LO-001

非割込み型EDFスケジューリングにおけるパフォーマンス予測：理論と実装
Theory and Experimental Results of Non-Preemptive EDF Scheduling Performance

花田真樹 †
Masaki HANADA
中里秀則 †
Hidenori NAKAZATO

1 まえがき

近年，計算機の高性能化やネットワークの高速化に伴い，リアルタイムシステムの適用分野が，ロボット制御やプラント制御などの制御系システムのみでなく，VODやテレビ会議システムなどのマルチメディア情報を扱う情報系システムへと広がっている．一般的に，これらの情報系システムはソフトリアルタイムに分類され，デッドライン(実行終了期限)ミスが起きても，システムは稼動し続ける．さらに，ユーザ要求のタイミングを事前に把握することが困難なために非決定的な環境下で動作することが多い．そのため，絶対的な保証を行うのは困難であり，統計的な保証が必要となる．

リアルタイム性を確保するためには，どのタスクをいつ実行するかを決定するスケジューリング手法が重要な要素である．これまでに，タスクのデッドラインを守るために，Earliest Deadline First(EDF)[1, 2, 3]などのリアルタイムスケジューリングが提案されている．EDFスケジューリングは，デッドラインの早い順に優先順位を付ける手法であり，事前に起動時間がわからない非周期タスクに適応可能である．

我々は，既にタスクがランダムに到着する仮定下で，非割込み型EDFスケジューリングを用いた場合のシステム性能を待ち行列理論を用いて解析している[3]．しかしながら，我々の提案している解析ではタイマの精度，割込み処理やコンテキストスイッチのオーバーヘッドなどを考慮しておらず，システム性能に関して，実システムの実測データとは一致しない可能性がある．そこで，本論文では，非割込み型EDFスケジューリングを行うタスクスケジューラをリアルタイムOS上に実装し，システム性能を理論解析と実測データの両面から評価する．

現在，リアルタイムアプリケーション構築の際のプラットフォームとして，様々なリアルタイムOSが用いられている．リアルタイムOSを用いる主な利点は，マルチタスクによるリアルタイム性の確保にある．特に，本論文ではスケジューラの変更を行うため，フリーで変更が比較的容易なタイムシェアリング系OSをリアルタイム拡張したRTLinux[4, 5, 6]を用いる．

2では既に提案している解析によるシステム性能の算出式について述べる．**3**ではRTLinuxについて述べる．**4**ではタスクモデルと非割込み型EDFスケジューリングを行うスケジューラの実装について述べ，実験結果を示し，システム性能に関して理論解析と実測データの両面から評価する．**5**ではまとめと今後の課題について述べる．

2 理論解析

(1) システムモデル

システムモデルは，単一サーバモデル(図1)とする．

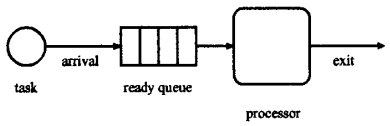

図1 単一サーバモデル

タスクの実行順序はデッドラインの早い順とし，キューは無限長とする．実行中のタスクよりも早いデッドラインをもったタスクが到着した場合でもそのまま実行を続行する非割込み型である．タスクが到着した場合には，デッドライン保証の判定は行わず，即座にキューに格納され，さらに，タスクがデッドラインを守れなかった場合でもその実行を中断することはないとする．

タスクは実行時間と相対デッドラインをもつ．タスクのデッドラインは，到着時刻と相対デッドラインを足した値として定義される．

(2) 性能解析[3]

タスクの到着は平均到着率λのポアソンとし，相対デッドラインを平均$1/\mu$の指数分布とする．

一般的に，相対デッドラインと実行時間の間には強い依存性があると考えられるので，相対デッドラインの短いタスクは，実行時間も同様に短いと仮定する．これより，各タスクの実行時間は，そのタスクの相対デッドライン時間の$1/n$(nは固定)とする(以下，nを実行時間比率と呼ぶ)．

