行時間は，タスクを実行するとトークンが傾き -1 の右下方向に移動して減少する．タスクを実行しないとトークンが傾き 0 の右方向に移動して変化しない．全トークンが T-N Plane の右頂点まで到達可能であることを**ノード内でスケジュール可能**であるという．ノード内でスケジュール可能であることを実現するために，LNREF は複数のイベントごとにノード内の残り実行時間の大きい最大 M 個のタスクを実行する（Largest Nodal Remaining Execution time First）．そのイベントとは，タスクの到着（T-N Plane の左端）と後述するイベント B，イベント C である．イベント B は，トークンが T-N Plane の底辺に到達した際に発生する．イベント C は，トークンが T-N Plane の斜辺（NNLD）に到達した際に発生する．これらのイベントごとに実行するタスクの選択を繰り返すことにより，$U_s \leq 1$ であれば LNREF は全タスクのデッドラインを守ることが可能である．イベ

ント B を発生させたタスクを実行することはできないため，TNPA の LNREF は仕事量を保存しない．イベント B を発生させたタスクでも，プロセッサがアイドル状態になれば実行可能という点については，1 章で示した実装上の仕事量の保存であるため，議論の対象外である．実装上の仕事量の保存では，各トークンが T-N Plane の底辺で実行を一時中断しなければならないため，同じタスクを連続実行することが難しい．また，理論的にはタスクの実行が終了するときに実際は実行が終了していなくても，イベント B で強制的に実行が終了させられるため，他のタスクの実行時間を奪ってしまうということはない．したがって，実行が終了しないタスクが他のタスクの実行に影響を与えないため，LNREF には信頼性がある．

図 3 を利用して LNREF のスケジュール例を示す．図 3 上段には，T-N Plane の中でトークンが動く様子，下段にはその結果によりタスクを実行する様子を示した．図のように 4 つのタスク（$T_1, \ldots, T_4$）を 2 つのプロセッサ（P_1，P_2）上で実行するシステムを想定する．プロセッサ数が 2 であるため，同時に 2 つのタスクを実行可能である．時刻 t_0 では，すべてのタスクが fluid スケジュールパス上に存在する．ここで，ノード内の残り実行時間の大きい 2 個のトークン（T_1，T_2）を選択することにより対応するタスクを実行する．時刻 t_1 まで実行すると，トークン T_2 が三角形の底辺まで到達することにより，イベント B が発生する．同様に，この段階でノード内の残り実行時間の大きな T_1 と T_3 を実行する．さらに実行を進めると，時刻 t_2 にタスク T_4 が三角形の斜辺まで到達して，イベント C が発生して T_1 と T_4 を選択する．これを繰り返すことにより，トークンを T-N Plane の右頂点まで導く．

4. 拡張 T-N Plane Abstraction

拡張 TNPA（E-TNPA）は，T-N Plane の上下の位置を制御することにより，仕事量を保存する最適な実時間スケジューリングを実現する．TNPA は各ノード内で各タスク T_i に一定のノード内の残り実行時間 $u_i(t_f - t_0)$ を与えるのに対して，拡張 TNPA ではいくつかの制約のうえでノード内の残り実行時間を自由に変更可能である．TNPA との相異点は，T-N Plane の上下の位置が制御されるか否かという点であり，スケジューリング自体は同じ概念に基づいて行われる．

T-N Plane の上下の位置を制御するということを厳密に表現するため，拡張 TNPA 用の T-N Plane を**図 4** と**図 5** のような台形で表現する．この T-N Plane は，タスク T_i の T-N Plane を表しており，同じノード内であっても他のタスクの T-N Plane とは合同ではない可能性がある．各タスク T_i には，a_i で表されるノード内の追加時間（additional nodal time）が与えられる．ノード内の追加時間とは，各ノードで余った時間をタスクに追加するということを意味している．T-N Plane は，TNPA の T-N Plane の下に高さ $|a_i|$ の長方形があわさる形であり，左右の辺が平行な台形である．$a_i \geq 0$ のとき，fluid スケジュールパスは，図 4 のように T-N Plane の右上の頂点を通過する．$a_i < 0$ のとき，図 5 のように T-N Plane の右下の頂点を通過する．トークンの初期位置は，fluid スケジュールパスと T-N Plane の左端の交点 $(t_0, u_i(t_f - t_0) + a_i)$ である．TNPA と同様の概念に基づいてスケジュールを行うため，**仮想 T-N Plane** と呼ばれる直角二等辺三角形を利用する．仮想 T-N Plane は，図 4 と図 5 の影付きの直角二等辺三角形のように，$a_i \geq 0$ のときは T-N Plane の下側に，$a_i < 0$ のときは上側に配置する．トークンが仮想 T-N Plane の斜辺に当たったときはイベント C，底辺に当たったときはイベント B が発生する．全トークンが仮想 T-N Plane の右頂点 $(t_f, 0)$ に到達可能であることを**ノード内でスケジュール可能**であるという．仮想 T-N Plane は，同じノード内では合同であるため，TNPA と同様のスケジュー

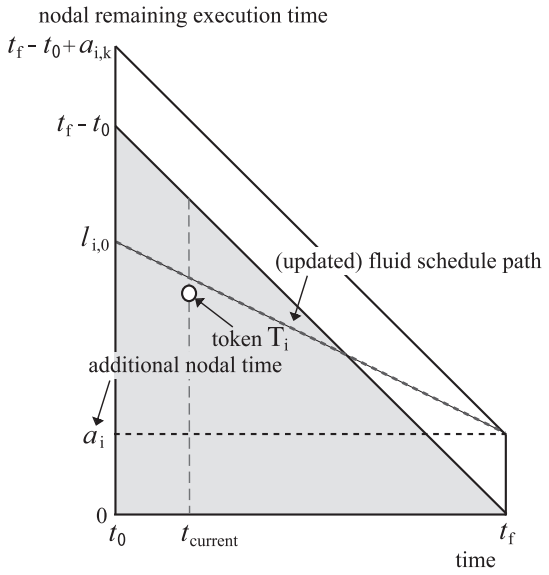

図 4　拡張 T-N Plane（$a_i \geq 0$）
Fig. 4　An extended T-N plane ($a_i \geq 0$).

