修 士 論 文

スラックを利用した実行権移譲スケジューリングアルゴリズム

北陸先端科学技術大学院大学
情報科学研究科

鈴木 隆元

2016 年 9 月

修 士 論 文

スラックを利用した実行権移譲スケジューリングアルゴリズム

1310753　鈴木 隆元

主指導教員　田中 清史
審査委員主査　田中 清史
審査委員　金子 峰雄
審査委員　井口 寧

北陸先端科学技術大学院大学
情報科学研究科

平成 28 年 8 月

概要

組み込みアプリケーションは複数のタスクから構成され, それらはそれぞれ異なる重要度を持つ. 特に, ハード・デッドラインを持つタスク (ハードタスク) は意味的に重要であり, ソフト・デッドラインを持つタスクより優先されるべきである. ハードタスクのグループにおいても, 特に重要なタスクには高いリアルタイム性が求められる. リアルタイム性の確保, すなわち, タスク応答時間とジッタの短縮はリアルタイムシステムにおける重要なテーマの一つである. リアルタイムシステムの開発において一般的となっている静的優先度リアルタイムスケジューリングでは, 周期が短いタスクほど優先度が高い. 一方で, 長い周期のタスクの優先度は下がるため, タスク応答時間は長くなりジッタも増える. 本研究の目的は, 周期は長いが意味的に重要なタスクの応答時間の短縮である. タスクセット全体のスケジューラビリティを保ったまま, 高優先度サーバを利用することで特権的に特定タスクを動作可能とする Execution Right Delegation (**ERD**) 法と, タスク実行の早期終了により発生する時間的スラックを, 特定タスクのために利用する Slack Collection (**SC**) 法を提案・評価する.

目 次

1 はじめに

1.1 背景

1.1.1 周期タスクのスケジューリングアルゴリズム

ハードリアルタイム・オペレーティングシステムにおけるリアルタイムスケジューリングの主な目的は, タスクの周期やタスクの実行時間, あるいは締切時刻 (デッドライン) をもとに最適なスケジューリングを行うことにある. 各タスクのデッドラインを守ることに加え, 重要タスクの応答時間やジッタの短縮もスケジューリングの目的に含まれる.

決められた周期毎に実行要求を出すタスクを周期タスクという. 周期タスクに対するリアルタイムスケジューリングは, タスクの優先度が静的に決まる Rate Monotonic (**RM**) 法 [1] と, タスクの優先度が動的に決まる Earliest Deadline First (**EDF**) 法に大別できる. **RM** 法は **EDF** 法に比べると許容される CPU 使用率に劣るが, 実行時オーバヘッドが少なく動作の予測がしやすい. また, タスクの周期を固定優先度に反映させてアプリケーションを実装すれば, ITRON[2] に代表される固定優先度ベースのリアルタイムオペレーティングシステムで実現が可能であるなど, **EDF** 法より一般的なアルゴリズムと言える. しかし, **RM** 法ではタスク優先度はタスクの周期によって決まるため, 周期が短いタスクほど応答時間が短くなる一方で, 周期の長いタスクは優先度が低く, 応答時間が長くなりジッタは増える傾向がある. よって周期の長いタスクの応答時間短縮という要求には **RM** 法では応えることができない.

1.1.2 スラックの利用

リアルタイムスケジューリング理論で扱われてきたタスクの実行時間とは, タスクの最悪実行時間を意味することがほとんどであった. 実際の稼働システム上では, 多くの場合タスクは 最悪実行時間より短い時間で処理を終了し, CPU 時間に余りが生じる. この余りをスラックと呼ぶ. 本研究では, 特定の重要タスクの応答時間を短くするためにスラックに着目する.

タスクの優先度が動的に変わる **EDF** ベースのスケジューリングを対象とした, スラックを利用することで非周期タスクの応答時間を短縮することを目的とする研究が従来より行われてきた [3]. タスクの優先度が静的に決まる **RM** ベースのスケジューリングで提案されている手法 [4] [5] では, 計算オーバヘッドやスケジューラビリティの点で劣る等, スラックの利用に関する十分な研究がされているとは言い難い.

1.2 本研究の目的

リアルタイムスケジューリングの研究は, 周期タスクのスケジューリングに始まり, 非周期タスクのスケジューリングアルゴリズムの研究を経て, 近年ではクリティカリティ混在システム, マルチコアスケジューリングとその幅を広げてきた [6]. 本研究はもはや古典となっている周期タスクのスケジューリングアルゴリズムを対象とするが, 周期タスクのスケジューリングアルゴリズムの研究分野において応答時間やジッタ短縮に関する研究は十分にされているとは言い難い. 本研究では, 周期タスクのスケジューリングアルゴリ

ズムの後に研究された非周期タスクのスケジューリングアルゴリズムの考えを取り入れている. 本研究は, Liu, Layland によるモデル [1], すなわちタスクは周期と実行時間しか持たないモデルを採用しているが, 本研究より複雑なモデルである Deadline Monotonic (**DM**) 法 [9] に比べ, 提案手法は応答時間に関して有利であることを示す.

1.3 論文の構成

第 2 章で本研究と関連する研究について述べる. 第 3 章では本研究で提案するスケジューリングアルゴリズムについて述べる. 第 4 章で提案手法の評価結果を, 第 5 章で考察を述べる. 最後に第 6 章で本論文をまとめる.

2　関連研究

2.1　スケジューリングアルゴリズム

周期タスクを静的優先度でスケジューリングするアルゴリズムが 70 年代より研究されてきた. **RM** 法は静的優先度リアルタイムスケジューリングの代表的なアルゴリズムである.

スケジューリングアルゴリズムではコンピュータにさせる仕事をタスク (プロセス, スレッドに置き換えても良い) という単位に分けて扱う. **RM** 法で扱うモデルでは, タスク τ_i は無限の Job を生成 (リリース) する. Job はリリースされると規定の時間 C_i だけ実行する. また, Job は周期 T_i 毎にリリースされる. Job が終了したらタスクは次のリリースまで何もしない. タスクは $\tau_i = (C_i, T_i)$ で表記される. 以下は $\tau_1 = (1, 5)$ の図の例である.

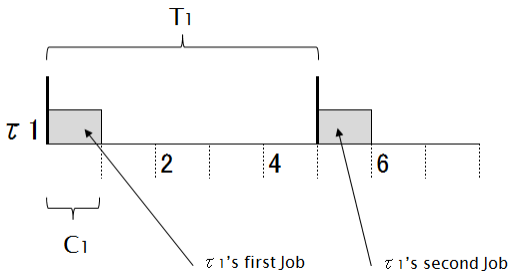

図 2-1. タスクと Job

一つの CPU で複数のタスクを実行可能である. 複数のタスクがある場合, タスクは $\tau_1, \tau_2, ..., \tau_n$ のように表わす. 数字の小さいほうが優先度が高くなり, 優先度は周期が短いほうが高い (すなわち $T_1 \leq T_2 \leq ...T_n$ である). タスク τ_i の Job が終了すると, τ_{i+1} の Job に CPU の実行権は割り当てられる. タスクセット Γ には 1 つ以上のタスクが含まれる. タスクセットを $\Gamma = \{\tau_1, \tau_2, ..., \tau_n\}$ または $\Gamma = \{(C_1, T_1), (C_2, T_2), ..., (C_n, T_n)\}$ のように表記する.

以下は $\Gamma = \{(2, 5), (2, 8), (2, 10)\}$ を **RM** 法でスケジューリングした結果である.

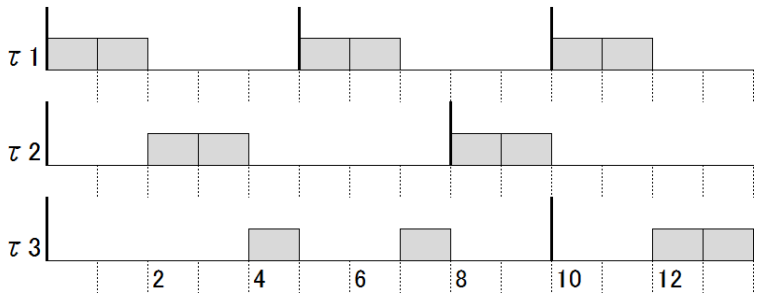

図 2-2. **RM** 法によるスケジューリング例

リリースされた Job は次のリリース (すなわち, 次のタスク周期) までに仕事を終えなければならない. 他のタスクに実行権を取られる等して仕事を終えられない場合, デッドラインミスとなり, 特にハードリアルタイムシステムにおいては致命的な結果を引き起こす. よってスケジューリングアルゴリズムはデッドラインミスが起きないようスケジュールしなくてはならない.

RM 法と別のモデルを扱うスケジューリングアルゴリズムに Deadline Monotonic (**DM**) 法がある. **DM** 法ではタスクは周期 T_i と実行時間 C_i に加え相対デッドライン D_i を持つ. **RM** 法とは異なり, **DM** 法におけるタスク優先度はデッドラインが短いものほど高くなる. **DM** 法はタスク周期とは独立したデッドラインにより優先度を決定可能なため, たとえ周期が長いタスクであってもデッドラインを短くすることで応答時間とジッタを短縮可能である.

RM 法, **DM** 法は周期タスクを対象とするアルゴリズムであるが, 非周期タスクを対象としたスケジューリングアルゴリズムも従来より活発に研究されてきた. 非周期タスクとは, 周期タスクのように周期 T_i を持たず, 実行時間 C_i のみを持つ. いつタスクのリクエストが発生するかは定義されない†. 多くの非周期タスクスケジューリングアルゴリズムでは, 非周期タスクを実行するための専用タスク (サーバ) を用いる. サーバは短い周期を持ち, 高優先度で動作する. サーバに実行権が渡ったさい, 非周期タスクのリクエストがある場合はサーバの実行時間 (キャパシティ) で実行してよい.

非周期タスクスケジューリングアルゴリズムの一つに Priority Exchange (**PE**) 法 [8] がある. **PE** 法では, サーバに実行権があり, かつ非周期リクエストが存在しない場合, サーバの優先度と (サーバの次に優先度の高い) 周期タスクの優先度が入れ替わり, 周期タスクが実行可能状態となる. もし周期タスクが実行している際に非周期タスクのリクエストが発生した場合, 再びサーバと周期タスクの優先度の交換が行われ, サーバのもとのキャパシティの範囲で即非周期タスクが実行可能である. 優先度を交換することでサーバの実行権あるなしによらず, 非周期タスクの応答時間を縮めることを狙いとしている.

以下は サーバ $S = (1, 5), \tau_1 = (3, 10), \tau_2 = (9, 20)$ を **PE** 法でスケジューリングした結果である.

