仮想サーバによるタスクの応答時間短縮手法

鈴木 隆元[†1,a)]　田中 清史[†1,b)]

概要：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, Rate Monotonic (**RM**) 法に基づく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，長い周期のタスクは優先度が下がるため，応答時間およびジッタが増大する傾向がある．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ちつつ，高優先度サーバを利用することで優先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．

キーワード：リアルタイムスケジューリング，Rate Monotonic，Deadline Monotonic，Priority Exchange

A Virtual Server for Shortening Task Response Time

SUZUKI Taka'HAL'[†1,a)]　TANAKA Kiyofumi[†1,b)]

Abstract: In fixed-priority scheduling algorithms based on Rate Monotonic (**RM**) method widely used in development of real-time systems, tasks with shorter periods have higher priorities. In contrast, ones with longer periods are likely to suffer from increased response times and jitters due to their lower priorities. This study aims at shortening response times of tasks which have relatively long periods but are important. We propose Execution Right Delegation (**ERD**) method, which introduces a high-priority server for particular tasks to be scheduled preferentially, and Slack Collection (**SC**) method, which provides particular tasks with slack time left by early completion of other tasks. These methods keep schedulability.

Keywords: Real time scheduling, Rate Monotonic, Deadline Monotonic, Priority Exchange

1. はじめに

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．

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**) 法に大別出来る．**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という利点がある一方で, 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．

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, 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（スラック）が生じる．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

[†1] 北陸先端科学技術大学院大学
JAIST, Tokyo 101–0062, Japan
[a)] suzuki.takaharu@jaist.ac.jp
[b)] kiyofumi@jaist.ac.jp

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.

本研究では周期タスクを対象とし, 古典的なモデル, 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 [8] 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.

2. 関連研究

2.1 スケジューリングアルゴリズム

RM 法で扱うモデルでは, タスク τ_i は無限の Job をリリースする. Job はリリースされると規定の時間 C_i の間実行する. また, Job は周期 T_i 毎にリリースされる. タスクは $\tau_i = (C_i, T_i)$ で表記される. 複数のタスクは $\tau_1, \tau_2, ..., \tau_n$ のように表わされ，数字の小さいほうが周期が短く，優先度が高い (すなわち $T_1 \leq T_2 \leq ...T_n$). 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. リリースされた Job は次のリリース (すなわち, 次の周期) までに仕事を終えなければ，デッドラインミスとなる.

RM 法と異なる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. タスク優先度はデッドラインが短いものほど高い. 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, 周期が長いタスクの場合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.

非周期タスクの応答時間を短縮することを目的としたスケジューリングアルゴリズムの一つに Priority Exchange (**PE**) 法 [7] がある. **PE** サーバは周期と，実行時間に相当するキャパシティを持ち，周期タスクセットと共に **RM** によってスケジュールされ，サーバのキャパシティは非周期タスクの実行に使用される. 非周期タスクのリクエストが存在しない場合は, サーバの次に優先度の高い周期タスクが実行され，その実行時間に相当する量のサーバキャパシティの優先度は実行タスクの優先度まで下がる（優先度の交換）. 非周期タスクのリクエストが到着した場合, サーバのキャパシティの範囲およびその優先度で非周期タスクを実行可能である.

Extended Priority Exchange (**EPE**) 法 [5] 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, これを非周期タスクのキャパシティとして使う. これは，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みなすことができる.

Dual Priority (**DP**) 法 [9] は, **RM** 法に代表される方式とは異なり，各タスクが二つの優先度を持つモデルに対するアルゴリズムである. 周期タスクは低，高の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なる. **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.

Zero Slack Scheduling 法 [10] はクリティカリティ混在システムを対象モデルとしており，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, タスクは従来の優先度に加えて Criticality を持つ. タスクは Normal/Critical の二つのモードで動作し, スケジューラビリティを確保するために, 実行のある時刻で 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

Liu らの研究 [1] では, 全タスクの CPU 占有率をもとにスケジュール可能性を判定する. 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 らによって Liu 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 [6].