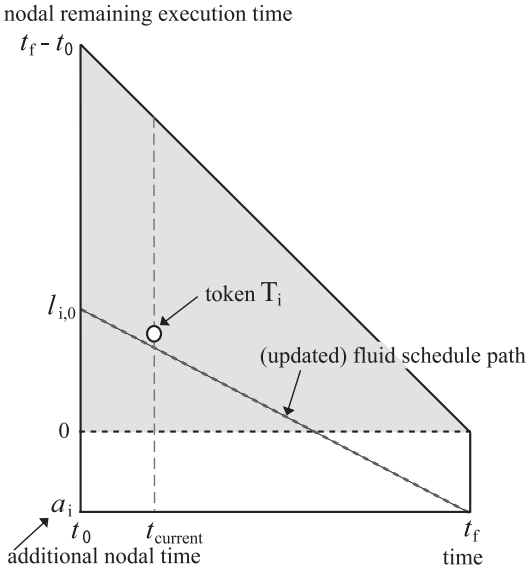

図 5　拡張 T-N Plane（$a_i < 0$）
Fig. 5　An extended T-N plane ($a_i < 0$).

ルを行うことが可能である．したがって，ノード内の追加時間 a_i を注意深く割り当てることにより，全タスクのデッドラインを守ることが可能となる．

TNPA と拡張 TNPA の簡単な比較を**図 6** に示す．図 6 はタスク T_i の連続した T-N Plane を図示している．TNPA の T-N Plane の右頂点はつねに fluid スケジュールパス上に存在するのに対して，拡張 TNPA の仮想 T-N Plane は異なる位置に存在する．具体的な仮想 T-N Plane の位置を決定するアルゴリズムの説明は後の節で行うため，以下ではその概略についてのみ説明する．拡張 TNPA は，現在のノードにおいて，タスク T_i のノード内の追加時間が $a_i = 0$ でも実行を終了するとき，そのタスクは負 $a_i < 0$ のノード内の追加時間が与えられる．そうではないとき，正 $a_i > 0$ のノード内の追加時間が与えられるが，追加可能な時間がないときは $a_i = 0$ となる．正のノード内の追加時間 $a_i > 0$ を与えられたタスクが T-N Plane の右頂点に到達したとき，図 6 のように，fluid スケジュールパスは，T-N Plane の右下頂点から元の fluid スケジュールパスと平行になるように更新される．したがって，次のノードの仮想 T-N Plane は，TNPA の T-N Plane よりも下側に位置する

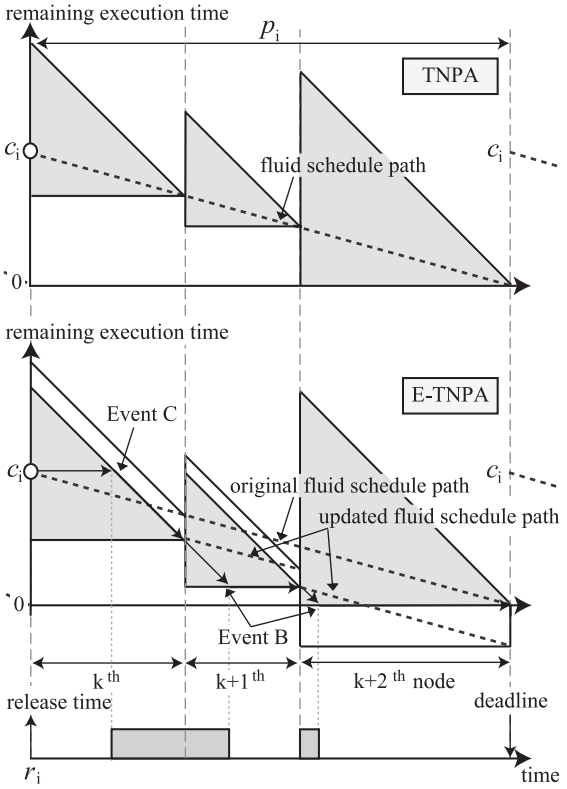

図 6　TNPA と拡張 TNPA の比較
Fig. 6　A comparison between TNPA and E-TNPA.

ことになる．図中の $k+2$ 番目のノードの T-N Plane が負のノード内の追加時間 $a_i < 0$ を得ているのは，$a_i = 0$ でも実行を終了可能であるためである．この負のノード内の追加時間は，他のタスクによって利用される．

拡張 TNPA で利用する概念について定義を行う．時刻 t_0 から t_f に存在する重ね合わせた仮想 T-N Plane において，j 番目のイベントが発生した時刻を t_j と仮定する．ただし，時刻 t_0 はタスクの到着時刻（T-N Plane の左端）である．時刻 t_j のタスク T_i について，残り実行時間を $e_{i,j}$，ノード内の残り実行時間を $l_{i,j}$，ノード内のプロセッサ使用率を

$r_{i,j} = l_{i,j}/(t_f - t_j)$ と定義する．時刻 t_j における全タスクのノード内のプロセッサ使用率の和を $S_j = \sum_{T_i \in T} r_{i,j}$ と定義する．

4.1　スケジュール可能性

拡張 TNPA は，各タスクがノード内でスケジュール可能であるためのノード内の追加時間 a_i の条件を満たすと同時に，スケジュール可能であるための a_i の条件を満たすことにより，全タスクの実時間性を保証する．TNPA ではノード内でスケジュール可能であればスケジュール可能であったのに対して，拡張 TNPA ではノード内でスケジュール可能であってもスケジュール可能ではない可能性がある．すなわち，必ず仮想 T-N Plane の右頂点に到達していたとしても，たとえば図 6 の $j+2$ 番目のノードの仮想 T-N Plane が図の位置よりも上に存在していたら実時間性を保証できないため，そのようなことがないように a_i を決定する．本節では，スケジュール可能であるための条件を示すために，k 番目のノードの T-N Plane を考える．