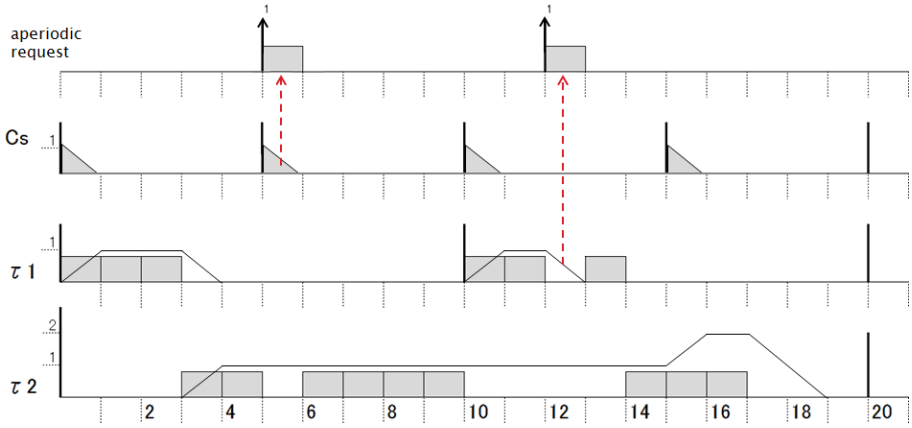

図 2-3. **PE** 法によるスケジューリング例

時刻 0 において, 高優先度サーバに実行権が渡るが非周期リクエストが無いため τ_1 とサーバの優先度が入れ替わり τ_1 の Job が実行する. このさい, サーバの実行時間 1 が τ_1 の優先度で蓄積される (台形は蓄積されたキャパシティを示す). 時刻 3 において τ_1 の最初の Job の実行終了と同時に, サーバの優先度はより優先度の低い τ_2 と交換される. 時刻 5 に発生した非周期リクエストは, サーバの 2 回目の実行と同じタイミングであるため, サーバのキャパシティを用いて実行される. 時刻 10 に再び優先度の交換が行われ, τ_1 の優先度でサーバのキャパシティが蓄積される. 時刻 12 に再び非周期リクエストが発生

†モデルによっては, 最短リクエスト周期が定義されるものもある.

する. この非周期リクエストは τ_1 の優先度で蓄積されたサーバのキャパシティを使い即実行される (蓄積されたキャパシティは消滅する). 時刻 15 にサーバに実行権が渡るが, 既に τ_1 の Job は実行を終えているため, サーバの優先度は τ_2 と入れ替わる. サーバのキャパシティは, それまでに蓄積されていたキャパシティに上乗せする形で蓄積される. 時刻 17 において τ_2 の Job が終了してシステムがアイドル状態になると, τ_2 に蓄積されていたキャパシティはその量と同じ時間をかけて消費され, 消滅する.

Extended Priority Exchange (**EPE** 法)[5] は, **PE** 法を拡張したアルゴリズムである. **EPE** 法ではサーバの代わりにあらかじめ各タスクの最悪実行時間 C_i を水増しし, これを非周期タスクのキャパシティとして使う. **EPE** 法では, あらかじめ非周期タスクのための帯域を水増しにより予約していると考えられる.

PE 法, **EPE** 法等とは別のアプローチによる, CPU 使用率向上と非周期タスクの反応速度向上を目的とした非周期タスクのスケジューリングアルゴリズムが存在する. Dual Priority (**DP**)[10] 法は, **RM** 法に代表されるモデルとは異なり一つのタスクに二つの優先度を持たせるアルゴリズムである. 周期タスクは低優先度と高優先度, 二つの優先度を持ち, 非周期タスクは中優先度で動作する. 周期タスクはリリース直後は低優先度で動作するが, デッドラインが近くなると高優先度に昇格する. この間, 非周期タスクのリクエストがあれば中優先度で動作する. **DP** 法の狙いは, 周期タスクをデッドラインミスしない限界まで遅らせることで, 優先的に非周期タスクを実行可能とする点にある.

DP 法と同様の考え方を持つ手法に Zero Slack Scheduling[11] 法がある. このスケジューリングアルゴリズムはクリティカリティ混在システムをモデルとしている. このモデルでは一つのシステム上に異なる重要度のタスクが混在 (e.g. ブレーキシステムとラジオ) し, 従来の優先度に加えて, タスクは Criticality(重要度) を持つ. タスクは Normal/Critical の二つのモードで動作するが, スケジューラビリティを確保するために, タスクはある時刻で Critical モードに昇格し優先して実行される.

2.2 スケジューラビリティの判定

与えられたタスクセットがスケジュール可能か, すなわち全てのタスクがデッドラインミスを起こさずに仕事を終えることが可能かを判定する手法が従来より研究されてきた. Liu, Layland の研究 [1] において, 全タスクの CPU 占有率の合計値をもとにスケジュール可能か判断する手法が提案された. この手法は多項式時間で計算可能であるため, タスクが動的に追加されるシステムにおいて有効である [‡]. しかしこの手法で得られる結果は十分条件であり, 見積もりも悲観的である. その後, Bini, Buttazo らによって Liu, Layland らの手法より悲観度の低い, 同じく多項式時間で計算可能な手法が提案された [7].

両手法は CPU 占有率を元に判定する手法であったが, タスク応答時間を求めることによる判定手法 Responce Time Analysis (RTA) [12] が Joseph, Pandya, Audsley らによって提案された. この手法では, タスク応答時間は疑似多項式時間で決定可能であり, NP 困難のクラスに属することが近年証明されたが [13], 静的優先度スケジューリングにおけるスケジューラビリティの必要十分条件を示すことが可能である. RTA については次章で詳細に扱う.

[‡]タスクの追加要求があったとき, そのタスクを追加してもタスクセット全体のスケジューラビリティが損なわれないかを判定するシステムが存在する.

本提案手法は RTA を使うため, オンタイムスケジューリングには向かない. あらかじめタスクセットが固定されている (タスクが動的に追加されない) 前提で適用する手法である.

3　提案スケジューリング手法

本提案で扱うタスクモデルは, 実行時間, 周期のみを持ち, 相対デッドラインは周期と等しい. また位相は持たない (時刻 0 で最初のリリースが行われる). スケジューリング方式は **RM** 法に代表される固定優先度スケジューリング方式を基本とする. 本節ではスケジューラビリティを示すための定理について述べ, 続いてそれらを用いて Execution Right Delegation (**ERD**) 法と Slack Collection (**SC**) 法を提案する.

3.1　スケジューラビリティを示すための定理

定義 1 (Responce Time Analysis(RTA)[12])

静的優先度スケジューリングにおけるタスク τ_i の最長応答時間 [a] R_i は以下で与えられる.

$$R_i \quad = \quad C_i + \sum_{j=1}^{i-1} \lceil \frac{R_i}{T_j} \rceil Cj \tag{1}$$

[a]時刻 0 で全タスクの Job が同時にリリースされた場合 (Critical Instant [1]), 各タスクの応答時間は最長となる [12].

R_i は左辺と右辺両方に含まれる. よって R_i の計算は, 初期値に適当な値 (例えば C_i とする) を設定し, 右辺と左辺が一致するまで R_i の値を増やしていくことで計算を行う.

例 2 (タスク応答時間)

$\Gamma = \{(1,5),(1,6),(2,8),(4,14)\}$ を考える. ***RM*** 法でスケジュールした図を以下に示す.

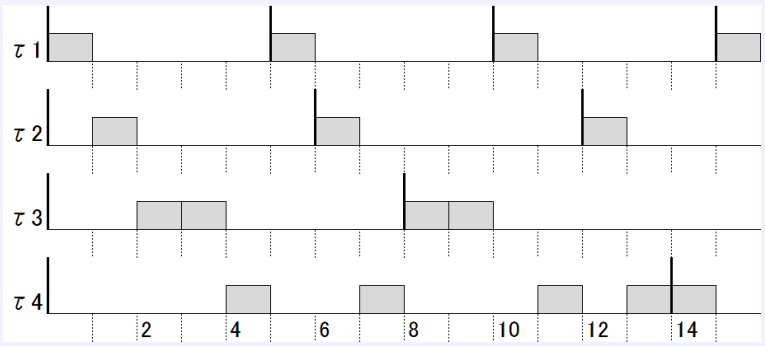

図 *3-1.* ***RM*** 法によるスケジューリング例

定義 *1* による各タスク応答時間を計算する. R_1 は最高優先度のタスクであるため, 値は即求めることができる.

$$R_1 = C_1 = 1$$

R_2 は以下の式で求められる.

$$R_2 = C_2 + \lceil \frac{R_2}{T_1} \rceil \times C_1$$

ここで, 初期値として $R_2 = 1$ とおくと,

$$\begin{aligned} R_2 &= 1 + \lceil \frac{1}{5} \rceil \times 1 \\ &= 1 + 1 \\ &= 2 (! = 1) \end{aligned}$$

よって R_2 は *1* ではない. 次に $R_2 = 2$ とおくと,

$$\begin{aligned} R_2 &= 1 + \lceil \frac{2}{5} \rceil \times 1 \\ &= 1 + 1 = \\ &= 2 \end{aligned}$$

よって $R_2 = 2$ である. 同様に R_3 を計算する.

$$R_3 = C_3 + \lceil \frac{R_3}{T_1} \rceil \times C_1 + \lceil \frac{R_3}{T_2} \rceil \times C_2$$

$R_3 = 4$ とすると

$$\begin{aligned} 4 &= 2 + \lceil \frac{4}{5} \rceil \times 1 + \lceil \frac{4}{6} \rceil \times 1 \\ &= 2 + 1 + 1 \\ &= 4 \end{aligned}$$

R_4 も同様である.

$$R_4 = C_3 + \lceil \frac{R_4}{T_1} \rceil \times C_1 + \lceil \frac{R_4}{T_2} \rceil \times C_2 + \lceil \frac{R_4}{T_3} \rceil \times C_3$$

$R_4 = 14$ とすると

$$\begin{aligned} 14 &= 4 + \lceil \frac{14}{5} \rceil \times 1 + \lceil \frac{14}{6} \rceil \times 1 + \lceil \frac{14}{8} \rceil \times 2 \\ &= 4 + 3 + 3 + 4 \\ &= 14 \end{aligned}$$

これらのタスクの応答時間は, 図で示した応答時間と一致する.

定理 3 (スケジュール可能性の判定 [12])

タスクセット Γ に含まれる全てのタスク τ_i について, 各タスクの最長応答時間 R_i がタスク周期 T_i 以下であれば, Γ はスケジュール可能である.

定義 4 (タスク優先度の設定)

RM 法に代表されるタスク周期を元にしたタスク優先度に加え, 提案手法では特定のタスクの優先度を周期によらず設定可能とする.

補題 5 (タスク優先度の入れ替え)

あるタスク τ_p の最長応答時間 R_p が, τ_p よりひとつ高優先度のタスク τ_h の周期 T_h 以下である場合, τ_p の優先度と τ_h の優先度を入れ替えてもスケジュール可能である.

証明

タスクセット Γ に含まれる τ_p と τ_h の優先度を入れ替えたとしてもスケジューラビリティが保たれること, すなわち定理 3 より, 優先度を入れ替えた後のタスクセットに含まれるすべてのタスクの最長応答時間 R'_i が周期 T_i 以下であることを示せば良い. 以後, 優先度を入れ替えた後の時刻には, R'_i 等, 全てカンマをつける.

タスクセット Γ を, τ_h より高優先度のタスクのグループ Γ_{high}, τ_p より低優先度のタスクのグループ Γ_{low}, そして τ_h と τ_p からなるタスクのグループに分ける. 以下, 優先度を入れ替えた後の全てのタスクについてデッドライン制約が満たされることを示す.

Γ_{high} に含まれるタスク τ_{high} は応答時間に変化は無い. なぜなら, τ_{high} より低優先度のタスク間での優先度入れ替えに起因する変化は τ_{high} の最初の Job の実行終了後の事象であるためである. 次に, Γ_{low} に含まれるタスク τ_{low} についても応答時間に変化は無い. τ_{low} より高優先度のタスクの実行順が入れ替わろうと, τ_{low} のタスクに実行権が渡る時刻は変わらない. よって, 証明は τ_h および τ_p についてデッドライン制約が満たされることを示せば十分である.

τ_p に関しては元の優先度より高くなるため, 応答時間が増大するということは無い. 優先度の関係より $T_h \leq T_p$, 前提より $R_p \leq T_h$, 応答時間に関し $R'_p < R_p$ であるため $R'_p < T_h \leq T_p$ となりデッドライン制約を満たす.