タスク応答時間を求めることによる判定手法 Responce Time Analysis (RTA) が Joseph らによって提案された [11]. この手法では疑似多項式時間でタスク応答時間を決定可能であり, NP 困難のクラスに属することが近年証明されたが [12], 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与えている.

3. 提案スケジューリング手法

本稿では, 各タスクは実行時間と周期を持ち, 相対デッドラインは周期と等しく，位相は持たない（時刻 0 で最初のリリースが行われる）. また，提案手法の適用対象タスク以外は **RM** 法に基づく静的優先度割当を基本とする. 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 3.1 (Responce Time Analysis(RTA) [11]): 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 [i] は以下で与えられる.

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

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

[i] 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 [11].

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 3.2 (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結果を **図 *1*** に示す.

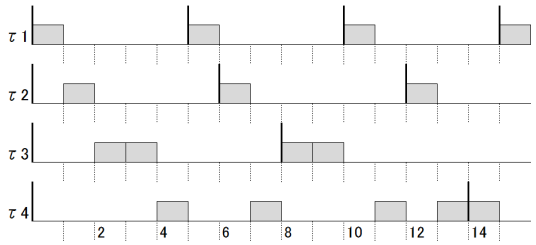

図 1 RM 法によるスケジューリング例

定義 *3.1* 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 3.3 (スケジュール可能性の判定 [11]): 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 3.4 (タスク優先度の設定): ***RM*** 法によるタスク優先度付け方法に加え, 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 3.5 (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: 定理 3.3 より, 優先度を入れ替えた後，全てのタスクについて最長応答時間 R'_i が周期 T_i 以下となることを確認することで，スケジュール可能であることを示す. (以後, 優先度を入れ替えた後の時刻には R'_i 等, カンマをつける．)

タスクセットを，τ_h より高優先度のタスクのグループ Γ_{high}, τ_p より低優先度のグループ Γ_{low}, τ_h と τ_p からなるグループに分割する.

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} も，τ_{low} より高優先度のタスクの実行順の入れ替えにより，τ_{low} のタスクに実行権が渡る時刻は変化しない.

τ_p は元の優先度より高くなるため, 応答時間は増大しない. $T_h \leq T_p$，$R_p \leq T_h$，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，Γ_{high} に含まれるタスクのジョブが τ_p, τ_h をブロックするか否かの二つの場合に分けて考える. 最初にブロックしない場合について，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.

$$R_h = s_h + C_h, \quad R_p = s_p + C_p$$

一方で, 優先度を入れ替えると,R'_p, R'_h は以下となる.

$$R'_p = s'_p + C_p, \quad R'_h = s'_h + C_h$$

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（$s_p = R_h$）. 優先度を入れ替えた後の s'_h も同様である（$s'_h = R'_p$）. $s'_p = s_h$ であるため

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

前提の $R_p \leq T_h$ より，$R'_h \leq T_h$ となりデッドライン制約を満たす．

Γ_{high} にブロックされない場合の例を 図 **2**, 図 **3** に示す．

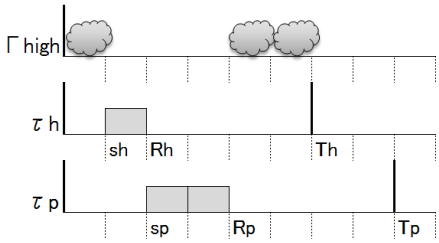

図 **2** Γ_{high} にブロックされない場合 (優先度入れ替え前)

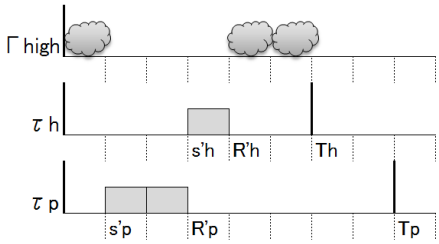

図 **3** Γ_{high} にブロックされない場合 (優先度入れ替え後)

次に, Γ_{high} 内のタスクが T_h までに新たな Job をリリースし, τ_p, τ_h をブロックする場合を扱う．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．τ_h の Job が s_h 以降にブロックされる時間 BT は以下の上限を持つ．