初めに，個々のタスクがノード内でスケジュール不可能ではないための条件を示す．時刻 t_0 に仮想 T-N Plane の斜辺より上に存在しているトークンは，いかなるスケジュールを行ってもノード内でスケジュール可能とすることができない．定理 1 は，そこから導かれるノード内の追加時間の最大値を示している．

定理 1（ノード内の追加時間の最大値）． 時刻 t_0 に全トークンが仮想 T-N Plane の斜辺より上に存在しないためのノード内の追加時間の条件は，$\forall i : a_i \le (1-u_i)(t_f - t_0)$ である．

証明． 仮想 T-N Plane の高さは $t_f - t_0$ である．トークンがこれより下に存在するためには，

$$\begin{aligned} & \forall i : l_{i,0} \le t_f - t_0 \\ \Leftrightarrow\ & \forall i : u_i(t_f - t_0) + a_i \le t_f - t_0 \\ \Leftrightarrow\ & \forall i : a_i \le (1-u_i)(t_f - t_0) \qquad (1) \end{aligned}$$

とならなければならない．□

次に，定理 1 の条件を満たしたタスクがノード内でスケジュール可能であるための条件を示す．Cho ら[7] は，Critical Moment が TNPA におけるノード内でスケジュール可能であるための必要十分条件であることを示した．拡張 TNPA において，Critical Moment とは，**図 7** のように M 個を超えるトークンが仮想 T-N Plane の斜辺に到達することである．Critical Moment が発生すると実行できないタスクが仮想 T-N Plane から飛び出してしまう．定理 2 は，拡張 TNPA で Critical Moment が発生しないための条件を示す．

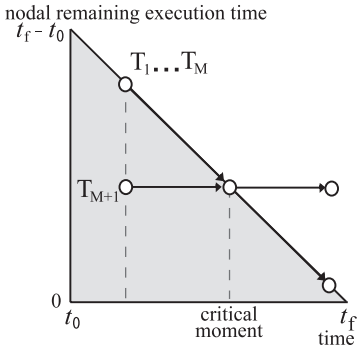

図 7　Critical Moment
Fig. 7　Critical Moment.

定理 2（Critical Moment が発生しないための条件）． Critical Moment が発生しないための条件は，すべての j について $S_j \le M$ である．

証明． TNPA と拡張 TNPA は同じ概念でスケジュールを行うため，TNPA の証明[7] と同様である．□

Critical Moment を発生させないための条件として，ノード内でスケジュール可能であるためのノード内の追加時間の和の条件を定理 3 に示す．定理 3 では，時刻 t_0 の条件 $S_0 \le M$ を示している．これ以降の条件 $^{\forall}j > 0 : S_j \le M$ は LNREF によって満たされる．

定理 3（ノード内の追加時間の和の最大値）． $S_0 \le M$ と $A \le (M-U)(t_f - t_0)$ は同値である．

証明．

$$\begin{aligned} & S_0 \le M \\ \Leftrightarrow\ & \sum_{T_i \in T} r_{i,0} - \sum_{T_i \in T} u_i \le M - \sum_{T_i \in T} u_i \\ \Leftrightarrow\ & \sum_{T_i \in T} \left(\frac{l_{i,0}}{t_f - t_0} - u_i \right) \le M - U \\ \Leftrightarrow\ & \sum_{T_i \in T} (l_{i,0} - u_i(t_f - t_0)) \le (M-U)(t_f - t_0) \end{aligned}$$

$$\Leftrightarrow \sum_{T_i \in T} a_i \leq (M-U)(t_f - t_0) \quad (2)$$

□

ノード内の追加時間が条件 (1) と (2) を満たすことが可能であれば，ノード内でスケジュール可能となる．

最後にスケジュール可能であるための条件を示す．本論文では，仮想 T-N Plane が TNPA の T-N Plane と同じか下の位置に存在することにより，スケジュール可能であることを確立する．定理 4 は，スケジュール可能であるための仮想 T-N Plane の位置から導かれるノード内の追加時間の最小値を示している．

定理 4（ノード内の追加時間の最小値）． $\forall i : a_i \geq e_{i,0} - c_i + u_i(t_0 - r_i)$ のとき，仮想 T-N Plane は TNPA の T-N Plane と同じか下の位置に存在する．

証明． 仮想 T-N Plane の右頂点に着目する．仮想 T-N Plane の右頂点の高さは $e_{i,0} - u_i(t_f - t_0) - a_i$ であり，TNPA の T-N Plane の右頂点の高さは $c_i - u_i(t_f - r_i)$ である．ただし，r_i はタスク T_i の到着時刻とする．仮想 T-N Plane と TNPA の T-N Plane の位置関係は，

$$c_i - u_i(t_f - r_i) \geq e_{i,0} - u_i(t_f - t_0) - a_i$$
$$\Leftrightarrow a_i \geq e_{i,0} - c_i + u_i(t_0 - r_i) \quad (3)$$