① 待ち時間の条件付き期待値の各要素

システム性能評価指標として，相対デッドラインがxであるタスクの待ち時間の条件付き期待値$W(x)$を導出する．

$W(x)$は以下の3つの要素で構成される．ここで，相対デッドライン時間がxであるタスクを注目タスク，注目タスクの待ちに寄与する相対デッドライン時間がtであるタスクを対象タスクと呼ぶ．

i. 注目タスクの到着時に実行中の対象タスクによる待ち時間W_o

ii. 注目タスクがキューに到着した時点で，既にキューにある対象タスクの中で注目タスクより先に実行される対象タスクによる待ち時間$W1(x)$

iii. 注目タスクがキューにいる間に到着した対象タスクの中で，注目タスクより先に実行される対象タスクによる待ち時間$W2(x)$

これより，$W(x)$は以下である．

$$W(x) = W_o + W1(x) + W2(x) \quad (1)$$

② 待ち時間の条件付き期待値

待ち時間の条件付き期待値の各要素を求め，条件付き分布関数$W(t|X=x)$を用いると式(1)は以下の

† 早稲田大学大学院 国際情報通信研究科
〒 367-0032 本庄市大字西富田字大久保山 1101-3
GITS,Waseda University
1101-3 Ohkuboyama,Nishitomida,Honjo-shi,
Saitama 367-0032,Japan

ようになる．

$$
\begin{aligned}
W(x) &= \int_0^\infty (1 - W(z|X=x))\,dz \\
&= \int_0^\infty \frac{\rho(t)t}{2n}dt \\
&+ \int_0^x \frac{t}{n}\lambda(t)\left(\int_0^\infty 1 - W(z|X=t)dz\right)dt \\
&+ \int_x^\infty \frac{t}{n}\lambda(t)\left(\int_{t-x}^\infty 1 - W(z|X=t)dz\right)dt \\
&+ \int_0^x \frac{t}{n}\lambda(t)\left(\int_0^{x-t} 1 - W(z|X=x)dz\right)dt.
\end{aligned} \tag{2}
$$

また，$W(t|X=x)$ は依存係数 $K(x)$ を用いることにより，近似的に以下の式で表せる．

$$
W(t|X=x) \quad = \quad 1 - \frac{\lambda}{n\mu}e^{-(n\mu-\lambda)K(x)t}. \tag{3}
$$

式 (3) を式 (2) に代入することにより，$K(x)$ が求まる．

また，$W(x)$ は式 (3) を用いると以下の式となる．

$$
W(x) \quad = \quad \int_0^\infty \frac{\lambda}{n\mu}e^{-(n\mu-\lambda)K(x)z}dz. \tag{4}
$$

式 (2)，式 (3) より求めた $K(x)$ を式 (4) に代入することにより $W(x)$ が求まる．

3 RTLinux

(1) 概要

RTLinux は，Linux を修正してリアルタイム処理をできるようにした OS である．Linux ではタイムシェアリングを用いており，タスクが CPU への割り当てられた時間枠を超えるとカーネルがそれを停止し，他のタスクに割り当てるので，リアルタイム性の必要なタスク (以下，リアルタイムタスクと呼ぶ) も他のタスクと平等に扱われることになる．そこで，RTLinux では，新しいレイヤ (リアルタイムカーネル) をハードウェアと Linux カーネルの間に導入し，リアルタイムカーネルの RT スケジューラにより，Linux カーネルを優先順位の低い一つのタスクとして実行し，リアルタイムタスクが CPU を必要としたときは，横取りするように設計されている．図 2 に RTLinux の構造を示す．

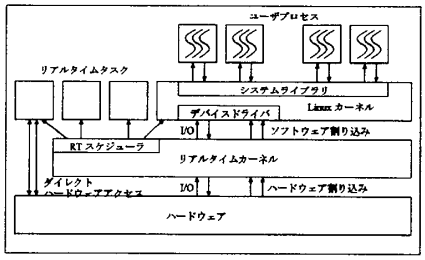

図 2　RTLinux カーネルの構造

(2) スケジューリング方式

RTLinux では，割込み型の固定優先度スケジューリング方式を提供している．同一優先順位内での方式として，先入れ先出しを行うスケジューリング方針とシステムごとのタイムスライス (割り当てられた時間の長さ) を使用するラウンドロビン スケジューリングがある．

(3) 割込み処理

RTLinux では，特定の割込みに対してのみ，適切な割り込みサービスルーチンが動作する．それ以外の割込みは全て一旦保留され，RTLinux カーネルがアイドル状態になり，Linux カーネルが稼働したときにソフトウェア割り込みとしてカーネルにわたる仕組みとなっている．ハードウェア割り込みに関しては，最悪のケースで割り込み発生から 15 マイクロ秒の遅れで実行され，周期的なプロセスに対してはスケジューリングされた時刻から 25 マイクロ秒以内の遅れで実行される仕様となっている [7]．