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

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, $s'_p = s_h$，および BT を左辺へ移動することにより，

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

式 (2)　に対し, BT を展開すると，

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

よってデッドライン制約を満たす．

Γ_{high} にブロックされる場合の例を 図 **4**, 図 **5** に示す．

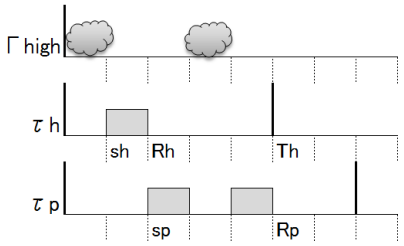

図 **4** Γ_{high} にブロックされるケース

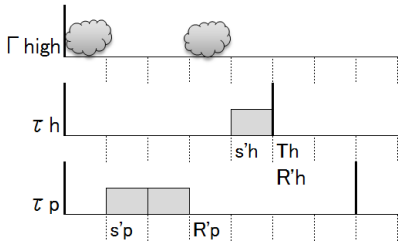

図 **5** Γ_{high} にブロックされるケース (優先度入れ替え後)

■

定理 3.6 (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 [ii].

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 3.5 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 3.5 により優先度を入れ替えてやれば良い． ■

定義 3.7 (アイドル時間): いかなるタスクの *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 3.8 (アイドル時間): 図 ***6*** において, $idle(4) = 1$, $idle(6) = 2$ である．

補題 3.9 (アイドル時間と Job の終了): タスクセット Γ 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

[ii] 補題 3.5 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．

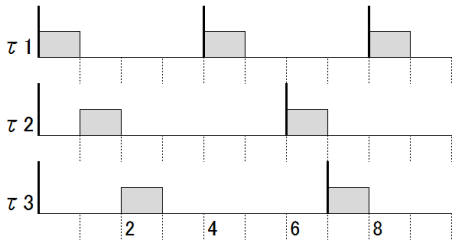

図 6　$idle(t)$

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 が存在しない．すなわち各タスクの最初の Job は実行を終えている． ■

補題 3.10 (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のタスクから成る集合を $\Gamma_{low} = \{\tau_h, \tau_{h+1}, \tau_{h+2}, ..., \tau_n\}$ とする．Γ_{high} に含まれるタスクについては補題 3.5 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 3.9 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 [iii] 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.

[iii] 最大と断ったのは, 図 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.

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

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 3.3 によりデッドライン制約を満たす． ■

例 3.11 (τ_{idle} の追加): 図 6 で示したタスクセットに対し, $\tau_h = \tau_2, T_h = T_2 = 6$ とした場合の τ_{idle} を追加した例を図 **7** に示す.

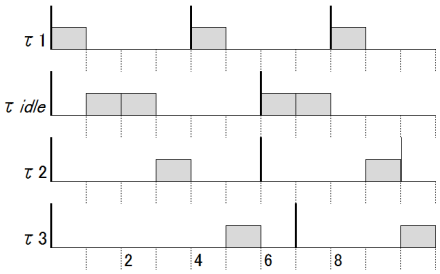

図 7　τ_{idle} の追加

3.2 Execution Right Delegation (ERD) 法

ERD 法は特定の周期タスク τ_p を優先的に実行するための仮想サーバ VS を用い, 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.

定義 3.12 (実行権の移譲): ***ERD*** 法において *VS* は ***RM*** に従ってスケジュールされ, *VS* に実行権が渡った場合, *VS* 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* が既に終了している場合, *VS* の優先度は, 他の *Job* の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.