であればよい．□

これまでの条件 (1)，(2)，(3) を満たすことにより，拡張 TNPA は，全タスクの実時間性を保証可能となる．

4.2 時 間 分 配

図 8 に前節で示したスケジュール可能であるための条件を満たすアルゴリズム ApportionTime を示す．ApportionTime は，2 行目で TNPA によって利用されないプロセッサ時間 $(M-U)(t_f - t_0)$ を変数 L' に回収する．次に，4 行目で各タスク T_i のノード内の残り実行時間を一時的に TNPA と同じ $u_i(t_f - t_0)$ とする．5 から 9 行目において，残り実行時間よりもノード内の残り実行時間が多いタスクの追加時間を変数 L' に回収する．回収された時間 L' は，14 から 25 行目で時間の足りないタスクに分配される．全タスクは 7 か 18 か 20 行目を必ず通過する．7 行目を通過したタスクは，$a_i = 0$ でもノード内で終了するタスクである．18 行目を通過したタスクは，十分な時間を与えればノード内で終了するタスクである．20 行目を通過したタスクは，そのノード内では終了しないタスクである．仕事量の保存の証明に利用するため，26 行目でタスクに分配できなかった時間を L に代入する．

```
Algorithm: ApportionTime
 1: when time t_0 in each node
 2:   L' = (M - U)(t_f - t_0)
 3:   foreach 1..N as i
 4:     l_{i,0} = u_i(t_f - t_0)
 5:     if e_{i,0} ≤ l_{i,0}
 6:       // retrieve time (a_i ≤ 0)
 7:       a_i = e_{i,0} - l_{i,0}
 8:       l_{i,0} = l_{i,0} + a_i
 9:       L' = L' - a_i
10:     else
11:       a_i = NULL
12:     end if
13:   end foreach
14:   foreach 1..N as i
15:     if a_i is NULL
16:       // apportion time (a_i ≥ 0)
17:       if e_{i,0} ≤ t_f - t_0
18:         a_i = min{e_{i,0} - l_{i,0}, L'}
19:       else
20:         a_i = min{(t_f - t_0) - l_{i,0}, L'}
21:       end if
22:       l_{i,0} = l_{i,0} + a_i
23:       L' = L' - a_i
24:     end if
25:   end foreach
26:   L = L' // leftover time for proofs
27: end when
```

図 8　アルゴリズム ApportionTime
Fig. 8　Algorithm ApportionTime.

ApportionTime について着目すべき点は，任意のタスクに優先して時間を分配可能であるという点である．なぜなら，時間分配の順序はタスクのパラメータに依存しないタスクの番号に基づいている．優先的に実行したいタスクには，各ノードで小さいタスク番号を与えることにより優先的に時間を与えることが可能である．拡張 TNPA と ApportionTime の組合せでは，オーバヘッドを削減するために，残り実行時間が小さいタスクに優先的に時間を与えることにより，実行可能なタスクを早く減らすことが可能である．

ApportionTime は，スケジュール可能であるための条件式 (1)，(2)，(3) を満たすことを以下の定理に示す．

定理 5（ApportionTime によるノード内の追加時間の最大値）． ApportionTime によるノード内の追加時間は $\forall i : a_i \leq (1-u_i)(t_f - t_0)$ を満たす．

証明． 7 行目を通過したタスクは，$a_i \leq 0$ と $(1-u_i)(t_f-t_0) \geq 0$ より式 (1) を満たすことは明らか．18 行目を通過したタスクも以下のように条件を満たす．

$$
\begin{aligned}
& (1-u_i)(t_f-t_0) - a_i \\
=\ & (1-u_i)(t_f-t_0) - (e_{i,0} - l_{i,0}) \\
\Downarrow\ & e_{i,0} \leq t_f - t_0 \text{ と } l_{i,0} = u_i(t_f - t_0) \text{ より} \\
\geq\ & (1-u_i)(t_f-t_0) - (t_f-t_0) + u_i(t_f-t_0) = 0 \\
\Rightarrow\ & (1-u_i)(t_f-t_0) \geq a_i
\end{aligned}
$$

20 行目を通過したタスクは，18 行目を通過したタスクと同様である． □

定理 6（ApportionTime によるノード内の追加時間の和の最大値）． ApportionTime によるノード内の追加時間の和は $A \leq (M-U)(t_f - t_0)$ を満たす．

証明． ノード内の追加時間に分配される L' は 2 行目で $(M-U)(t_f-t_0)$ に初期化される．また，7 行目を通過したタスクは非正の追加時間が与えられ，その時間は L' に回収される．18 と 20 行目を通過するタスクについて，ノード内の追加時間の最大値は L' に制限されている．したがって，ノード内の追加時間の和は，最大でも $(M-U)(t_f-t_0)$ である． □

定理 7（ApportionTime によるノード内の追加時間の最小値）． ApportionTime によるノード内の追加時間は $\forall i : a_i \geq e_{i,0} - c_i + u_i(t_0 - r_i)$ を満たす．

証明． 7 か 18 行目を通過するタスクは，以下のとおり条件を満たす．

$$
\begin{aligned}
& a_i - (e_{i,0} - c_i + u_i(t_0 - r_i)) \\
=\ & (e_{i,0} - u_i(t_f - t_0)) - (e_{i,0} - c_i + u_i(t_0 - r_i)) \\
=\ & -u_i t_f + c_i + u_i r_i \\
=\ & c_i - (c_i/p_i)(t_f - r_i) \\
=\ & c_i\,(1 - (t_f - r_i)/p_i) \\
\Downarrow\ & p_i \geq t_f - r_i \text{ より} \\
\geq\ & 0 \\
\Rightarrow\ & a_i \geq e_{i,0} - c_i + u_i(t_0 - r_i)
\end{aligned}
$$

20 行目を通過するタスクは，以下のとおりである．このタスクは，現在のノード内では実行を終えないため，時刻 r_i から時刻 t_f の間は，$a_i \geq 0$ となるノード内の追加時間を与えられている．そのため，その間の T-N Plane の形は図 4 に示されたように，仮想 T-N Plane の右頂点は，fluid スケジュールパスよりつねに下側に位置している．そのため，定理 4 の証明と同様に $a_i \geq e_{i,0} - c_i + u_i(t_0 - r_i)$ を満たす． □

4.3　仕事量の保存

本節では，拡張 TNPA に基づくスケジューリングが仕事量を保存していることを示す．そのために，残り実行時間が正であることを**実行可能**，ノード内の残り実行時間が正であることを**ノード内で実行可能**であると定義する．TNPA が仕事量を保存できない原因は，3 章で解説したとおり，イベント B を発生させたタスクを実行できないことにある．いい換えると，実行可能であるにもかかわらず，ノード内で実行可能ではないタスクが存在する可能性がある．そのため，拡張 TNPA では，任意の時刻 t_j に実行可能なタスクを最大 M 個まで必ず実行可能であることを示す．