次に τ_h について考える. Γ_{high} に含まれるタスクが T_h までに 1 つ以上の Job をリリースして τ_p, τ_h をブロックするか否か, 二つのケースを考える. まずは Γ_{high} に含まれるタスクが τ_p, τ_h をブロックしないケースについて考える. 優先度を入れ替える前の τ_h の Job の開始時刻を s_h, τ_p の Job の開始時刻を s_p とすると, それぞれのタスクの応答時間は以下となる.

$$R_h = s_h + C_h$$
$$R_p = s_p + C_p$$

一方で, 優先度を入れ替えると,R'_p, R'_h は以下となる.

$$\begin{aligned} R'_p &= s'_p + C_p \\ R'_h &= s'_h + C_h \end{aligned}$$

ここで, τ_p の Job の開始時刻 s_p は (Γ_{high} にブロックされないため) τ_h の Job の終了時刻 R_h と等しい. また, 優先度を入れ替えた後の s'_h も同様である.

$$s_p = R_h$$
$$s'_h = R'_p$$

ここで, $s'_p = s_h$ であるので

$$\begin{aligned} R'_h &= s'_h + C_h \\ &= R'_p + C_h \\ &= s'_p + C_p + C_h \\ &= s_h + C_p + C_h \\ &= R_h + C_p \\ &= s_p + C_p \\ &= R_p \end{aligned}$$

前提より $R_p \leq T_h$ であるので, 優先度を入れ替えたとしても $R'_h \leq T_h$ でありデッドライン制約を満たす.

以下に Γ_{high} にブロックされない場合の例を図で示す.

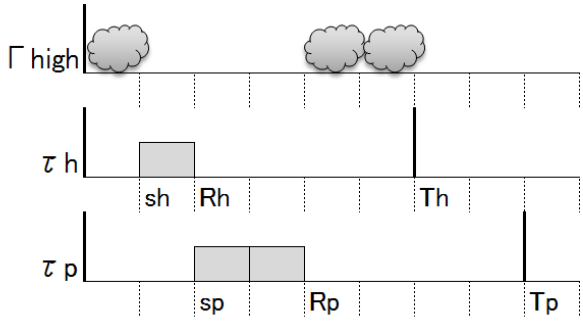

図 3-2. Γ_{high} にブロックされないケース

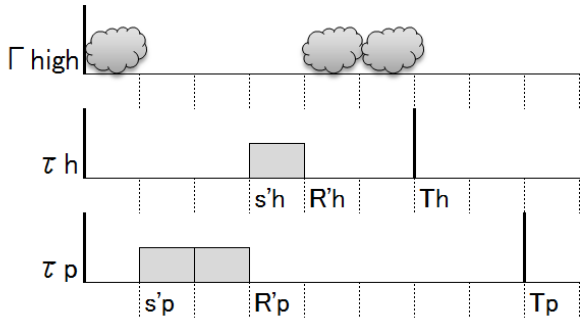

図 3-3. Γ_{high} にブロックされないケース (優先度入れ替え後)

次に, Γ_{high} に含まれるタスクが T_h までに新たな Job をリリースし, τ_p, τ_h をブロックするケースについて考える. 前提より T_h の時点で τ_h と τ_p はその Job を終了している. よって τ_h の Job の開始時刻 s_h から数えて, ブロックされる時間 BT は以下で与えられる.

$$BT \leq T_h - s_h - (C_h + C_p) \tag{2}$$

R_p は以下である.

$$R_p = s_h + C_h + C_p + BT$$

優先度を入れ替えた後の R'_h は以下である.

$$R'_h = s'_p + C_p + C_h + BT$$

ここで, $s'_p = s_h$ であるので,

$$R'_h \quad = \quad s_h + C_p + C_h + BT$$

BT を左辺へ移動すると,

$$BT \quad = \quad R'_h - s_h - C_p - C_h$$

式 (2) に対し, BT を展開すると,

$$\begin{aligned} R'_h - s_h - C_p - C_h &\leq T_h - s_h - (C_h + C_p) \\ R'_h &\leq T_h - s_h - (C_h + C_p) + s_h + C_p + C_h \\ R'_h &\leq T_h \end{aligned}$$

よって, ブロックされる最大時間を考慮しても $R'_h \leq T_h$ であるので, デッドライン制約を満たす.

以下に Γ_{high} にブロックされる場合の例を図で示す.

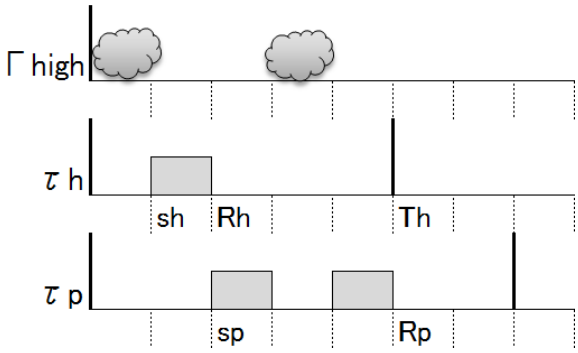

図 3-4. Γ_{high} にブロックされるケース

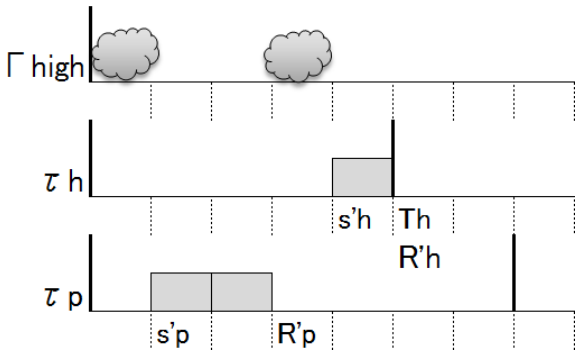

図 3-5. Γ_{high} にブロックされるケース (優先度入れ替え後)

(証明終わり)

定理 6 (タスク優先度の変更)

あるタスク τ_p の最長応答時間 R_p が, τ_p より高優先度のタスク τ_h の周期 T_h 以下である場合, τ_p の優先度を τ_h より一つ上としてもスケジュール可能である [a].

[a] 補題 5 では τ_p と τ_h の間にタスクは存在しなかったが, この定理では τ_p と τ_h の間にタスクが存在する.

証明

τ_h と τ_p の間に存在するタスク $\tau_{h+1}, \tau_{h+2}, ..., \tau_{p-1}$ の周期について, 以下が成り立つ.

$$T_h \leq T_{h+1} \leq T_{h+2} \leq ... \leq T_{p-1}$$

前提より $R_p \leq T_h$ であるため, R_p と各タスク周期について以下の関係が成り立つ.

$$R_p \leq T_h \leq T_{h+1} \leq T_{h+2} \leq ... \leq T_{p-1}$$

ここで τ_p の一つ上の優先度のタスク τ_{p-1} に注目すると, 補題 5 より τ_p と τ_{p-1} の優先度は入れ替え可能である. 以降, 優先度を入れ替えた τ_p を $\tau_{p-2}, \tau_{p-3}, ..., \tau_{h+1}, \tau_h$ の順に, 補題 5 により優先度を入れ替えてやれば良い.

(証明終わり)

定義 7 (アイドル時間)

どのタスクの *Job* も実行されなかった時間をアイドル時間という. $idle(t)$ を, 時刻 t までに存在するアイドル時間の合計とする.

例 8 (アイドル時間)

以下に示す図において, $idle(4) = 1$, $idle(6) = 2$ である.

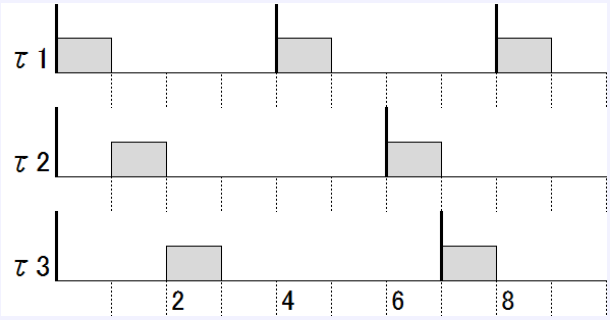

図 *3-6.* $idle(t)$

補題 9 (アイドル時間と Job の終了)

タスクセット Γ をスケジュールした結果において, $0 < idle(t)$ であれば, Γ に含まれる全てのタスクの最初にリリースされた Job は, 時刻 t において実行を終えている.

証明

固定優先度スケジューリング方式においては, リリース後の実行可能な Job が存在する限り, 常にその中の最高優先度の Job がスケジュールされる. したがって, アイドル時間が存在すればその時点で Job は存在しない. すなわち各タスクの最初の Job は実行を終えている.

(証明終わり)

補題 10 (アイドル時間へのタスクの追加)

タスクセット Γ をスケジュールした結果において, あるタスク τ_h の周期 T_h 内でアイドル時間が存在する場合, 実行時間を $idle(T_h)$, 周期を T_h とした新たなタスク τ_{idle} を Γ に追加してもスケジュール可能である.

証明

Γ に含まれるタスクについて, 周期が T_h 未満の高優先度タスクの集合を $\Gamma_{high} = \{\tau_1, \tau_2, ..., \tau_{h-1}\}$, 周期が T_h 以上である低優先度タスクの集合を $\Gamma_{low} = \{\tau_h, \tau_{h+1}, \tau_{h+2}, ..., \tau_n\}$ とする. Γ_{high} に含まれるタスクについては補題 5 の証明で述べたものと同じくスケジューラビリティに関して影響が無い. τ_{idle} も, Job の応答時間が T_h 以下となるのは明らかであり, デッドライン制約を満たす. よって証明は Γ_{low} に含まれるタスクについてスケジューラビリティを示せば十分である.

Γ_{low} の最低優先度のタスク τ_n に注目する. τ_n の最初の Job の応答時間 R_n と T_h について, 以下の式が成り立つ.

$$R_n + idle(T_h) + BT = T_h \tag{3}$$

ここで BT は, Γ_{high} のタスクの 2 回目以降の Job が, 時刻 R_n から T_h の間に実行される時間である.

$\tau_{idle} = (idle(T_h), T_h)$ を Γ に追加した後の, τ_n の応答時間を R'_n とする. 時刻 0 から T_h の間に τ_{idle} が Job をリリースするのは 1 度のみであり, τ_n に $idle(T_h)$ 時間の影響を及ぼす. τ_{idle} を追加する前の Γ について, 補題 9 より, 時刻 T_h より前に全てのタスクの最初の Job は実行を終了している. τ_{idle} の追加による Γ_{high} への影響は無いため, 時刻 T_h において Γ_{high} に含まれるタスクは最初の Job を終了している. すなわち τ_{idle} を追加した場合の τ_n への影響は, 最大で[§] Γ_{high} の各タスクの 2 回目以降の Job の実行時間の合計である. この値は BT である. 以上から, R'_n は以下の上限を持つ.

$$R'_n \quad \leq \quad R_n + idle(T_h) + BT \tag{4}$$

[§]最大と断った理由は, 図 3-6 で $T_h = T3(= 7), \tau_n = \tau_3$ とした場合, $BT = 2$ であるが, τ_{idle} の追加により τ_3 は τ_2 の 2 回目の Job の影響は受けず $R'_3 = 6(< 7)$ となる場合があるためである.

式 (3) と式 (4) より

$$\begin{aligned} R'_n &\leq R_n + idle(T_h) + BT \\ &\leq R_n + idle(T_h) + T_h - R_n - idle(T_h) \\ &\leq T_h \end{aligned}$$

$T_h \leq T_n$ であるため $R'_n \leq T_h \leq T_n$ となり, τ_n はデッドライン制約を満たす.

