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

2004－ＳＬＤＭ－114　（９）
2004／3／19

実時間オペレーティングシステムRT-Frontierにおけるインプリサイス計算支援機構

小林　秀典†　　山崎　信行†

† 慶應義塾大学大学院理工学研究科
〒223-8522 横浜市港北区日吉 3-14-1
E-mail: †{kobahide,yamasaki}@ny.ics.keio.ac.jp

あらまし　実時間オペレーティングシステム *RT-Frontier* におけるインプリサイス計算支援機構は，独自の拡張インプリサイス計算モデルおよびそのスケジューリングアルゴリズムの二つから構成される．拡張インプリサイス計算モデルと従来のインプリサイス計算モデルの大きな違いは，付加部分の中断を容易にするために終端部分という第二の必須部分を有することである．この拡張インプリサイス計算モデルに基づくタスクの時間制約を満たす方式として，実時間スケジューリングアルゴリズム *Slack Stealer for Optional Parts*（SS-OP）の開発を行なった．さらに，本機構をコストおよびパフォーマンスの観点から評価した．

キーワード　インプリサイス計算モデル，実時間オペレーティングシステム，スケジューリングアルゴリズム，スラックスティーリング方式，組み込みシステム

A Mechanism for Supporting Imprecise Computation in the RT-Frontier Real-Time Operating System

Hidenori KOBAYASHI† and Nobuyuki YAMASAKI†

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

Abstract　This paper describes a mechanism developed in the *RT-Frontier* real-time operating system for supporting imprecise computation. It is composed of an extended imprecise computation model and an algorithm for scheduling tasks based on this model. The extended imprecise computation model substantially differs from the traditional model by having a second mandatory part, called the wind-up part. The wind-up part is used to terminate the optional part in less demanding manners. The scheduling algorithm that handles the timing constraints of the wind-up part is called the *Slack Stealer for Optional Parts* (SS-OP) algorithm. The experimental results in terms of cost and performance are shown.

Key words　Imprecise Computation Model, Real-Time Operating Systems, Scheduling Algorithms, Slack-Stealing, Embedded Systems

1. はじめに

実時間システムにおいては，タスクの挙動を正確に把握する事が重要である．しかし，実時間タスクの実行時間を予測する事は必ずしも容易ではない．これは，実世界における事象を扱う実時間タスクの実行時間には，環境からの入力に大きく依存するものがあるためである．この様なタスクの実行時間を予測する事は，動作環境が完全に予測できない場合には非常に困難である．さらに，近年の実時間システムは複雑なハードウエアやCommercial-Off-The-Shelfコンポーネント等の時間予測性を低下させる原因を多く含んでいる．これにより，最悪実行時間の概算値は悲観的なものになる傾向にある．一般に，変動の大きい負荷に対応可能な実時間システムを構築するためには，必要最大量の資源を予約し，想定される最悪の事態に備えなければいけない．しかし，この様なシステムでも，実際には確保した資源の一部のみが利用されるため，多くの資源は無駄に確保されていたことになる．

一方，実行時間が正確には把握出来ないタスクならびに実行

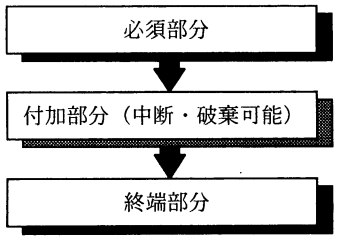


図 1　RT-Frontier における拡張インプリサイス計算モデル

時間に大きな変動のあるタスクを効率的に扱うための手法の一つとして，インプリサイス計算モデル [1] がある．インプリサイス計算モデルに基づくタスクはインプリサイスタスクと呼ばれ，必須部分及び付加部分に分割される．必須部分は最低限の精度の結果を生成するのに必要な処理に該当し，付加部分は結果の品質を向上させるのに必要な処理に該当する．

インプリサイスタスクを実行する場合には，必須部分を先に終了させる事が重要である．必須部分が終了しているタスクは，全体を終了するだけの資源が利用可能でない場合でも，付加部分を切り捨てる事でデッドラインを守る事が可能である．またこの場合，少なくとも必須部分は正常に終了しているので，精度が低くても結果の論理的な正しさは保証されている．

本稿では組み込みシステムを対象とし，この様な特徴を有するインプリサイス計算モデルを支援する実時間オペレーティングシステム RT-Frontier [2] について述べる．組み込みシステムでは動作時に負荷を調整する事が出来ない．そこで，インプリサイス計算を低コストで実現する事で，過剰な資源予約を排除し，過負荷に対する柔軟性を備えたシステムの構築を目指す．

2.　拡張インプリサイス計算モデル

従来より用いられてきたインプリサイス計算モデルには，実装に関する制約が多い．特に，付加部分を即座に中断する事が可能であると仮定している事は，システム設計者及びアプリケーション設計者の大きな障害になり得る．それに対し，この付加部分の中断方法に着目し，その問題点を改善したのが RT-Frontier における拡張インプリサイス計算モデルである．このモデルは，図 1 に示すように，従来のインプリサイス計算モデルが有する必須部分及び付加部分に加えて，終端部分を有する．この終端部分は付加部分の後にのみ実行可能になるが，必須部分と同じ属性を有するので，付加部分の中断を許容するために必要な処理を含める事が可能である．