仕事量の保存を示すために以下の補題を利用する．

補題 8（ノード内のプロセッサ使用率の和の単調減少性）． S_j は単調減少である．

証明． S_{j-1} と S_j を比較して $S_{j-1} \geq S_j$ であることを示す．そのため，$S_{j-1} = S$（$\leq M$）とおく．

$$
\begin{aligned}
& S_{j-1} = S \\
\Leftrightarrow\ & \sum_{T_i \in T} \frac{l_{i,j-1}}{t_f - t_{j-1}} = S \\
\Leftrightarrow\ & \sum_{T_i \in T} l_{i,j-1} = S(t_f - t_{j-1}) \qquad (4)
\end{aligned}
$$

ここで N'（$\leq M$）個のタスクを時刻 t_{j-1} から t_j の間で実行すると仮定する．すべてのトークンはノード内でスケジュール可能であるため，仮想 T-N Plane の中に存在している．したがって，$S \leq M$ と $r_{i,j-1} \leq 1$ より $S \leq N'$ が導かれる．時刻 t_{j-1} から t_j の間に，残り実行時間の和は $N'(t_j - t_{j-1})$ だけ減少する．以上より，S_j は以下のように算出される．

$$
S_j = \frac{1}{t_f - t_j} \sum_{T_i \in T} l_{i,j}
$$

$$= \frac{1}{t_f - t_j}\left(\left(\sum_{T_i \in T} l_{i,j-1}\right) - N'(t_j - t_{j-1})\right)$$

$\Downarrow$ 式 (4) より

$$= \frac{S(t_f - t_{j-1}) - N'(t_j - t_{j-1})}{t_f - t_j} \tag{5}$$

S_{j-1} と S_j の関係は,

$$\begin{aligned} & S_{j-1} - S_j \\ =\ & S - \frac{S(t_f - t_{j-1}) - N'(t_j - t_{j-1})}{t_f - t_j} \\ =\ & \frac{t_j - t_{j-1}}{t_f - t_j}(N' - S) \geq 0 \\ \Rightarrow\ & S_{j-1} \geq S_j \end{aligned}$$

となる. □

次に，スケジュール可能な限界の状態を考える．任意の時刻 t_j から t_f の間のプロセッサ時間の和は $M(t_f - t_j)$ である．ノード内の残り実行時間の和が $M(t_f - t_j)$ のとき，実時間性を保証するためにはプロセッサ時間を無駄にできない．これを最大負荷状態と呼ぶ．このとき，ノード内のプロセッサ使用率は,

$$S_j = \sum_{T_i \in T} \frac{l_{i,j}}{t_f - t_j} = \frac{M(t_f - t_j)}{t_f - t_j} = M$$

である．$S_j = M$ となる j が存在するとき，そのノード内において，ノード内のプロセッサ使用率の和はつねに M となり最大負荷状態であることを以下に示す.

補題 9（最大負荷状態以降のノード内のプロセッサ使用率）. $\exists j : S_j = M$ ならば, $\forall j : S_j = M$ である.

証明. $S_{x-1} = M$ を満たす整数 x が存在すると仮定する．⟨A⟩ 式 (4) と (5) の S に M を代入すると，S_x も M となる．したがって，$j > x-1$ について $S_j = M$ である．⟨B⟩ 補題 8 より S_j は単調減少であるため，$^{\forall}j < x-1 : S_j \geq M$ である．また，拡張 TNPA の中で LNREF は最適であることから Critical Moment は発生しないため，$^{\forall}j < x-1 : S_j \leq M$ である．$^{\forall}j < x-1$ について，$S_j \geq M$ と $S_j \leq M$ より，$S_j = M$ が導かれる．⟨A⟩ と ⟨B⟩ より，$\exists j : S_j = M$ ならば，$\forall j : S_j = M$ である. □

S_0 と A の関係は以下のとおりである.

補題 10（最大負荷時のノード内の追加時間の和）. $A = (M-U)(t_f - t_0)$ と $S_0 = M$ は同値である.

証明. 定理 3 の等号成立時による. □

システムが最大負荷状態のとき $L = 0$ であることを示す．ノード内の追加時間の和の最大値は，定理 3 より $(M-U)(t_f - t_0)$ である．タスクに分配できなかった時間 L は，$L = (M-U)(t_f - t_0) - A$ となる.

補題 11（最大負荷状態時の余り時間）. $\exists j : S_j = M$ と $L = 0$ は同値である.

証明.

$\exists j : S_j = M$

$\Downarrow$ 補題 9 と 10 より

$\Leftrightarrow A = (M-U)(t_f - t_0)$

$\Downarrow L = (M-U)(t_f - t_0) - A$ より

$\Leftrightarrow L = 0$

□

拡張 TNPA は仕事量を保存していることを示す.

定理 12（仕事量の保存）. 拡張 TNPA は仕事量を保存する.

証明. 時刻 t_0 における状態を ⟨A⟩, ⟨B-1⟩, ⟨B-2⟩ に場合分けして，すべての実行可能なタスクを最大 M 個まで必ず実行可能であることを示す．⟨A⟩ $S_0 = M$ のとき，補題 9 より $\forall j : S_j = M$ である．また，拡張 TNPA は最適であるためトークンが T-N Plane の外に飛び出ることはないことから $\forall i, j : r_{i,j} \leq 1$ である．任意の時刻 t_j について $S_j = M$ と $\forall i : r_{i,j} \leq 1$ より少なくとも M 個は $l_{i,j} > 0$ となるタスクが必ず存在する．全タスクは図 8 の ApportionTime の 7 か 18 か 20 行目を通過するが，各 a_i の式を $l_{i,0}$ の式に代入することにより，どこを通過しても $l_{i,0} \leq e_{i,0}$ であることは明らかである．残り実行時間とノード内の残り実行時間の減少量は同じであるため，任意の時刻 t_j について少なくとも M 個は $e_{i,j} > 0$ となるタスクが存在する．よって，つねに M 個の終了していないタスクが存在しており実行可能である．⟨B-1⟩ $S_0 < M$ かつ $L = 0$ の状態は，補題 9 と 11 より