τ_{idle} 追加後の Γ_{low} に含まれるタスクの応答時間と周期について, 以下の関係が成り立つ.

$$R'_h \leq R'_{h+1} \leq ... \leq R'_n \leq T_h \leq T_{h+1}, \leq ... \leq T_n$$

よって Γ_{low} に含まれる全てのタスクは τ_{idle} の追加後も, 定理 3 によりデッドライン制約を満たす.

(証明終わり)

例 11 (τ_{idle} の追加)

以下に図 *3-6* で示したタスクセットに対し, $\tau_h = \tau_1$, $T_h = T_1 = 4$ とした場合と, $\tau_h = \tau_2, T_h = T_2 = 6$ とした場合の τ_{idle} を追加した図をそれぞれ示す.

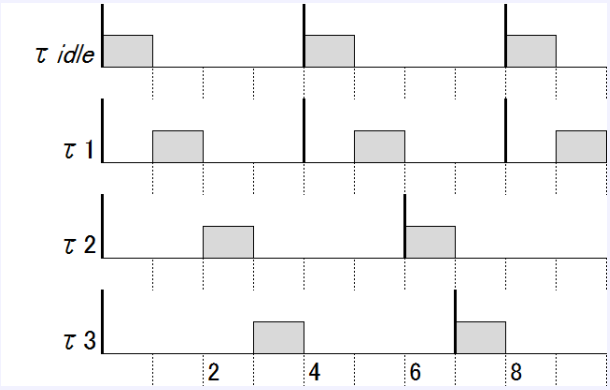

図 *3-7.* τ_{idle} の追加 *1*

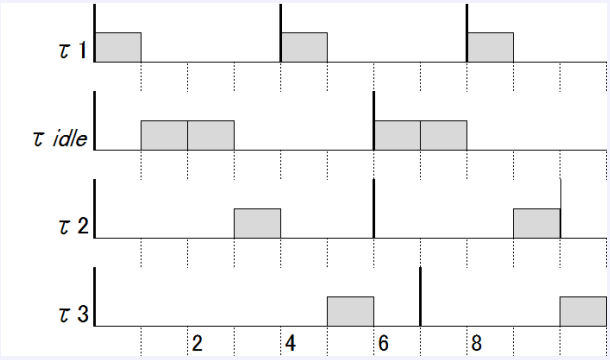

図 *3-8.* τ_{idle} の追加 *2*

3.2 Execution Right Delegation (ERD) 法

ERD 法は, 特定の周期タスク τ_p を特権的に動作させるために仮想的なサーバ VS を用いることで, τ_p の応答時間とジッタを短縮する手法である. VS はキャパシティ C_s と周期 T_s を持つが, サーバのキャパシティは別タスクに移譲するのみである.

定義 12 (実行権の移譲)

ERD 法においてサーバは他のタスクと同様にスケジュールされるが, サーバに実行権が渡った場合, サーバの代わりに特定の周期タスク τ_p が動作して良い. この振る舞いを実行権の移譲という. τ_p の *Job* の実行が既に終了している場合, サーバの優先度は, サーバの次に実行すべき *Job* のあるタスク優先度と順に入れ替わる. 優先度入れ替えのルールは ***PE*** 法に従う.

サーバ周期 T_s が短い場合, サーバの優先度は高くなる. すなわちサーバのキャパシティ C_s を使って, 高優先度で τ_p が動作可能である.

以下にサーバのキャパシティ, 周期の候補を求めるアルゴリズムを示す.

定義 13 (VS の候補)

Γ をスケジュール可能なタスクセット, τ_p を応答時間改善の対象となる特定タスクとする. タスク $\tau_1, ..., \tau_p$ に対し式 *1* の *RTA* を実施する. τ_p の応答時間 R_p と, τ_p より高優先度のタスク $\tau_1, ..., \tau_{p-1}$ の周期 $T_1, ..., T_{p-1}$ の関係より, 以下の式で ***VS*** のキャパシティ C_s と, 周期 T_s の候補を得る.

$$C_s = \begin{cases} C_p, & if\ R_p \le T_{p-1} \quad (5) \\ idle'(T_s), & otherwise \quad (6) \end{cases}$$

$$T_s = \begin{cases} T_h, & if\ R_p \le T_{p-1} \quad (7) \\ t \in \Psi, & otherwise \quad (8) \end{cases}$$

where

$$\Psi = \{T_1, T_2, ..., T_{p-1}\} \quad (9)$$

$$T_h = min(\ \{t \mid t \in \Psi,\ R_p \le t\}\ ) \quad (10)$$

$$idle'(t) = t - \sum_{j=1}^{p-1} \lceil \frac{t}{T_j} \rceil Cj \quad (11)$$

以下, VS の候補からどのように VS を決定しスケジュールするかを説明する. 最初に式 (5), (7) で得られる VS について説明し, 式 (6), (8) で得られる (otherwise の) VS は後述する.

式 (5), (7) は, τ_p を τ_p より優先度の高いタスクと同等の優先度で動作可能とする VS を定義している. 以下にこの式で VS が得られるタスクセットの例を図で示す. $R_p \nleq T_1$ だが $R_p \leq T_2$ であるため, $\mathsf{VS} = (C_p, T_2) = (1, 6)$ となる.

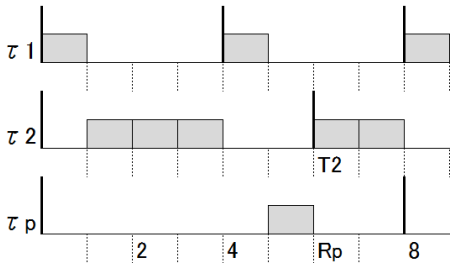

図 3-9. $R_p \leq T_{p-1}$ となる例

定理 14 (VS によるスケジュール (優先度の入れ替え))

スケジュール可能なタスクセット Γ に対し, 式 (5), (7) によって得られた *VS* を使い, τ_p の優先度と τ_p より高優先度のタスク τ_h の優先度を入れ替えた新たなタスクセット Γ' を得る. このとき Γ' はスケジュール可能である.

証明

VS を得るにあたり $R_i \leq T_h$ を満たすため, 定理 6 により Γ' のスケジューラビリティが保たれる.

(証明終わり)

例 15 (**ERD** 法 - 1)

$\Gamma = \{(2, 4), (3, 12), (3, 14)\}$ に対し, $\tau_p = \tau_3$ とする. *RTA* により得られた各タスクの応答時間は $R_1 = 2, R_2 = 7, R_3 = 12$ となる.

このとき, $R_3 \leq T_2 (= 12)$ であるので, 式 (5), (7) より $VS = (3, 12)$ が得られる.

RM 法と ***ERD*** 法によるスケジュールの結果を以下に図で示す.

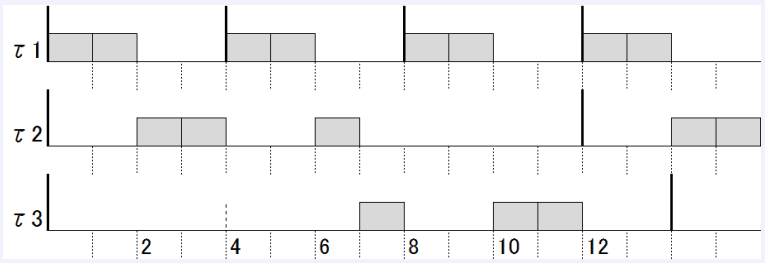

図 *3-10.* ***RM*** 法によるスケジューリング例

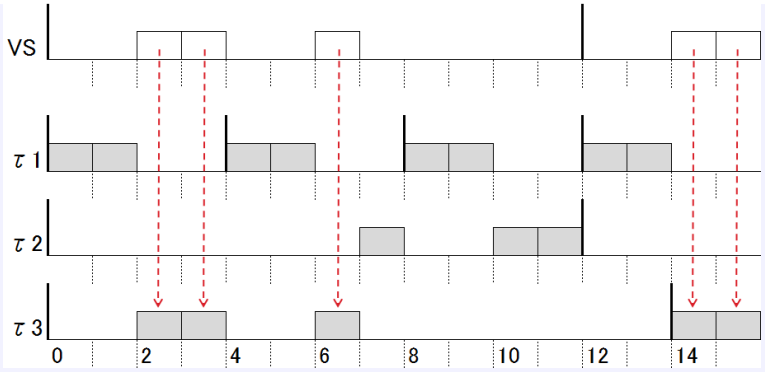

図 *3-11.* ***ERD*** 法によるスケジューリング例

時刻 *2, 3, 6, 14, 15* においてサーバの実行権の移譲が発生する．この例では ***ERD*** 法により τ_3 の応答時間は *12* から *7* に短縮した．

ERD 法では以下の定理および定義により, 応答時間をさらに短縮可能である．

定理 16 (再帰的な VS の適用)

式 (5), (7) で得られた *VS* を使いスケジューリングをした結果の τ_p の応答時間 R'_p について, $R'_p \leq T_h$ となる場合, 定義 *13* を再帰的に適用し, 新たな *VS'* を求めてスケジューリング可能である．

証明

式 (5), (7) によって得られた VS をタスクとみなし, 代わりに τ_p を Γ より省き, 新たなタスクセット Γ' を得たとする．すなわち $\Gamma = \{\tau_1, \tau_2, ..., \tau_{p-1}, \tau_p, \tau_{p+1}, ...\}, \Gamma' = \{\tau_1, \tau_2, ..., \tau_{p-1}, \mathsf{VS}, \tau_{p+1}, ...\}$ である．VS は Γ に含まれる τ_p 以外のタスクのデッドライン制約を満たすようキャパシティと周期を求めたので, Γ' はスケジュール可能である．Γ' はスケジュール可能なタスクセットであるため定義 13 によって新たな VS' と, VS' を使った新たなスケジュール可能なタスクセット Γ'' を求めることが可能である．

(証明終わり)

定義 17 (VS の周期の短縮)

式 (5), (7) によって得られた VS について, もし $R_p \leq T_1$ であれば *VS* の周期 T_s を C_p として良い (すなわち $VS = \{C_p, C_p\}$[a] である) [b].

[a] あるタスクについて, 周期と実行時間を同じにしてしまうと CPU 占有率が 100% となってしまうが, **ERD** 法におけるサーバは, その実行権を別タスクへ移譲するのみである．よってスケジューラビリティへの影響が無い．

[b] **ERD** 法では, τ_p の周期 T_p と, VS の周期 T_s にズレがある場合, 優先度の交換がいかずキャパシティをロスする例がある．具体的な例は第 5 章の考察で述べる．定義 17 を使うことで, 少なくとも τ_p が最高優先度で動作する場合においてはロスが起きなくなり, 応答時間について **ERD** 法は **DM** 法と同じ性能となる．

例 18 (**ERD** 法 - 2)

$\Gamma = \{(2,5),(2,8),(2,10)\}$ に対し, $\tau_p = \tau_3$ とする. RTA により得られた各タスクの応答時間は $R_1 = 2, R_2 = 4, R_3 = 8$ となる.

このとき, $R_3 \leq T_2 (= 8)$ であるので, 式 (5), (7) より $VS = \{2,8\}$ が得られる. ***RM*** 法と ***ERD*** 法によるスケジューリング結果を以下に図で示す.