T_s を短くすることで VS の優先度は高くなり，C_s を使って高優先度で τ_p が動作可能となる．以下に VS 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 3.13 (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 $\tau_1, ..., \tau_p$ に対し定義 *3.1* 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 C_s と T_s

の候補を得る．

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

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

$$where$$

$$\Psi = \{T_1, T_2, ..., T_{p-1}\}$$
$$T_h = min(\ \{t \mid t \in \Psi,\ R_p \le t\}\ )$$
$$idle'(t) = t - \sum_{j=1}^{p-1} \lceil \frac{t}{T_j} \rceil Cj$$

式 (5) ，(7) は，τ_p を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．この式で VS が得られるタスクセットの例を図 8 に示す．$R_p \not\le T_1$ だが $R_p \le T_2$ であるため，$\mathsf{VS} = (C_p, T_2) = (1, 6)$ となる．

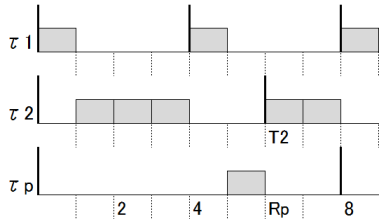

図 8 $R_p \le T_{p-1}$ となる例

定理 3.14 (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 $R_p \le T_h$ を満たすため，定理 3.6 により Γ' のスケジューラビリティが保たれる． ■

例 3.15 (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，$R_3 \le T_2(= 12)$ であるため，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 図 ***9***, 図 ***10*** に示す．時刻 *2, 3, 6* において *VS* 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 3.16 (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，$R'_p \le T_{h-1}$ となる場合，定義 *3.13* を再帰的に適用し，スケジューリング可能となる新たな *VS′* を得ることが可能である．

証 明: 定 理 3.14 に よ り $\Gamma' = \{\tau_1, \tau_2, ..., \tau_{h-1}, \mathsf{VS}, \tau_h, ..., \tau_{p-1}, \tau_{p+1}, ...\}$ はスケジュール可能なタスクセットである．よって定義 3.13 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． ■

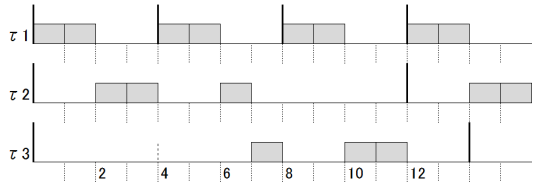

図 9 **RM** 法によるスケジューリング例

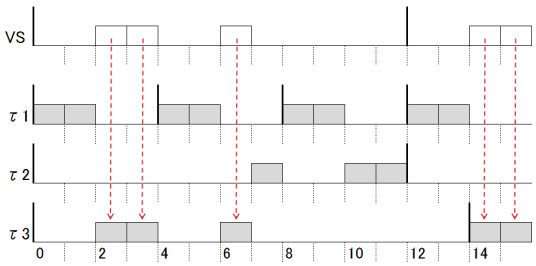

図 10 **ERD** 法によるスケジューリング例

定義 3.17 (VS の周期の短縮): 式 (5) ，(7) によって得られた VS について，$R_p \le T_1$ であれば周期 T_s を C_p として良い（すなわち $VS = \{C_p, C_p\}$ [iv]）．

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，実行権を移譲できずキャパシティをロスする場合がある．（4.4 考察で改めて述べる．）定義 3.17 が適用可能な場合はキャパシティロスが発生しない．

例 3.18 (ERD 法 - 2): $\Gamma = \{(2,5), (2,8), (2,10)\}$ に対し，$\tau_p = \tau_3$ とする．*RTA* により応答時間は $R_1 = 2, R_2 = 4, R_3 = 8$ となる．$R_3 \le T_2(= 8)$ であり，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ルの結果を 図 ***11***, 図 ***12*** に示す．

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，$R'_3 \le T1(= 5)$ となった．よって定理 *3.16* より新たな $VS' = (2, 5)$ が得られるが，さらに $R_p \le T_1$ となるため，定義 *3.17* により $VS' = (2, 2)$ となる．*VS′* を用いたスケジュールの結果を 図 ***13*** に示す．

式 (6) ，(8) で得られる VS は 1 つ以上の候補がある．すなわち，T_p より短い周期の集合 $\Psi = \{T_1, T_2, ...T_{p-1}\}$ に

[iv] 見かけ上の CPU 占有率が 100% となるが，**ERD** 法における VS はその実行権を別タスクへ移譲するのみであり，スケジューラビリティへの影響が無い．