$L = 0 \Leftrightarrow \forall j : S_j = M$ であるため存在しない．⟨B-2⟩ $S_0 < M$ かつ $L \neq 0$ のときを示す．ApportionTime の 7 行目を通過するタスクは，現在のノードで実行を終了するため，18 行目と 20 行目を通過するタスクのみを考えればよい．$L \neq 0$ であるため，ノード内の追加時間は 18 と 20 行目において max の L' に制限されない．したがって，これらのタスクは，22 行目において最大のノード内の残り実行時間 $e_{i,0}$ と $t_f - t_0$ をそれぞれ受け取ることになる．18 行目を通過するタスクは，現在のノードで実行を終了する．20 行目を通過するタスクは，つねに仮想 T-N Plane の斜辺上に存在して実行される．⟨A⟩，⟨B-1⟩，⟨B-2⟩ より，拡張 TNPA では最大 M 個の実行可能な全タスクを必ず実行可能である．□

4.4 T-N Plane の更新

これまでは理論上の議論を行ってきたが，本節では実際のシステムでの拡張 TNPA について議論する．拡張 TNPA は，U が M に近いとき，理論的には分配可能な時間が少ないため，TNPA に近いスケジューリングを行い，有効性が著しく低い．また，前節において拡張 TNPA は理論的に仕事量を保存することを示したが，実際のシステム上では仕事量を保存しない．これは，c_i が最悪実行時間を意味するために，理論よりも早く実行が終了する可能性があるためである．これらの問題を解決するためにノード内の追加時間の再分配を行う．時間の再分配を行うことにより，タスクの実行時間の変動に対応することが可能となる．そのため，たとえ U が M に近いために分配可能な時間が少なくても，タスクが早期終了した時間を分配することにより，オーバヘッドを削減することが可能となる．

ノード内の追加時間の再分配アルゴリズム ReapportionTime を**図 9** に示す．**図 10** のようにタスク T_c が時刻 $t_{j'}$ に早期終了したと仮定すると，余った時間 $l_{c,j'}$ は，タスク T_c に保持されているため，他のタスクは利用することができない．そのため，早期終了時に ReapportionTime を実行して余った時間を再分配する．ReapportionTime は ApportionTime と同じ手法を利用して時間分配を行っている．ApportionTime と ReapportionTime の相異点は，ReapportionTime が早期終了したタスクの時間のみを分配するということである．早期終了時に ReapportionTime を実行することにより，全タスクはノード内でスケジュール可能であり，拡張 TNPA は実際のシステムでも仕事量を保存することが可能であることを定理 13 に示す．

定理 13（ReapportionTime による理論と実際の補正）． タスクの早期終了時に ReapportionTime を行うと，全タスクはノード内でスケジュール可能であり，かつ，仕事量を保存する．

Algorithm: ReapportionTime

```
1: when T_c completes at time t_{j'}
2:     // retrieve the time of the task T_c
3:     L' = l_{c,j'}
4:     foreach 1..N as i
5:       if e_{i,j'} > l_{i,j'}
6:         // apportion time (Δ ≥ 0)
7:         if e_{i,j'} ≤ t_f − t_{j'}
8:           Δ = min{e_{i,j'} − l_{i,j'}, L'}
9:         else
10:          Δ = min{(t_f − t_{j'}) − l_{i,j'}, L'}
11:        end if
12:        α_i = α_i + Δ
13:        l_{i,j'} = l_{i,j'} + Δ
14:        L' = L' − Δ
15:      end if
16:    end foreach
17: end when
```

図 9　アルゴリズム ReapportionTime
Fig. 9　Algorithm ReapportionTime.

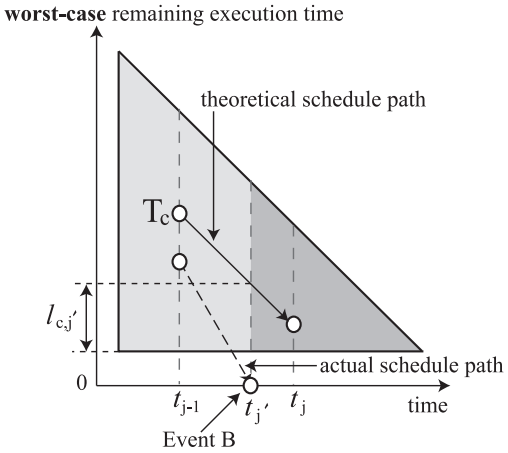

図 10　実際のスケジュールと理論上のスケジュール
Fig. 10　Actual schedule and theoretical schedule.

証明. T-N Plane を更新する時刻 $t_{j'}$ より前はノード内でスケジュール可能である．そのため，時刻 $t_{j'}$ までは Critical Moment が発生しないことから，定理 2 より $S_{j'} \leq M$ である．ここで，$t_{j'}$ を t_0 と読みかえると，図 10 の時刻 $t_{j'}$ から t_f の濃い影の三角形を時刻 t_0 に移動することに等しい．ReapportionTime は，ApportionTime と同様の規則に従って時間分配を行うため，全タスクはノード内でスケジュール可能であり，かつ，仕事量を保存する．□

5. 評　　価

拡張 TNPA の有効性をタスクのプリエンプション数から評価する．タスクのプリエンプションには，コンテキストの保存と復元やデータの操作が必要になることから，レイテンシの大きい多くのメモリアクセス命令が必要となる．比較対象は，拡張 TNPA の元である TNPA，TNPA よりオーバヘッドの面で優れている最適なアルゴリズムである EKG，非最適なアルゴリズムとして最も効率的であると考えられている EDZL[3] である．拡張 TNPA の時間分配は，実行可能なタスクを可能な限り早く減らすため，残り実行時間の少ないタスクに優先的に時間を与える．EDZL が実時間性を保証可能なシステム使用率について正確な上限は示されていないが，$U_s = 0.5$ 以下では実時間性を保証可能であり[12]，$U_s = (1-1/e) \simeq 0.6321$ を超えるとスケジュール不可能なタスクセットが存在する[13]．EDZL の評価では，EDZL がスケジュール可能であったタスクセットの結果のみを考慮する．