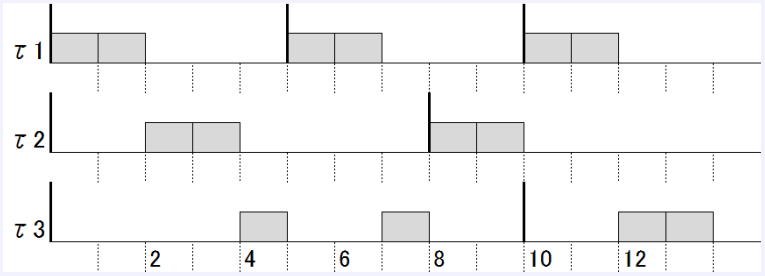

図 *3-12.* ***RM*** 法によるスケジューリング例

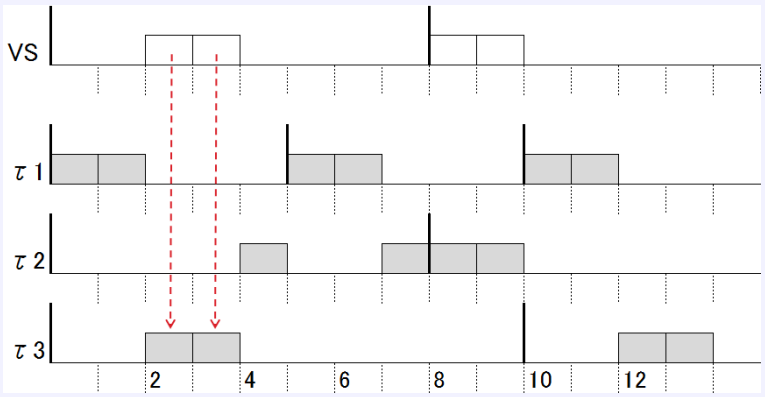

図 *3-13.* ***ERD*** 法によるスケジューリング例

時刻 *2, 3* においてサーバの実行権の移譲が発生する. 時刻 *8, 9* において, *VS* と τ_2 の優先度が ***PE*** 法のルールに従い交換される. すなわちサーバの優先度は τ_2 まで下がってしまう. 時刻 *10* で τ_3 の *Job* がリリースされたが, サーバの優先度は同じ時刻にリリースされた τ_1 より低くなったため, τ_1 の *Job* が優先となる.

ERD 法によるスケジューリングの結果, $R'_1 = 2, R'_2 = 8, R'_3 = 4$ となり, $R'_3 \leq T1 (= 5)$ となった. よって定理 *16* より, 式 (5), (7) を使い新たな $VS' = (2,5)$ が得られるが, $R_p \leq T_1$ であるため定義 *17* により $VS' = (2,2)$ となる. *VS'* を用いたスケジュールの結果を以下に図で示す.

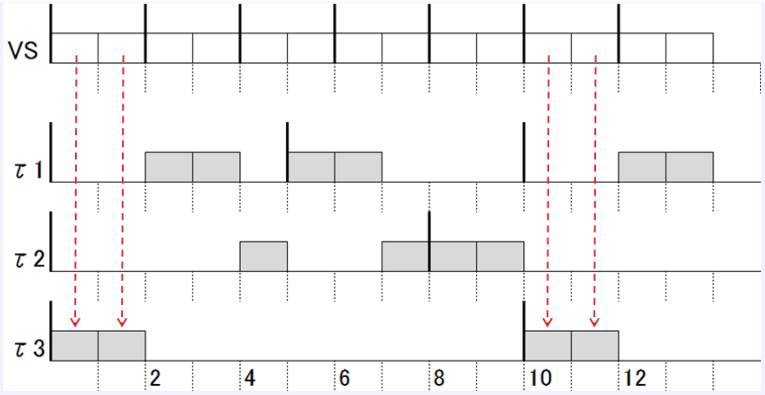

図 *3-14.* ***ERD*** 法によるスケジューリング例

このケースでは τ_p は最高優先度で動作可能となった.

式 (6), (8) で得られる VS は 1 つ以上の候補がある (VS の周期 T_s は, 集合 Ψ から選択する). すなわち, T_p より短い周期の集合 $\Psi = \{T_1, T_2, ...T_{p-1}\}$ に含まれる各周期までの間の, τ_p の Job の実行された部分の長さ $(= idle'(T_i))$ により, 複数の候補を持つ. 定義 7 の $idle()$ と式 (11) の $idle'()$ は異なる. 前者が任意の時刻における Γ 全体を考えたときのアイドル時間を示しているのに対し, 後者は $\tau_1, ..., \tau_{p-1}$ から成る Γ のサブセットにおける, 特定周期のタイミングにおけるアイドル時間を計算する ¶.

以下に図による例を示す. $\Psi = \{T_1, T2\}$ であり, $T_1 = 4, T_2 = 6$ である. $idle'(4) = 1, idle'(6) = 2$ であるため VS は $(1,4), (2,6)$ 二つの候補がある.

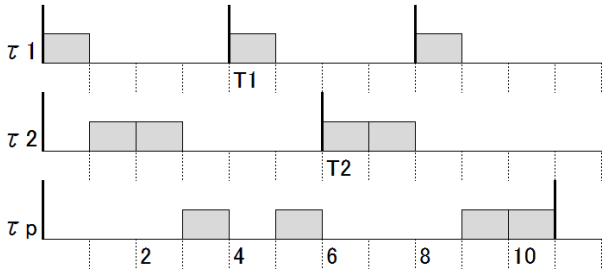

図 3-15.$R_p > T_(p-1)$ となる例

式 (6), (8) で得られる VS を用いてスケジューリングする場合, τ_p の最初にリリースする Job は必ず分割して実行される. 分割実行される場合, VS の候補によっては応答時間短縮に効果は無いが, ジッタ短縮に効果のある場合がある. 例えば図 3-15 において VS $= (1,4)$ を選択した場合, タスクの一部が高優先度で動作可能となる. このとき, ジッタの短縮に意味のある一部 (例:センサからの入力) をタスクプログラムの前半に配置すれば, その部分は高優先度で実行されるため意味的にジッタを短縮可能である. また, タスクは最悪実行時間よりも早期に終了することが一般的であり, その実行時間がサーバのキャパシティよりも短くなる場合は応答時間の大幅な短縮が期待できる.

¶式 (11) にはさらに制約がある. $\Gamma = \{(2,4), (1,5), (1,6)\}, \tau_p = \tau_3$ を考えた時, $idle'(4) = 1$ であるが $idle'(5) = 0$ となってしまう. **ERD** 法において, このとき, $VS = (1,4)$ は得られるが $VS = (1,5)$ は得られないため, 周期のズレへの対応に関し一部制約が出る.

定理 19 (VS によるスケジュール (タスクの分割))

スケジュール可能なタスクセット Γ に対し, 式 (6), (8) によって得られる *VS* により分割して実行するタスク τ_p はデッドライン制約を満たす. また, τ_p を含む, 新たなタスクセット Γ' もスケジュール可能である.

証明

τ_p より優先度の高いタスクからなる集合 $\Gamma_{high} = \{\tau_1, \tau_2, ..., \tau_{p-1}\}$, τ_p より優先度の低いタスクからなる集合 $\Gamma_{low} = \{\tau_{p+1}, ..., \tau_n\}$, および分割して実行される τ_p について, それぞれスケジューラビリティとデッドライン制約が満たされることを示す.

τ_p の Job に対し, VS によって実行される部分を τ_{ps}, 残りの実行部分を τ_{po} とする. 但し, VS によって Job の全体が実行された場合 τ_{po} は存在しない.

Job の一部の早期実行がその Job 全体の応答時間を延長することが無いのは自明である. よって τ_p がスケジュール可能なタスクセット Γ に含まれることから, τ_{po} はデッドライン制約を満たす.

次に τ_{ps} が Γ_{high} のスケジューラビリティに影響を与えないことを示す. VS の存在は, Γ_{high} に対して, 時刻 T_p に関してアイドル時間 ($idle'(T_p)$) が存在することを意味する. よって補題 10 より Γ_{high} には τ_{idle} が追加可能である. この τ_{idle} は VS, すなわち τ_{ps} に他ならないため, Γ_{high} はスケジュール可能である.

Γ_{low} に関して, 補題 5 の証明と同じく, Γ_{low} より高優先度のタスクの実行順の入れ替えにより Γ_{low} のタスクに実行権が渡る時刻は変わらないため, Γ_{low} はスケジュール可能である. (証明終わり)

例 20 (**ERD** 法 - 3)

$\Gamma = \{(1,5), (1,6), (2,8), (4,14)\}$ に対し, $\tau_p = \tau_4$ とする. *RTA* により得られた各タスクの応答時間は $R_1 = 1, R_2 = 2, R_3 = 4, R_4 = 14$ となる.

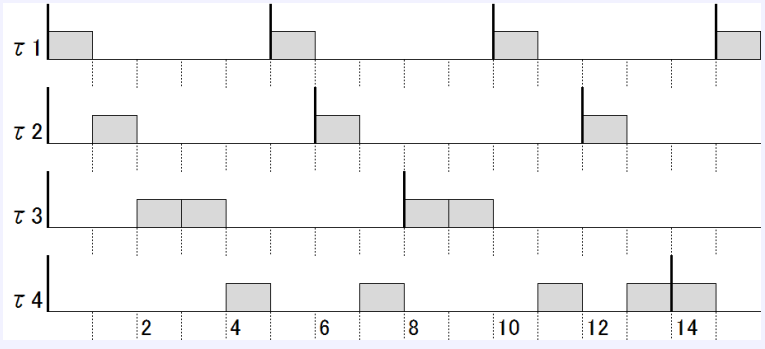

図 *3-16.* ***RM*** 法によるスケジューリング例

このとき, R_4 は T_3 より大きいため, 式 (6), (8) により *VS* を求める. $\Psi = \{T_1, T_2, T_3\}$ であるので, それぞれの周期について $idle'$ を求める. まず, $idle'(T_1)$ を求める.

$$
\begin{aligned}
idle'(T_1) &= T_1 - \sum_{i=1}^{3} \lceil \frac{T_1}{T_i} \rceil Ci \\
&= T_1 - (\lceil \frac{T_1}{T_1} \rceil C_1 + \lceil \frac{T_1}{T_2} \rceil C_2 + \lceil \frac{T_1}{T_3} \rceil C_3) \\
&= 5 - (\lceil \frac{5}{5} \rceil 1 + \lceil \frac{5}{6} \rceil 1 + \lceil \frac{5}{8} \rceil 2) \\
&= 5 - (1 + 1 + 2) \\
&= 1
\end{aligned}
$$

よって $VS_1 = (1, 5)$ である. しかし VS_1 によるスケジューリングの結果は $R'_4 = 14$ となり, 応答時間は短縮されない. [a]

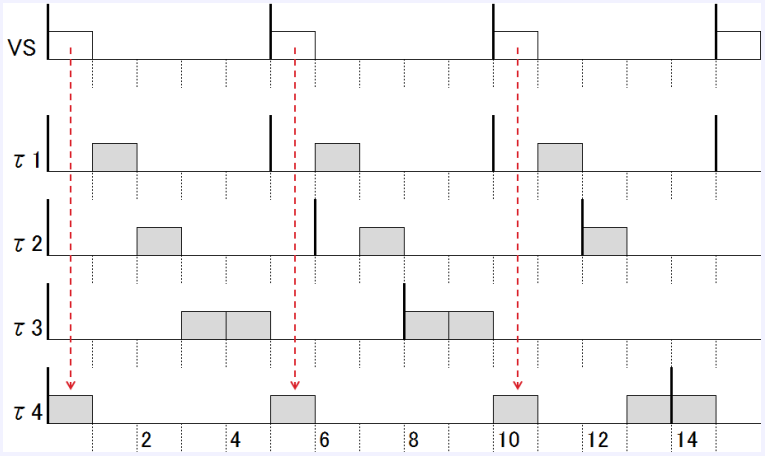

図 *3-17.* ***ERD*** 法 *(VS_1)* によるスケジューリング例

同様に $idle'(T_2)$ を求める.

$$
\begin{aligned}
idle'(T_2) &= T_2 - \sum_{i=1}^{3} \lceil \frac{T_2}{T_i} \rceil Ci \\
&= T_2 - (\lceil \frac{T_2}{T_1} \rceil C_1 + \lceil \frac{T_2}{T_2} \rceil C_2 + \lceil \frac{T_2}{T_3} \rceil C_3) \\
&= 6 - (\lceil \frac{6}{5} \rceil 1 + \lceil \frac{6}{6} \rceil 1 + \lceil \frac{6}{8} \rceil 2) \\
&= 6 - (2 + 1 + 2) \\
&= 1
\end{aligned}
$$

よって $VS_2 = (1, 6)$ から, VS_2 による結果は $R'_4 = 14$ となり, 応答時間は短縮されない.