4 実装と実験

この章では，まず，実システムにおける割込みハンドラとタスクについて述べ，次に，タスクモデルとスケジューラの実装について述べる．次に，実験結果を示し，理論解析と実測データの両面より評価する．

(1) 実システムにおける割込みハンドラとタスク

リアルタイムシステムの実行単位は，タスクとハンドラに分類される．一般的に，CPU は通常の実行状態と割込み/例外処理実行状態をもっており，タスクは通常の実行状態に対応し，ハンドラは割込み/例外処理実行状態に対応する動作単位である．実システムでは，割込みハンドラは，最小限の割込み応答処理と，イベントを検出し，タスクへ通知する処理を行うのが一般的である (図 3)．

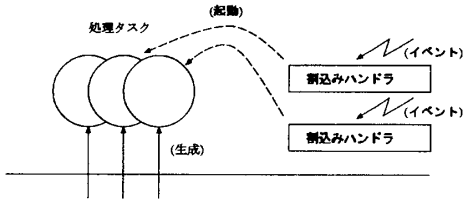

図 3　実システムにおける割込みハンドラとタスク

(2) 実装

① タスクモデル

上記で述べたように，実システムでは一般的に割込みハンドラがイベントを検出し，対応するタスクを起動するという実装が多く行われる．割込みハンドラはある特定のタスクと対応している場合が多い．本実験では，タスクのポアソン到着を擬似するために，割込みハンドラを用いず，処理を振り分けるタスク (以下，振り分けタスクと呼ぶ) から，処理を行うタスク (以下，処理タスクと呼ぶ) を起動する (図 4)．振り分けタスクはタイマ割込みにより起動する．また，各タスクは，リアルタイム性が要求されるため，リアルタイムカーネル上のリアルタイムタスクとして扱う (図 2)．

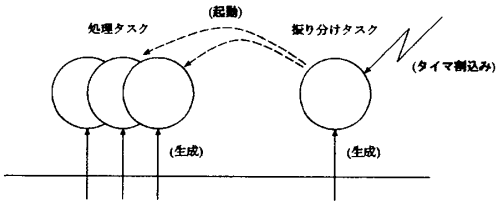

図 4　タスクモデル

振り分けタスクでは，必要な前処理 (全タスクに共通な処理など) が行われ，デッドラインなどの必要なパ

ラメータを付加して，対応するタスクを起動し，処理を依頼する．

② スケジューラ
リアルタイムカーネルの RT スケジューラを，既に実装されている割込み型の固定優先度スケジューリング方式から固定優先度非割込み型 EDF スケジューリング方式に変更する．タスクは固定の優先度とデッドラインをもっている．固定優先度非割込み型 EDF スケジューリング方式では，まず，優先度の高いタスクが先に実行される．さらに，タスクが同優先度を持つ場合は，タスクのデッドラインの早い順に実行される．
振り分けタスクは他の処理タスクへの通知を行う必要があるので，タイマ割込みがあると他の処理タスクが実行中であっても，処理タスクへ割込みを行えるものとする．処理タスクは，振り分けタスクからのみ割込みを許可し，他の処理タスクからの割込みを不可とする．図 5 は，処理タスク A が振り分けタスクに割り込まれ，処理タスク A が処理タスク C に割り込まれない状況を示している．

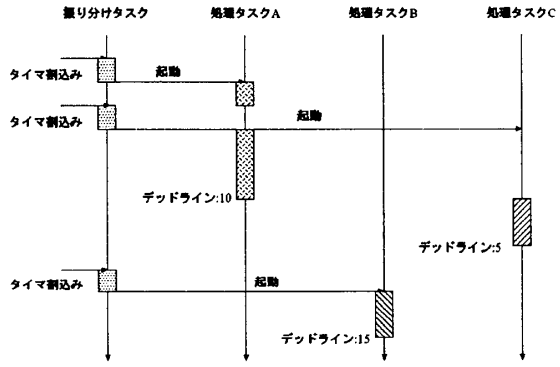

図 5　タスク実行の例

(3) 実験

実験では，Linux カーネル (version2.4.4) と RTLinux(version3.1pre3) を使用する．また，前提として，**2** の条件と同様に，タスクの到着は平均 λ のポアソン，相対デッドラインは平均 $1/\mu$ の指数分布とする．各タスクの実行時間は，簡易的に，そのタスクの相対デッドラインと等しいとする．