5.1 評 価 方 法

評価は，16 個のプロセッサを有するシステムにおいて，システム使用率が $[0.25, 1.0]$ の範囲で 0.025 ごとにそれぞれ 100 のタスクセット生成して行った．それぞれのタスクセットについて時刻 0 から時刻 $\min\{\mathrm{lcm}\{p_i | T_i \in T\}, 2^{32}\}$ までスケジューリングを行いタスクのプリエンプション数を算出して全タスクセットの平均値を求める．周期タスクモデルでは，周期の最小公倍数 $\mathrm{lcm}\{p_i | T_i \in T\}$ ごとに同じスケジュールが繰り返されるため，周期の最小公倍数までスケジュールを行い評価を行うことが望ましい．しかしながら，周期を一様乱数で生成しているため，評価時間が非常に長くなってしまう可能性がある．そのため，十分に長い時間として 2^{32} を評価時間の最大値とした．

それぞれのタスクセットの生成方法を述べる．タスクセットに含まれるタスクは，プロセッサ使用率 u_i が $[0.1, 1.0]$ の範囲で一様分布となるように生成した．タスクの周期 p_i は $[100, 3000]$ の整数値から一様乱数により生成する．最悪実行時間 c_i は $u_i p_i$ の整数部分とする．初めにタスクセットを空集合としてタスクを追加していき，タスクセットのプロセッサ使用率が目標値を超えたら最後に生成したタスクを破棄する．最後に，タスクセットのプロセッサ使用率が目標値となるようなタスクを生成して，これをタスクセットとする．

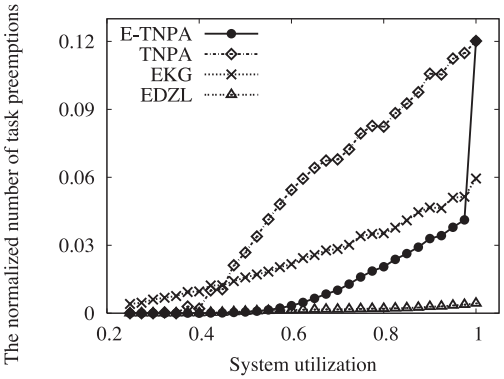

図 11　タスクのプリエンプション数（100%）
Fig. 11　Number of task preemptions (100%).

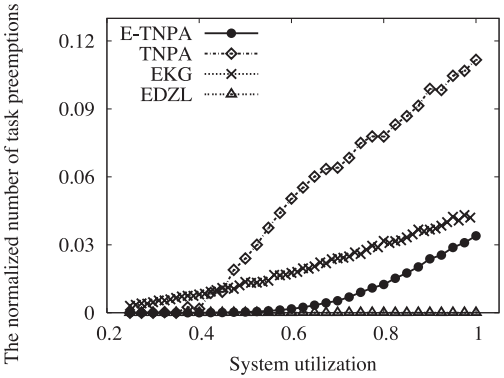

図 12　タスクのプリエンプション数（75%）
Fig. 12　Number of task preemptions (75%).

5.2 評 価 結 果

実際の実行時間 a_i と最悪実行時間 c_i について，つねに $a_i = c_i$ のときの結果を**図 11**，a_i が $[0.75c_i, c_i]$ の範囲で周期ごとに一様に変動するときの結果を**図 12**，a_i が $[0.5c_i, c_i]$ の範

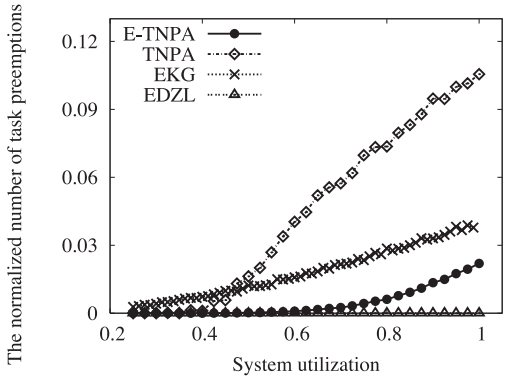

図 13　タスクのプリエンプション数（50%）
Fig. 13　Number of task preemptions (50%).

囲で周期ごとに一様に変動するときの結果を**図 13** に示す．図の横軸はタスクセットのシステム使用率，縦軸は評価時間とプロセッサ数によって正規化されたタスクのプリエンプション数である．すなわち，図の縦軸は，単位時間と単位プロセッサあたりのタスクのプリエンプション数を示している．

全体的な傾向として，システム使用率の高いところでは，EDZL，拡張 TNPA，EKG，TNPA の順に効率的なアルゴリズムである．拡張 TNPA と TNPA の差異は，仕事量を保存しているかどうかのみであるため，拡張 TNPA は仕事量を保存することによりタスクのプリエンプション数を大幅に削減したと判断可能である．EDZL は最も優れた結果を残しているが，システム使用率の高いところでは，実時間性を保証することができない．システム使用率の低い範囲で EKG によるタスクのプリエンプション数が多いのは，負荷を特定のプロセッサに偏らせるためである[6]．

これらの結果の中で最も重要なのは，実時間システムでは最悪時のオーバヘッドを考慮しなければならないため，最悪実行時間と実際の実行時間がつねに等しい図 11 である．EDZL が実時間性を保証可能であることを証明されている $U_s = 0.5$ 以下では，拡張 TNPA は EDZL と同程度に効率的である．拡張 TNPA と EKG を比較してみると，システム使用率が最大の $U_s = 1$ を除いて拡張 TNPA がつねに優れている．拡張 TNPA が最後に大きく変化しているのは，⟨A⟩ $U_s = 1$ では，拡張 TNPA によって分配可能な時間が存在しないために TNPA と同じスケジュールを行い，⟨B⟩ $U_s = 0.975$ すなわち $U = 15.6$ では，$M - U = 0.4$ すなわち 1 プロセッサの 40%の時間を分配可能であるためである．