次に $idle'(T_3)$ を求める.

$$
\begin{aligned}
idle'(T_3) &= T_3 - \sum_{i=1}^{3} \lceil \frac{T_3}{T_i} \rceil Ci \\
&= T_3 - (\lceil \frac{T_3}{T_1} \rceil C_1 + \lceil \frac{T_3}{T_2} \rceil C_2 + \lceil \frac{T_3}{T_3} \rceil C_3) \\
&= 8 - (\lceil \frac{8}{5} \rceil 1 + \lceil \frac{8}{6} \rceil 1 + \lceil \frac{8}{8} \rceil 2) \\
&= 8 - (2 + 2 + 2) \\
&= 2
\end{aligned}
$$

よって $VS_3 = (2, 8)$ である．VS_3 によるスケジューリングの結果，$R'_4 = 10$ となる．よって VS_3 のほうが応答時間に関して有利であるので，$VS = VS_3$ とする．

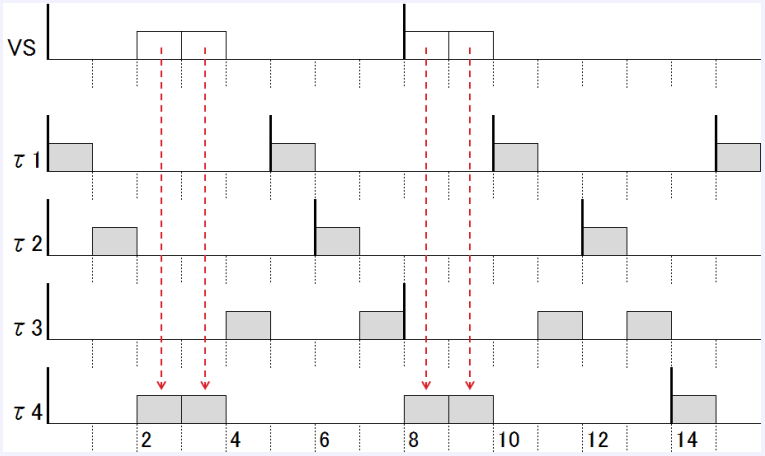

図 3-18. ***ERD*** 法 (VS_3) によるスケジューリング例

上記のタスクセットは ***DM*** 法では応答時間が短縮されない．τ_p の相対デッドラインを τ_p より高優先度のタスクの周期以下にする必要があるが，デッドラインを T_3 以下とすると τ_3 にデッドラインミスが発生するためである．

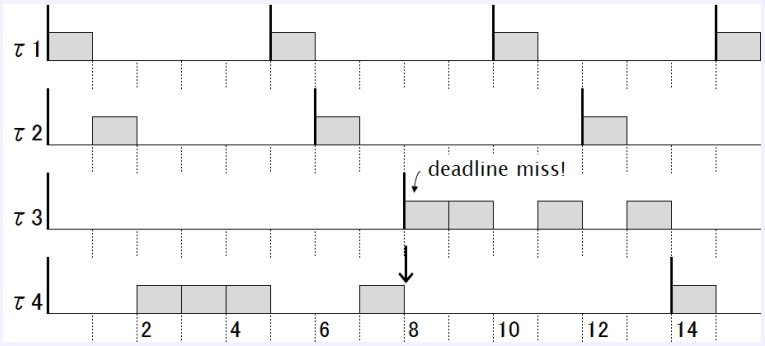

図 3-19. ***DM*** 法によるスケジューリング例

[a] ただし，τ_p の一部を優先的に実行して意味的なジッタを短縮する目的には，VS_1 は有効である．

3.3 Slack Collection (SC) 法

現実世界におけるリアルタイムシステムでは, タスクは通常, 最悪実行時間より早期に終了する. **SC** 法では, 早期終了により発生した時間的スラックを優先的に τ_p が使用する.

例 21 (**SC** 法)

図 *3-20* に *RM* 法の, 図 *3-21* に $\tau_p = \tau_3$ とした場合の ***SC*** 法によるスケジューリング例を示す. この例では τ_1 と τ_3 が常に最悪実行時間の半分で早期終了する. *RM* 法では τ_3 の最初の実行の応答時間は *4* であるが, *SC* 法では τ_1 の早期終了によるスラックを τ_3 が優先的に使用することで, *2* に短縮される.

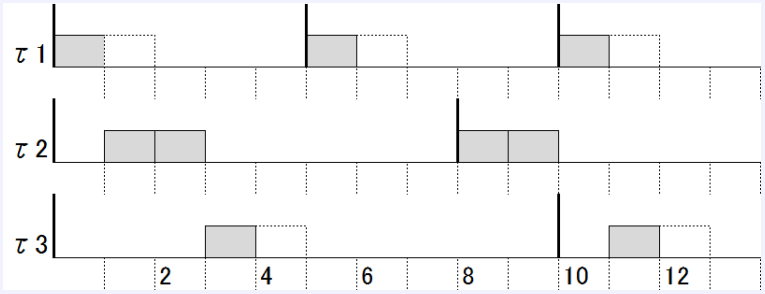

図 *3-20.* ***RM*** 法によるスケジューリング例

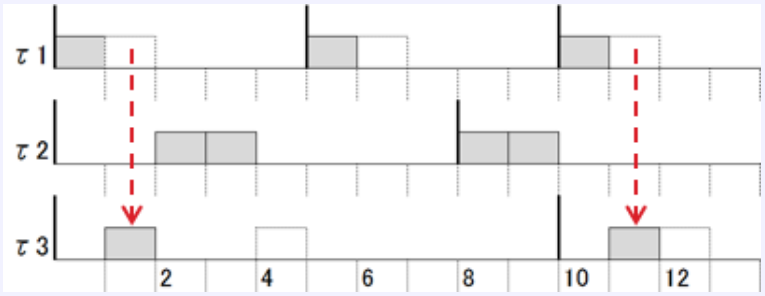

図 *3-21.* ***SC*** 法によるスケジューリング例

4 評価

4.1 評価環境

提案手法を **RM** 法，および特定タスクに最適な相対デッドラインを設定する **DM** 法[‖]と比較する．評価において，周期，実行時間に関して確率分布乱数によって生成したタスクセットを対象とするシミュレーションを行った．タスクセットは 100 作成し, 各タスクセットには $\tau_1, \tau_2, ..., \tau_n$ までタスクが含まれ[**], 数字が小さいほど優先度が高い．評価では, 中間の優先度を持つ τ_3 〜 τ_7 のタスクに注目し, これらの一つを応答時間短縮の対象とし, 時刻 10,000 までスケジューリングのシミュレーションを行った.

4.2 評価結果 - ERD 法

表 1 にタスク当たりのプロセッサ最大使用率が 25%, 表 2 にタスク当たりのプロセッサ最大使用率が 35%, 表 3 にタスク当たりのプロセッサ最大使用率が 50% とした場合の平均タスク応答時間を示す．各表における値は, 5 つのタスク (τ_3 〜 τ_7) のうちのひとつを応答時間短縮の対象タスク τ_p とした場合の平均応答時間, および平均値を, RM 法を 1 とした相対値として示している．この評価ではスラックは発生しない．表中の下線は, **DM** 法と比較し提案手法が優位となった結果を示している.

表 1:評価結果 **ERD** 法 (最大使用率 $\leq 25\%$)

τ_p	**RM**	**DM**	**ERD**
τ_3	1.000	0.676	<u>0.663</u>
τ_4	1.000	0.593	<u>0.581</u>
τ_5	1.000	0.472	<u>0.471</u>
τ_6	1.000	0.412	<u>0.406</u>
τ_7	1.000	0.365	<u>0.355</u>
平均	1.000	0.523	<u>0.515</u>

表 2:評価結果 **ERD** 法 (最大使用率 $\leq 35\%$)

τ_p	**RM**	**DM**	**ERD**
τ_3	1.000	0.723	<u>0.660</u>
τ_4	1.000	0.655	<u>0.594</u>
τ_5	1.000	0.404	<u>0.402</u>
τ_6	1.000	0.301	<u>0.290</u>
τ_7	1.000	0.258	0.258
平均	1.000	0.500	<u>0.468</u>

[‖] τ_p のデッドラインを C_p に設定し, その他のタスクは周期と等しいデッドラインを持つ方式である.

[**] 乱数のため n はタスクセットによって異なる．最低で 3, 最高で 10, 平均値は 6.6 である．評価結果における平均値は, 対象タスクセットの数を重みとして掛けている.

表 3:評価結果 **ERD** 法 (最大使用率 $\leq 50\%$)

τ_p	**RM**	**DM**	**ERD**
τ_3	1.000	0.642	0.571
τ_4	1.000	0.521	0.459
τ_5	1.000	0.394	0.325
τ_6	1.000	0.248	0.250
τ_7	1.000	0.130	0.130
平均	1.000	0.421	0.376

タスクのプロセッサ最大使用率が 25% 以下である場合は全ての評価において **ERD** 法が **DM** 法よりも優位であった．タスクのプロセッサ使用率が 35%, 50% となる場合は, あらかじめ優先度の比較的高い τ_3 〜 τ_5 のタスクにおいては 25% の結果に比べ応答時間短縮の割合がより高い結果となった．逆に元の優先度が低い場合は **DM** 法と同等か, 劣る結果となった．いずれの表においても, 平均値を見た場合, タスクのプロセッサ最大使用率が高いと **ERD** 法に有利な傾向が見られる．

以下に, タスク平均応答時間の関係をグラフで示す．

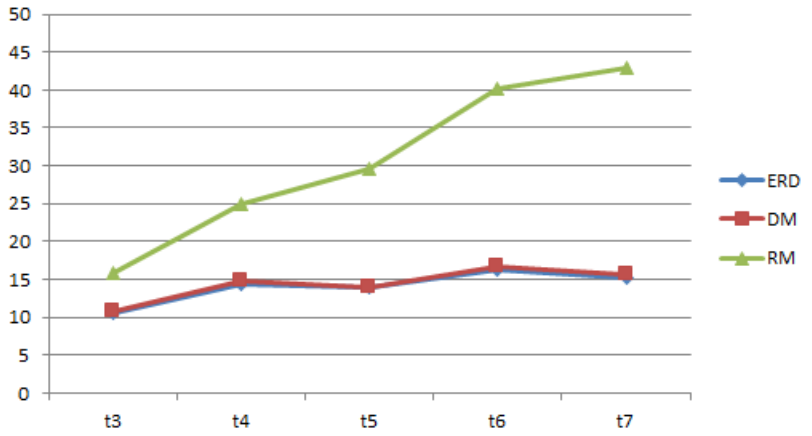

図 4-1. タスク平均応答時間 (最大使用率 $\leq 25\%$)