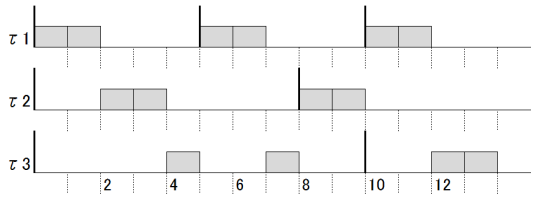

図 11　**RM** 法によるスケジューリング例

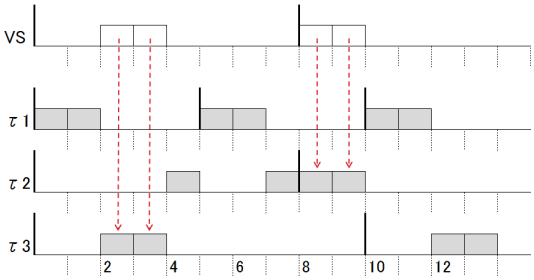

図 12　**ERD** 法によるスケジューリング例

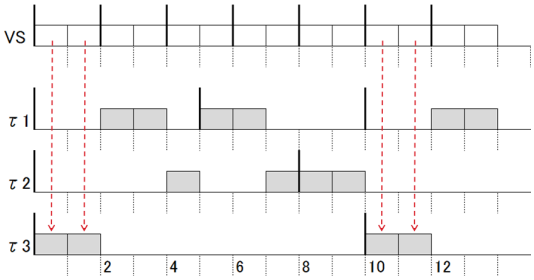

図 13　**ERD** 法によるスケジューリング例（定理 3.16 と定義 3.17 の適用）

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 3.7 の $idle()$ と式 (9)　の $idle'()$ は異なる．前者は任意の時刻における Γ 全体のアイドル時間であるが，後者は $\tau_1, ..., \tau_{p-1}$ から成るタスク群に対する特定周期のタイミングにおけるアイドル時間である．

図 14 に例を示す．$\Psi = \{T_1(=4), T2(=6)\}$ に対して $idle'(4) = 1, idle'(6) = 2$ であるため VS として $(1, 4)$ と $(2, 6)$ の二つの候補がある．

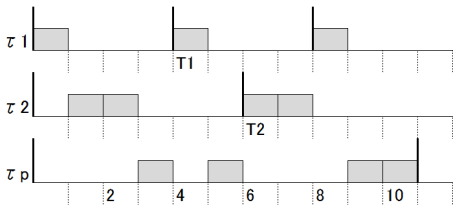

図 14　$R_p > T_{p-1}$ となる例

式 (6)　, (8)　で得られる VS を用いた場合, τ_p の最初の Job は必ずブロックされ，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は効果は無いが，ジッタ短縮に効果的な場合がある．例えば 図 14 において VS $= (1, 4)$ を選択した場合, タスクの一部が最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．

定理 3.19 (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されるタスク τ_p はデッドライン制約を満たし，*VS* 導入後の τ_p を含むタスクセット Γ' は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 $\Gamma_{high} = \{\tau_1, \tau_2, ..., \tau_{p-1}\}$，$\tau_p$ 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 3.10 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 3.5 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．■

例 3.20 (ERD 法 - 3)： $\Gamma = \{(1, 5), (1, 6), (2, 8), (4, 14)\}$ に対し, $\tau_p = \tau_4$ とする．*RTA* による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 15)*.

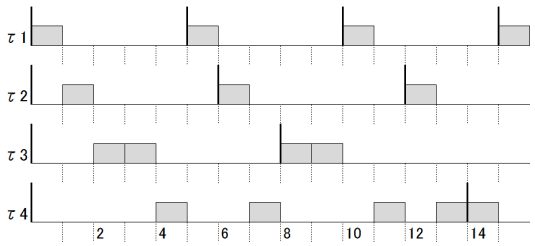

図 15　**RM** 法によるスケジューリング例

$R_4 > T_3$ であるため，式 (6)　, (8)　により *VS* を求め

る．$\Psi = \{T_1, T_2, T_3\}$ から，それぞれの周期について $idle'$ を求めることになる．式 (9) より $idle'(T_1) = 1$ となり，$VS_1 = (1, 5)$ が得られる．この場合，$R'_4 = 14$ となり，応答時間は短縮されない *(図 **16**)*．同様に $idle'(T_2) = 1$，$VS_2 = (1, 6)$ 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．