実験における振り分けタスクと処理タスクの処理シーケンスの概要を図 6 に示す．

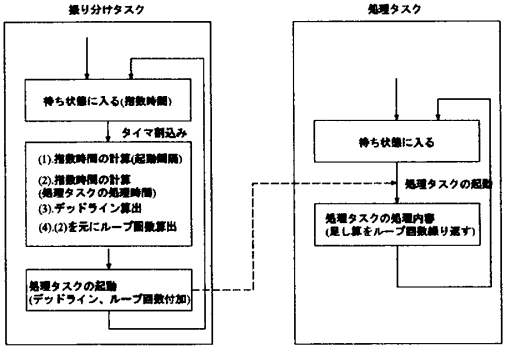

図 6　実験における処理シーケンス

- 処理タスク
  処理内容は，簡易的に足し算 (1+1) を繰り返す処理とする．
- 振り分けタスク
  タイマ割込みにより起動する．処理タスクの指数時間の処理を実現するために，処理時間に対応する繰り返し回数を求め，デッドラインと共に処理タスクに通知する．

RTLinux 上で足し算の繰り返し回数に対する計測した処理時間 (ナノ秒) を表 1 に示す．

表 1　ループ回数に対する処理時間

回数	100	1000	10000	100000	1000000	10000000
処理	337	3376	33498	333371	3334283	33341903

ループ回数を l，処理時間を s とすると，1 次式への近似は以下となる．

$$3.33 \times l + 41.19 = s$$

これより，指数分布の乱数 (処理時間 s) から繰り返し回数 l を算出する．

(4) 実験結果

パラメータと対応する実測データを表 2 に示す．$1/\lambda$ と $1/\mu$ に関しては，単位はマイクロ秒である．図中における実線は **2** で述べた我々の提案している解析結果であり，プロットは実験により得られた実測データである．また，横軸は相対デッドライン (処理時間) であり，縦軸は実行可能キューにおける待ち時間である．図中の単位はナノ秒である．

表 2　パラメータ

$1/\lambda$	1000	1000	1000	10000	10000	10000
$1/\mu$	200	500	800	2000	5000	8000
W(x)	図 7	図 8	図 9	図 10	図 11	図 12

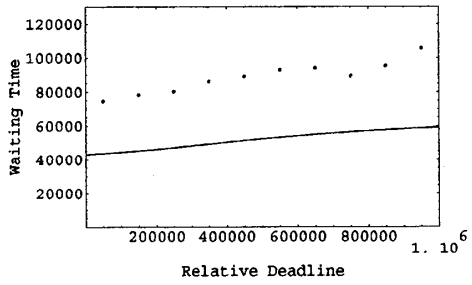

図 7　平均待ち時間 W(x)($1/\mu = 200$)

(5) 評価

図 10，図 11，図 12 においては，実測データと解析結果がほぼ一致しているが，図 7，図 8，図 9 においては，大きな誤差が表れている．また，高負荷 (図 12 と図 9) の時と比べて，低負荷 (図 10 と図 7) の方が誤差が大きい．これらは，パラメータに対して，コンテキストスイッチにかかる時間が大きいためである．図 13 に，コンテキストスイッチを含めた各処理にかかる時間を示す．

振り分けタスクには，2 回の指数時間算出，デッドラインとループ回数算出の処理時間 (図 13 の①) が含まれる．さらに，それに加えて，タスク起動 (図 13 の②) とコンテキストスイッチ (図 13 の③) の時間が必要となる．

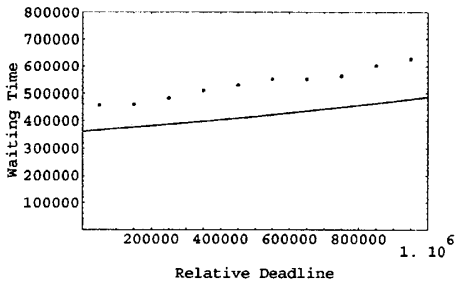

図 8　平均待ち時間 W(x)($1/\mu = 500$)

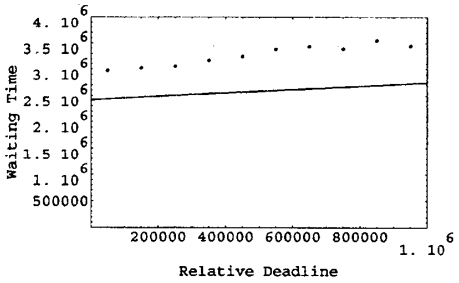

図 9　平均待ち時間 W(x)($1/\mu = 800$)