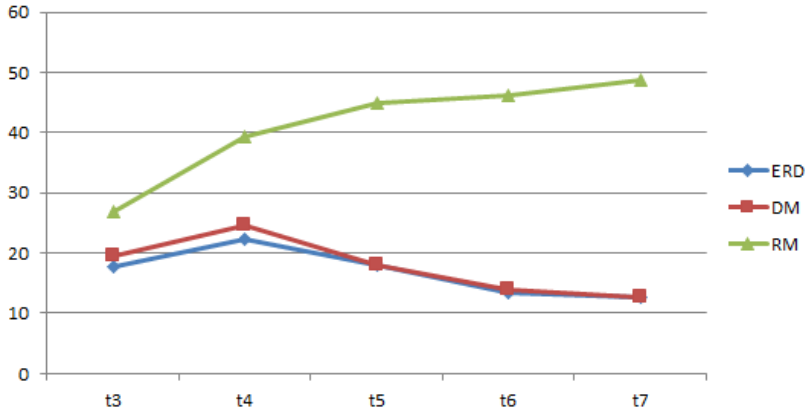

図 4-2. タスク平均応答時間 (最大使用率 $\leq 35\%$)

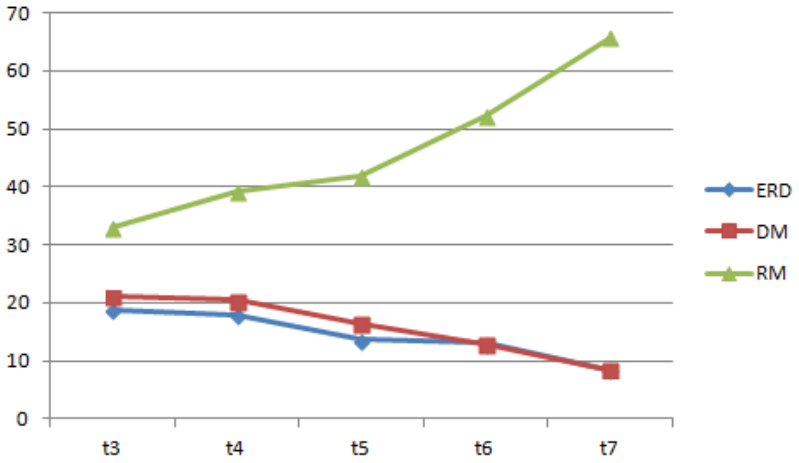

図 4-3. タスク平均応答時間 (最大使用率 $\leq 50\%$)

4.3 評価結果 - ERD+ SC 法

表 4, 5, 6 に, **ERD** 法に **SC** 法を追加した場合の, 平均応答時間の相対値を示す. タスクは平均して 78% の時間で早期終了する.

表 4:評価結果 **ERD+ SC** 法 (早期終了 $= 78\%$, 最大使用率 $\leq 25\%$)

τ_p	**RM**	**DM**	**ERD+ SC**
τ_3	1.000	0.715	0.705
τ_4	1.000	0.708	0.701
τ_5	1.000	0.542	0.542
τ_6	1.000	0.491	0.485
τ_7	1.000	0.440	0.438
平均	1.000	0.599	0.593

表 5:評価結果 **ERD+ SC** 法 (早期終了 $= 78\%$, 最大使用率 $\leq 35\%$)

τ_p	**RM**	**DM**	**ERD+ SC**
τ_3	1.000	0.754	0.704
τ_4	1.000	0.686	0.638
τ_5	1.000	0.489	0.488
τ_6	1.000	0.361	0.347
τ_7	1.000	0.321	0.323
平均	1.000	0.552	0.527

表 6:評価結果 **ERD+ SC** 法 (早期終了 $=78\%$, 最大使用率 $\leq 50\%$)

τ_p	**RM**	**DM**	**ERD+ SC**
τ_3	1.000	0.700	<u>0.651</u>
τ_4	1.000	0.576	<u>0.526</u>
τ_5	1.000	0.513	<u>0.452</u>
τ_6	1.000	0.297	0.298
τ_7	1.000	0.137	0.149
平均	1.000	0.484	<u>0.449</u>

以下に, タスク平均応答時間の関係をグラフで示す.

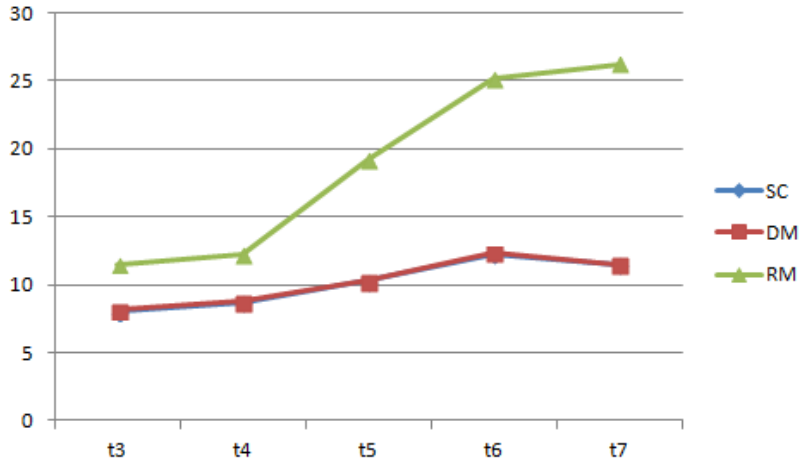

図 4-4. タスク平均応答時間 (早期終了 $=78\%$, 最大使用率 $\leq 25\%$)

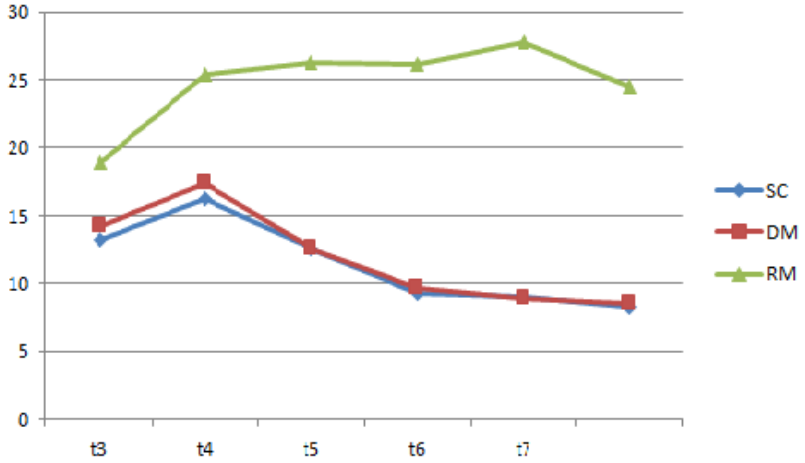

図 4-5. タスク平均応答時間 (早期終了 $=78\%$, 最大使用率 $\leq 35\%$)

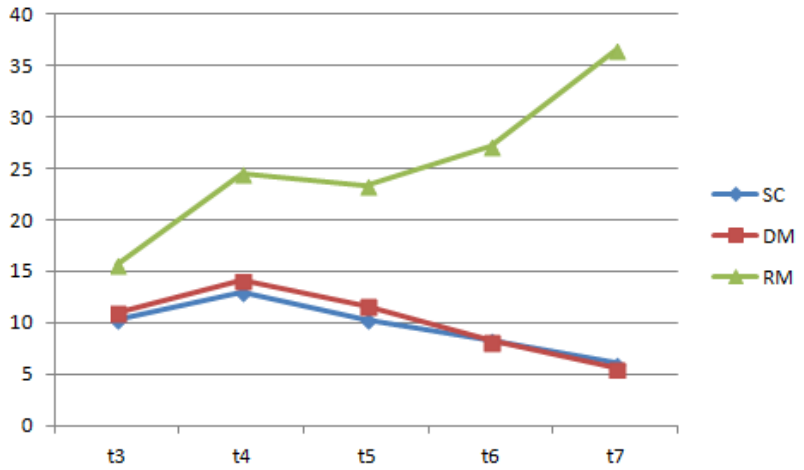

図 4-6. タスク平均応答時間 (早期終了 = 78%, 最大使用率 $\leq$ 50%)

表 7, 8, 9 に, タスクが平均して 56% の時間で早期終了する結果を示す.

表 7:評価結果 **ERD+ SC** 法 (早期終了 = 56%, 最大使用率 $\leq$ 25%)

τ_p	**RM**	**DM**	**ERD+ SC**
τ_3	1.000	0.755	<u>0.747</u>
τ_4	1.000	0.708	<u>0.700</u>
τ_5	1.000	0.683	0.735
τ_6	1.000	0.553	<u>0.548</u>
τ_7	1.000	0.496	<u>0.490</u>
平均	1.000	0.658	0.664

表 8:評価結果 **ERD+ SC** 法 (早期終了 = 56%, 最大使用率 $\leq$ 35%)

τ_p	**RM**	**DM**	**ERD+ SC**
τ_3	1.000	0.784	<u>0.742</u>
τ_4	1.000	0.725	<u>0.683</u>
τ_5	1.000	0.556	<u>0.553</u>
τ_6	1.000	0.420	<u>0.400</u>
τ_7	1.000	0.388	0.393
平均	1.000	0.602	<u>0.579</u>

表 9:評価結果 **ERD+ SC** 法 (早期終了 = 56%, 最大使用率 $\leq$ 50%)

τ_p	**RM**	**DM**	**ERD+ SC**
τ_3	1.000	0.700	<u>0.651</u>
τ_4	1.000	0.618	<u>0.567</u>
τ_5	1.000	0.557	<u>0.489</u>
τ_6	1.000	0.347	0.347
τ_7	1.000	0.188	0.214
平均	1.000	0.519	<u>0.484</u>

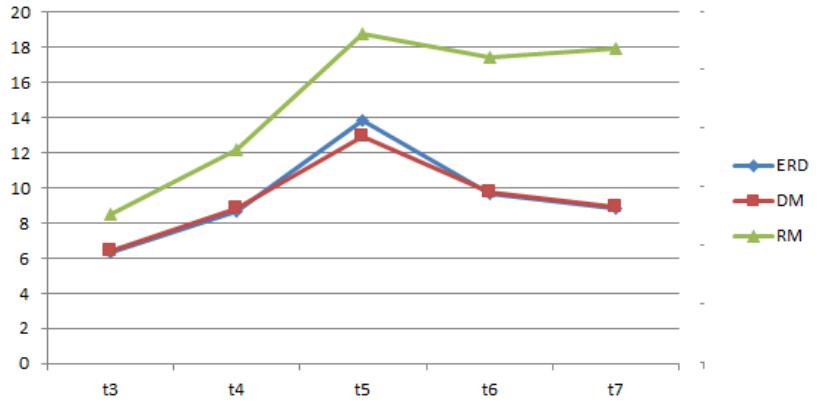

図 4-7. タスク平均応答時間 (早期終了 = 56%, 最大使用率 $\leq$ 25%)

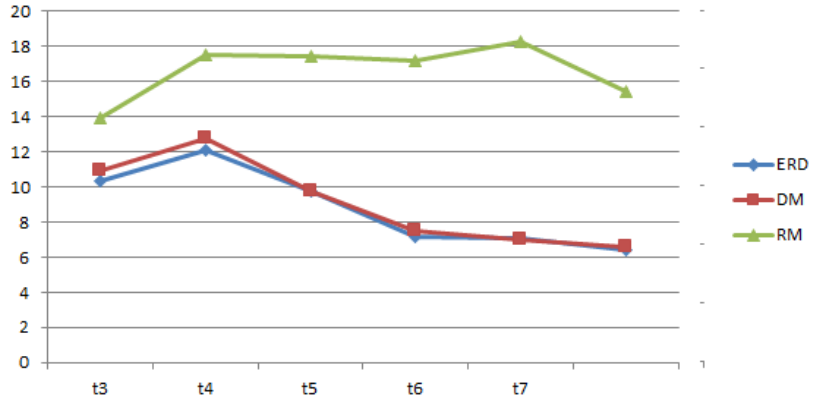

図 4-8. タスク平均応答時間 (早期終了 = 56%, 最大使用率 $\leq$ 35%)