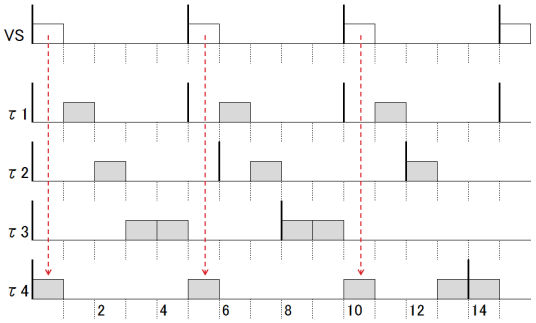

図 16　ERD 法 (VS_1) によるスケジューリング例

次に，$idle'(T_3) = 2$ から $VS_3 = (2, 8)$ が得られ，$R'_4 = 10$ となる *(図 **17**)*．VS_1, VS_2, VS_3 を比較した結果，VS_3 が最短応答時間となるため，$VS = VS_3$ とする．

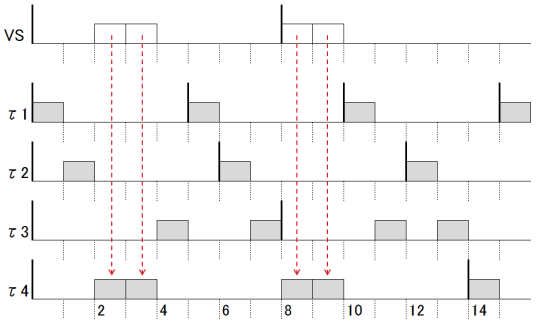

図 17　ERD 法 (VS_3) によるスケジューリング例

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．

3.3　Slack Collection (SC) 法

実際のシステムでは，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 3.21 (SC 法):　図 ***18*** に *RM* 法の，図 ***19*** 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．τ_1 と τ_3 は常に最悪実行時間の半分で終了する．*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．

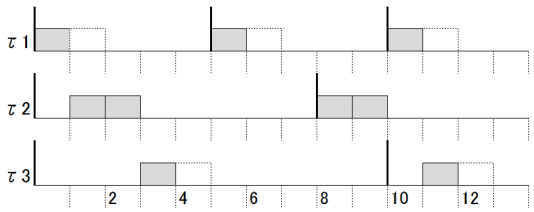

図 18　RM 法によるスケジューリング例

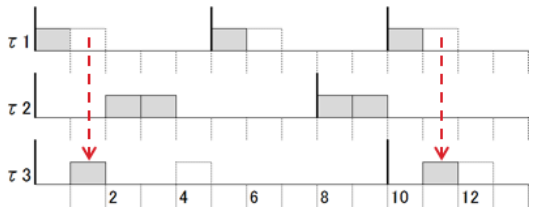

図 19　SC 法によるスケジューリング例

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 [v] 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 $\tau_1, \tau_2, ..., \tau_n$ [vi] 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 **ERD** 法の評価結果を示す．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とした場合の平均応答時間を，RM 法を 1 とした相対値として示している．また，タスク当たりのプロセッサ最大使用率が 25% と 50%，二通りの評価を行った．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 より，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 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，若干劣る結果となった．

表の平均は，対象タスクセットの数を重みとして掛けている．例えば τ_3 を短縮可能なタスクセットに比べ，τ_7 を短

[v] 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．
[vi] 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．

表 1　評価結果 **ERD** 法

	使用率 $\leq 25\%$			使用率 $\leq 50\%$		
τ_p	**RM**	**DM**	**ERD**	**RM**	**DM**	**ERD**
τ_3	1.000	0.676	0.663	1.000	0.642	0.571
τ_4	1.000	0.593	0.581	1.000	0.521	0.459
τ_5	1.000	0.472	0.471	1.000	0.394	0.325
τ_6	1.000	0.412	0.406	1.000	0.248	0.250
τ_7	1.000	0.365	0.355	1.000	0.130	0.130
平均	1.000	0.523	0.515	1.000	0.421	0.376