実際の実行時間が変動したときの影響を述べる．実際の実行時間と最悪実行時間がつねに等しい図 11 では，拡張 TNPA はシステム使用率が最も高いところでタスクのプリエンプション数が大幅に増加しており，$U_s = 1$ では EKG の方が優れていた．逆に実行時間が変動してプロセッサのアイドル時間が発生する図 12 と図 13 では，$U_s = 1$ であっても拡張 TNPA が EKG の結果よりも優れている．これは，拡張 TNPA が仕事量を保存するために，余った時間を効率的にタスクに再分配可能であるためである．実際にシステムを構築する際には最悪時である図 11 の結果を考慮しなければならないが，そのような状況は非常に稀であるため，拡張 TNPA のオーバヘッドは実行時間が変動してオーバヘッドが小さい図 12 や図 13 の結果を期待できる．実時間システムには，本論文で扱った実時間タスクのみではなく，バックグランドで処理する非実時間タスクも存在するため，オーバヘッドを削減した分だけ非実時間タスクを効率的に実行することが可能である．

評価を通して，本論文で提案した拡張 TNPA は，EDZL が保証可能なシステム使用率において同程度までプリエンプション数を削減可能であることを示した．理論的には拡張 TNPA，実装容易性では EDZL が優れているが，EDZL は信頼性がない[3] のに対して，拡張 TNPA は信頼性があるという点から優れている．

6. 結　　論

提案した拡張 TNPA は，時間分配と仮想 T-N Plane という概念を利用して，仕事量を保存する最適な実時間スケジューリングを実現する．従来手法の TNPA と異なる点は，タスクの到着と終了時に時間分配アルゴリズムを行わなければならないことのみである．時間分配アルゴリズムは，任意のタスクに優先的に時間を与えることが可能である．このわずかな操作により，タスクのプリエンプションに必要なレイテンシの大きい多くのメモリアクセスを避けることが可能となる．

拡張 TNPA のもう 1 つの貢献は，最適なアルゴリズムの現在の限界を突破できる可能性があることを示したことである．Pfair は，厳しい制約をもうけることにより，タスクの実行がほぼつねに fluid スケジュールパスを追跡する．TNPA では，Pfair スケジューリングの制約を緩和して，全タスクのデッドラインでのみ fluid スケジュールパスを追跡している．本論文で提案した拡張 TNPA と時間分配アルゴリズム ApportionTime の組合せでは，全タスクのデッドラインでのみ fluid スケジュールパスより下にタスクが存在すればよいという条件に緩和した．拡張 TNPA の抽象モデルは，時間分配手法を優れたものに置き換える

ことにより，各タスクが自身のデッドラインでのみ fluid スケジュールパスを追跡することが可能となる．これは，システムモデルで求められていたデッドラインまでに実行を終えるということにほかならない．これを実現する優れた時間分配手法を将来の課題とする．

謝辞　本研究は，科学技術振興機構 CREST の支援による．

参 考 文 献

1) Liu, C.L. and Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *J. ACM*, pp.46–61 (1973).
2) Lopez, J.M., Garcia, M., Diaz, J.L. and Garcia, D.F.: Worst-Case Utilization Bound for EDF Scheduling on Real-Time Multiprocessor Systems, *Proc. 12th Euromicro Conference on Real-Time Systems*, pp.25–33 (2000).
3) Lee, S.K.: On-line Multiprocessor Scheduling Algorithms for Real-Time Tasks, *Proc. IEEE Region 10's 9th Annual International Conference*, pp.607–611 (1994).
4) Andersson, B., Baruah, S. and Jonsson, J.: Static-Priority Scheduling on Multiprocessors, *Proc. 22nd IEEE Real-Time Systems Symposium*, pp.193–202 (2001).
5) Anderson, J.H. and Srinivasan, A.: Early-Release Fair Scheduling, *Proc. 12th Euromicro Conference on Real-Time Systems*, pp.35–43 (2000).
6) 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).
7) Cho, H., Ravindran, B. and Jensen, E.D.: An Optimal Real-Time Scheduling Algorithm for Multiprocessors, *Proc. 27th IEEE Real-Time Systems Symposium*, pp.101–110 (2006).
8) Cho, H., Ravindran, B. and Jensen, E.D.: Synchronization for an Optimal Real-Time Scheduling Algorithm on Multiprocessors, *Proc. 2nd IEEE International Symposium on Industrial Embedded Systems*, pp.9–16 (2007).
9) Baruah, S.K., Cohen, N.K., Plaxton, C.G. and Varvel, D.A.: Proportionate Progress: A Notion of Fairess in Resource Allocation, *Algorithmica*, Vol.15, No.6, pp.600–625 (1996).
10) Aydin, H. and Yang, Q.: Energy-Aware Partitioning for Multiprocessor Real-Time Systems, *Proc. 17th IEEE International Parallel and Distributed Processing Symposium*, pp.22–26 (2003).
11) Holman, P. and Anderson, J.H.: Adapting Pfair Scheduling for Symmetric Multiprocessors, *Journal of Embedded Computing*, Vol.1, No.4, pp.543–564 (2005).
12) Piao, X., Han, S., Kim, H., Park, M. and Cho, Y.: Predictability of Earliest Deadline Zero Laxity Algorithm for Multiprocessor Real-Time Systems, *Proc. 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, pp.24–26 (2006).
13) Wei, H.-W., Chao, Y.-H., Lin, S.-S., Lin, K.-J. and Shih, W.-K.: Current Results on EDZL Scheduling for Multiprocessor Real-Time Systems, *Proc. 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.21–24 (2007).

(平成 20 年 5 月 7 日受付)
(平成 20 年 9 月 12 日採録)

船岡　健司（正会員）

1982 年生．2006 年慶應義塾大学理工学部情報工学科卒業．2008 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，株式会社東芝勤務の傍ら，同大学院博士課程に在籍．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．

加藤　真平（正会員）

1982 年生．2004 年慶應義塾大学理工学部情報工学科卒業．2008 年同大学大学院理工学研究科開放環境科学専攻博士課程修了．博士（工学）．現在，同大学訪問研究員．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．工学博士．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．