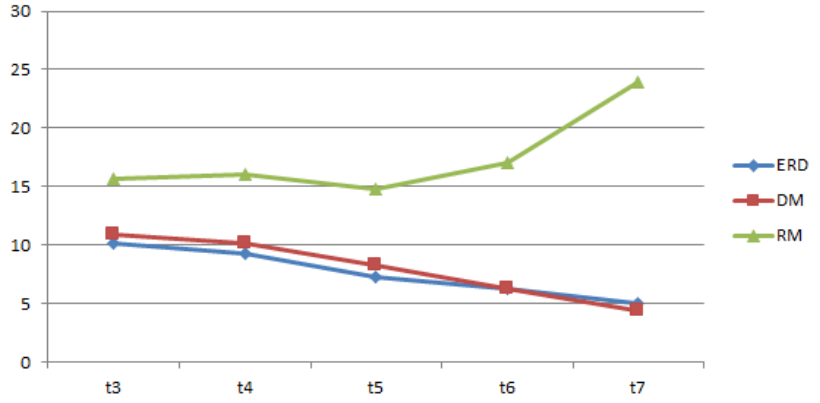

図 4-9. タスク平均応答時間 (早期終了 = 56%, 最大使用率 $\leq$ 50%)

ERD + SC 法の評価結果も, **ERD** 法単体の評価と同じ傾向となった. すなわちタスクのプロセッサ使用率が 25% の場合は短縮度合は少ないものの **DM** 法よりも優位となり, タスクのプロセッサ使用率が 35%, 50% の場合は, $\tau_3 \sim \tau_5$ のタスクについて短縮度合が高くなる.

5　考察

以上の結果から, 各タスクの最大使用率が高くなると **ERD+ SC** 法に有利なことが考察できる. **DM** 法では相対デッドラインを短くすることが難しくなる場合でも, **ERD+ SC** 法では τ_p が特に高優先度サーバの (短い) キャパシティ内で早期終了する場合に, 応答時間が短縮可能となる.

一方で, 一部では **DM** 法より **ERD+ SC** 法が劣る結果も得られた. **ERD** 法では τ_p の周期 T_p よりも短い周期のサーバを用いるが, T_p と VS の周期にズレが生じ, かつ優先度の交換によりキャパシティがロスする場合, **DM** 法に劣る結果となる. この傾向は τ_p の優先度が低く, かつ周期自体が絶対的に長い場合に見受けられた. 以下は, 説明を簡単とするため実際の **ERD** 法では存在しない例であるが [††], 優先度の交換がうまくいかずキャパシティがロスする一例である.

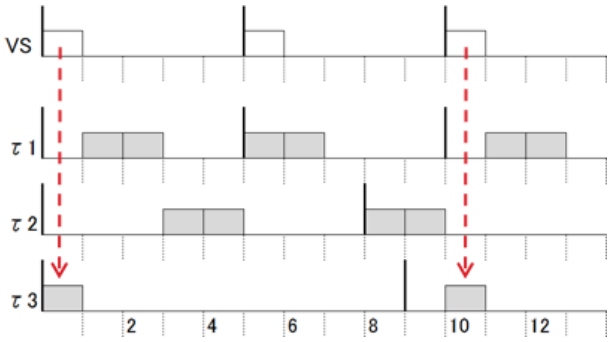

図 5-1. キャパシティのロスが起きる例

$\tau_p = \tau_3$ とする. 時刻 0 において VS のキャパシティを使い τ_3 の Job が動作する. 時刻 5 に VS に実行権が渡るが, 既に τ_3 は動作を終えているため τ_1 の優先度と VS の優先度が交換となる. VS のキャパシティは τ_1 の優先度で蓄積され, τ_1 のふたつ目の Job が動作する. 時刻 7 にこの Job は動作を終えるが, この時点で次に動作すべき Job は存在しない. よって **PE** 法のルールに従い, 蓄積された VS のキャパシティは失われる. 時刻 9 に τ_3 のふたつ目の Job がリリースされるが, 次に VS が実行可能状態となるのは時刻 10 であり, τ_2 の Job にブロックされてしまう. 一方で **DM** 法では τ_3 のデッドラインを T_1 以下とすることが可能なので, 時刻 9 においてもリリースされた τ_3 の Job は即実行可能である.

スラックの利用に関しては, **SC** 法と **DM** 法において性能差は無かった. 評価に用いた確率分布乱数により生成したタスクセットでは **DM** 法でも比較的優先度を上げやすいものが多く, スラックが発生すると結果的に優先度を上げたい特定タスクが動作可能となる例が多かったためである.

実システムに **ERD** および **SC** 法を適用する場合, VS の候補の計算に必要な RTA は NP 困難のクラスに属するため, 動的にタスクが追加されるシステムではなく, あらかじめタスクセットが固定されているシステムを対象とすべきである. RTOS のスケジューラ部分に変更が必要であるが, VS が実行状態となったさい, レディキューに τ_p が繋がれているか, また実行権移譲により τ_p が動作した後に VS のキャパシティに残りがあるか判断する処理を追加する程度で良く, 実行時オーバーヘッドは十分小さいと言える.

[††] **ERD** 法では VS= (1, 1) とできる.

6　まとめ

本研究では，周期は長いが意味的に重要なタスクのリアルタイム性を向上するスケジューリングアルゴリズムの提案を行った. スケジューリングアルゴリズムの提案に伴い, 定理・補題を示した. タスクセットに含まれるタスク優先度を一定条件のもとで変更しても, タスクセット全体のスケジューラビリティが損なわれないことを示した (補題 5, 定理 6). また, 一定条件のもとでタスクセットに新たな (高優先度の) タスクを追加してもタスクセット全体のスケジューラビリティが損なわれないことを示した (補題 10). 続いてスケジューリングアルゴリズム **ERD** 法と **SC** 法を提案した. **ERD** 法は，仮想的な高優先度サーバを用いることで, タスクセット全体のスケジューラビリティを保ちつつ，特定タスクの応答時間とジッタの短縮を可能とする手法である. **SC** 法は, 最悪実行時間より早期に Job が終了したことで得られる時間的スラックを, 優先的に特定のタスクへ与える手法である. 両手法を組み合わせ, 確率分布乱数で生成したタスクセットに対し評価を行った結果, **DM** 法と比較し，特にタスクの使用率が高くなる場合に応答時間が短縮された.

本研究の今後の発展, 課題について述べる. 一つ目の課題は, **DM** 法のように周期とは独立した優先度を設定可能とするモデルへの拡張である. 優先度を追加することで 5 章の考察で述べたキャパシティのロスは無くなる. **ERD** 法は **DM** 法とは異なりタスクを分割実行可能であるため, 分割実行した場合は **DM** 法と同等か上回る結果となる (分割実行しない場合は **ERD** 法と **DM** 法の性能は等しい). 二つ目の課題として **EDF** ベースの動的優先度モデルへの拡張, マルチコアスケジューリングへの拡張等, ベースモデルの拡張が挙げられる. 三つ目の課題は, 時間以外の資源を加えたモデルへの対応である. 現実のリアルタイムシステムは，各タスクは共有の資源 (メモリ等) へアクセスする等，依存を持つことがほとんどである. 一方で **RM** 法に代表されるスケジューリングアルゴリズムの研究分野では，前提としてタスク間の依存が無い. 依存を認めつつもリアルタイム性・予測性を保つため，セマフォと優先度上限・逆転プロトコルを使う等，スケジューリングアルゴリズムを補足する手法の研究が行われていた [14] [15] が，近年，リアルタイムスケジューリングの研究分野，特にマルチコアスケジューリングの研究において，CPU 時間以外の資源（メモリ等）をスケジューリングの段階で考慮した研究が行われている [16] [17]. これら CPU 時間以外の制約を **ERD**, **SC** 法に追加することで，より現実システムに即したモデルでの研究・評価が可能となる.

謝辞

本研究を行うにあたり, 丁寧な指導を頂きました田中清史准教授に心より感謝申し上げます. 中間審査で適切なご指導を頂きました金子峰雄教授, 井口寧教授, 副テーマを指導して下さった廣川直准教授に感謝申し上げます. また, 情熱的に授業をして下さった全ての先生方, 有意義な議論をして頂いた研究室の皆さまにお礼を申し上げます.

参考文献

[1] Liu, Chung Laung, and James W. Layland., *Scheduling Algorithms for Multiprogramming in a Hard- Real-Time Environment* Journal of the ACM 20(1): 40-61., 1973

[2] http://www.tron.org/ja/wp-content/themes/dp-magjam/pdf/specifications/ja/WG024-S001-04.03.03.pdf

[3] Giorgio C. Buttazzo, *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications.* Springer, 2011

[4] Lehoczky, John P., and Sandra Ramos-Thuel., *An optimal algorithm for scheduling soft-aperiodic tasks in Fixed-priority preemptive systems.* In Proceedings of the IEEE Real-Time Systems Symposium,December , 1992

[5] Sprunt, Brinkley, John Lehoczky, and Lui Sha., *Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm.* Proceedings IEEE Real-Time Systems Symposium, 1988.

[6] Lui Sha, et al, *Real Time Scheduling Theory: A Historical Perspective.* Real-Time Systems, 28, 101-155, 2004

[7] Bini, Enrico, and Giorgio Buttazzo., *A hyperbolic bound for the rate monotonic algorithm.* In Proceedings of the IEEE Euromicro Conference on Real-Time Systems, pages 59-66,June 2001.

[8] Lehoczky, John P., *Enhanced aperiodic responsiveness in hard real-time environments.* In Proceedings of the IEEE Real-Time Systems Symposium, December 1987

[9] Burns, A., et al. *Hard real-time scheduling: the deadline monotonic approach.* Proc. of the IFAC/IFIP workshop, UK., 1992.

[10] Davis, Robert, and Andy Wellings., *Dual Priority Scheduling.* In Proceedings of the 16th IEEE RTSS, 1995.

[11] De Niz, Dionisio, Karthik Lakshmanan, and Ragunathan Rajkumar., *On the scheduling of mixed-criticality realtime task sets.* In Proceedings of the Real-Time Systems Symposium, 2009.

[12] Joseph, Mathai, and Paritosh Pandya., *Finding response times in a real-time system.* BCS Computer Journal, 1986

[13] Eisenbrand, Friedrich, and Thomas Rothvos., *Static-priority real-time scheduling: Response time computation is np-hard.* 2008 Real-Time Systems Symposium. IEEE, 2008

[14] Goodenough, John B., and Lui Sha., *The priority ceiling protocol: A method for minimizing the blocking of high priority Ada tasks.* Vol. 8. No. 7. ACM, 1988.

[15] Baker, and Theodore P., *A stack-based resource allocation policy for realtime processes.* Real-Time Systems Symposium, 1990. Proceedings., 11th. IEEE, 1990.

[16] Yun, Heechul, et al., *Memory access control in multiprocessor for real-time systems with mixed criticality.* 2012 24th Euromicro Conference on Real-Time Systems. IEEE, 2012.

[17] Davis, Robert I., and Alan Burns., *A survey of hard real-time scheduling for multiprocessor systems.* ACM Computing Surveys (CSUR) 43.4 (2011): 35., 2011.