縮可能なタスクセット (つまり τ_7 の存在するタスクセット) は数が少ないためである．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．

4.3　評価結果 - ERD + SC 法

表 2 に **ERD** 法に **SC** 法を追加した場合の平均応答時間を示す． 表の見方は 表 1 と同じである．この評価条件では, タスクは平均して 78% の時間で早期終了する．

表 2　評価結果 **ERD + SC** 法 (早期終了=78%)

	使用率 $\leq 25\%$			使用率 $\leq 50\%$		
τ_p	**RM**	**DM**	**ERD+SC**	**RM**	**DM**	**ERD+SC**
τ_3	1.000	0.715	0.705	1.000	0.700	0.651
τ_4	1.000	0.708	0.701	1.000	0.576	0.526
τ_5	1.000	0.542	0.542	1.000	0.513	0.452
τ_6	1.000	0.491	0.485	1.000	0.297	0.298
τ_7	1.000	0.440	0.438	1.000	0.137	0.149
平均	1.000	0.599	0.593	1.000	0.484	0.449

同様にタスクが平均して 56% の時間で早期終了する場合の結果を**表 3** に示す．

表 3　評価結果 **ERD + SC** 法 (早期終了=56%)

	使用率 $\leq 25\%$			使用率 $\leq 50\%$		
τ_p	**RM**	**DM**	**ERD+SC**	**RM**	**DM**	**ERD+SC**
τ_3	1.000	0.755	0.747	1.000	0.700	0.651
τ_4	1.000	0.708	0.700	1.000	0.618	0.567
τ_5	1.000	0.683	0.735	1.000	0.557	0.489
τ_6	1.000	0.553	0.548	1.000	0.347	0.347
τ_7	1.000	0.496	0.490	1.000	0.188	0.214
平均	1.000	0.658	0.664	1.000	0.519	0.484

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 50% の場合は, 短縮度合が高くなる．

4.4　考察

以上の結果から，各タスクの使用率が高くなると **ERD+SC** 法が平均して有利となる．**DM** 法では相対デッドラインを短くすることが困難な場合でも，**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** 法が劣る結果も確認された．**ERD** 法では τ_p の周期 T_p よりも短い周期の VS 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は効果を得ることが困難となる．この傾向は τ_p の優先度が低く, かつ周期が比較的長い場合に見られた．

スラックの利用に関しては, **SC** 法と **DM** 法で性能差にばらつきは無かった．評価に用いたタスクセットの多くに対して，**DM** 法により比較的容易に対象タスクの優先度を上げることが可能であり， スラックの発生とともに結果的に対象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．

5.　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手法として **ERD** 法と **SC** 法を提案した．また, タスク優先度の変更と, タスクセットに新たなタスクを追加するための定理・補題を示した．**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．

謝辞

本研究の一部は JSPS 科研費 15K00073 の助成を受けて行われた．

参考文献

[1] C. L. Liu and J. 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] J. P. Lehoczky and S. 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] B. Sprunt, J. Lehoczky and L. Sha, *Exploiting unused periodic time for aperiodic service using the extended priority exchange algorithm. Proceedings IEEE Real-Time Systems Symposium*, 1988.

[6] E. Bini, G. C. Buttazzo, and G. M. 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.

[7] J. P. Lehoczky, L. Sha, and J. K. Strosnider, *Enhanced aperiodic responsiveness in hard real-time environments. In Proceedings of the IEEE Real-Time Systems Symposium*, December 1987.

[8] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, *Hard real-time scheduling: The deadline monotonic approach. In IEEE Workshop on Real-Time Operating Systems*, 1992.

[9] R. Davis, A. Wellings, *Dual Priority Scheduling. In Proceedings of the 16th IEEE RTSS*, 1995.

[10] De Niz, D., Lakshmanan, K., and Rajkumar, R. R., *On the scheduling of mixed-criticality realtime task sets. In Proceedings of the Real-Time Systems Symposium*, 2009.

[11] Joseph, M. Pandya, P., *Finding response times in a real-time system. BCS Computer Journal*, 1986.

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