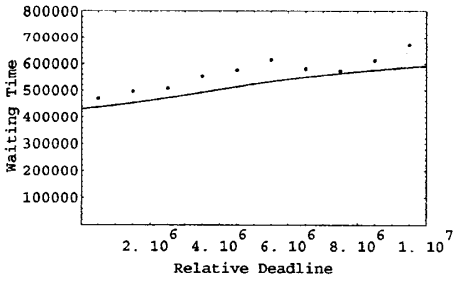

図 10　平均待ち時間 W(x)($1/\mu = 2000$)

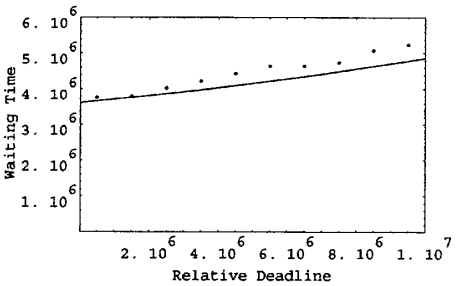

図 11　平均待ち時間 W(x)($1/\mu = 5000$)

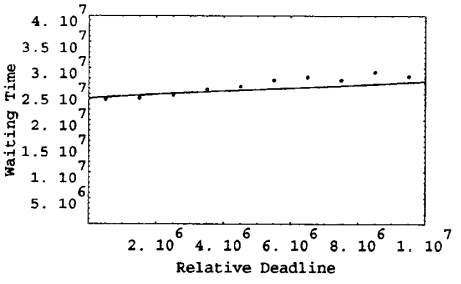

図 12　平均待ち時間 W(x)($1/\mu = 8000$)

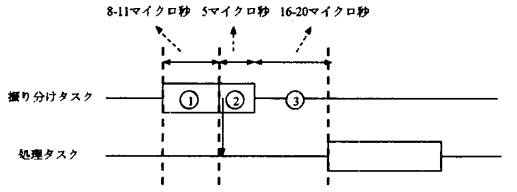

図 13　各処理にかかる時間

これより，パラメータに対して，コンテキストスイッチにかかる時間が大きい場合は，我々の提案している解析法の適用は困難である．しかし，パラメータがコンテキストスイッチにかかる時間に比べて，十分に小さい場合は，我々の提案している解析法は有効である．

5　おわりに

本論文では，非割込み型 EDF スケジューリングを行うスケジューラをリアルタイム OS 上に実装し，システム性能を実測データと理論的解析の両面から評価した．パラメータに対して，タスク起動やコンテキストスイッチにかかる時間が大きい場合，解析結果とは一致しない．しかし，パラメータがタスク起動やコンテキストスイッチにかかる時間に比べて，十分に小さい場合は，我々の提案している解析法は有効であることが分かった．

解析の課題としては，コンテキストスイッチは遊休期間と考えられるので，遊休期間を含めた解析が必要である．

今後の課題としては，本論文では待ち時間に注目したが，待ち時間に処理時間を加えた応答時間では，振り分けタスクが処理タスクに割り込む可能性が高いので，割込み回数を含めた算出が必要である．また，本論文では実験のために擬似的にポアソン到着，指数分布の処理時間を生成している．システムによっては到着や処理の分布が異なる可能性があるので，その場合の解析や実験も必要である．さらに，本論文ではリアルタイム OS を用いたが，予測が困難な一般的な OS での実験も必要であると考える．

参考文献

[1] J.R. Jackson, "Scheduling a production line to minimize maximum tardiness," Research Report 43, Management Science Research Project, University of California, Los Angeles, 1955.

[2] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," J. Assoc. Comput. Mach., Vol.20, No.1, pp.41-46, Jan 1973.

[3] 花田 真樹，中里 秀則，"非割込み型 EDF スケジューリングの近似解析，"電子情報通信学会論文誌, VOL.J87-A No.12, pp.1518-1527, Dec 2004.

[4] FSMLabs, http://www.fsmlabs.co.jp/.

[5] Matt Sherer, FSMLabs Technical Staff, RTLinux テキストブック, CQ 出版社, 2003 年 4 月.

[6] 船木陸議，"RT-Linux の構造と実装の詳細，"CQ 出版社インターフェース増刊 Vol.5 第 7 章, 2000 年 7 月.

[7] 石井和男，"リアルタイム Linux「RTLinux」の導入とロボットへの適用，"日本ロボット学会第 61 回講習会ロボット工学セミナーテキスト，pp.1-23，2000.