従来の終端部分の無いモデルを用いた場合，オペレーティングシステム等の下位レイヤのソフトウエアがインプリサイス計算の途中終了を補助する必要がある．オペレーティングシステムがインプリサイス計算の途中終了を支援する機構としては，チェックポイントを設ける方法 [3], [4] が一般的である．チェックポイント機構は中間結果を容易に保持可能であるが，いつ結果の精度が向上するか分からない場合には，チェックポイントの間隔を小さくしなければならないのでシステムオーバヘッドが大きくなる．また，最終チェックポイント後に生成された結果は付加部分を中断する際に失われてしまうので，最終的に利用可能な実行結果が最良の結果でないこともある．さらに，アプリケーションは下位レイヤのソフトウエアを前提として開発されなければならないので，移植性が低下する．

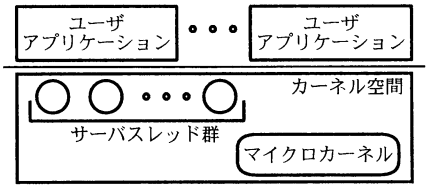


図 2　RT-Frontier の概観

これに対し，本モデルに従う場合，中断されたタスク自身が最終的な結果をユーザや他のタスクから利用可能にするための処理を行う．したがって，プログラマは付加部分を中断した後の動作を明示的かつ詳細に指定可能であり，確実に最良の計算結果を得る事が出来る．また，タスクが自ら中間結果を保持するようになるため，チェックポイント機構のような付加的な補助機構は必要無くなる．しかしその反面，終端部分を中断あるいは破棄する事は出来ないため，スケジューリングアルゴリズムの複雑化を招くという欠点がある．

3.　スケジューリング機構

実時間オペレーティングシステム RT-Frontier の概観を図 2 に示す．ユーザアプリケーションに対するサービスは，マイクロカーネルとカーネルレベルスレッドにより提供する．スケジューラはマイクロカーネル内に存在し，独自に開発したスケジューリングアルゴリズム Slack Stealer for Optional Parts (SS-OP) を実現する．

3.1　負荷に関する仮定

実時間性を要求する多くの処理は周期的な時間特性を有する．そこで，本稿では周期タスクにより構成される負荷を対象とする．以下では，周期タスクに属するジョブを周期ジョブと呼び，$J_i(r_i, d_i, m_i, w_i)$ で表すことにする．ここで，r_i はリリースタイム，d_i は絶対デッドライン，m_i は必須部分の最悪実行時間，w_i は終端部分の最悪実行時間である．ただし，w_i はゼロであっても構わない．また，ジョブ J_i の属するタスクの周期を T_i とし，相対デッドラインと等しいとする．

これらの属性より，ジョブ J_i の属するタスクの利用率 u_i を

$$u_i = \frac{m_i + w_i}{T_i} \tag{1}$$

と定義する．これにより，n のタスクが存在するシステムの利用率 U_e は

$$U_e = \sum_{i=1}^{n} u_i \tag{2}$$

と表す事が出来る．以降ではこの U_e を必須利用率を呼び，$U_e < 1$ を仮定する．また，U_o を $1 - U_e$ と定義する．

上記に加えて，付加部分の実行時間およびインプリサイス計算の誤差関数は与えられていないと仮定する．ここで，誤差関

数とは付加部分に割り当てられる実行時間とその結果の精度を表すものである．誤差関数は既存のインプリサイス計算をスケジュールするためのアルゴリズム [5], [6] を開発する際には重要な指標であるとされてきた．しかし，付加部分の実行時間が一定でない場合には，既存の誤差関数を用いた最適化手法は最適なスケジュールを生成することが出来ない．

例えば，実際の付加部分の実行時間が交互に 3 と 7 になるタスクを考える．この付加部分の実行時間を一定であると仮定し最適化アルゴリズムを用いた場合，周期的なインプリサイスタスクに対する最適スケジュールでは各タスクの付加部分に割り当てる実行時間がサイクル間で同一となる [6]．したがって，仮に二周期毎に合計 4 の時間が利用可能であった場合，その実行された割合は平均で $(2/3 + 2/7)/2 = 10/21$ となる．しかし，単純に付加部分の実行時に利用可能なだけの時間を割り当てていった場合，実行される割合は平均で $(3/3 + 1/7)/2 = 12/21$ となる場合が存在する．ゆえに，付加部分の実行時間が一定でない場合には，誤差関数が与えられているとする必要はない．

3.2 SS-OP アルゴリズム

実時間スケジューリングアルゴリズム SS-OP は，プロセッサ時間を有効に利用することを目的とし，デッドラインを優先度として用いたグリーディスケジューリングを行なう．

拡張インプリサイス計算をスケジュールする際に最も重要な事は，必須部分と終端部分の時間制約を満たすという事である．そこで，SS-OP では必須利用率を基にした資源予約を行ない，これを事前に保証する．一方，式 (1) 及び (2) に示したように，必須利用率は付加部分の実行時間を含まない．したがって，付加部分を実行可能であるのは，他に実行する部分が存在しない場合，あるいはいずれの時間制約も破らずにタスクが有するスラックを消費可能な場合に限定される．ただし，前者のアプローチは終端部分と付加部分の順序関係のために適当でない．それに対し，後者のアプローチには，周期タスクの数を n，ハイパーピリオド内のジョブの到着回数を N とすると，スラックを正確に計算するための計算量が最大で $O(nN)$ となってしまう [7] という欠点がある．

そこで，SS-OP では，利用率を基にシステムに存在するスラックを少ない方に概算する．この概算は全ジョブのリリース時に行なうが，実際にスラックを消費するのは付加部分を実行するときのみである．またスラックを割り当てた後に利用可能なスラックに変化した場合には，スケジューラがその情報を更新し，必要であればジョブ間でスラックの再配分を行なう．この管理手順を表 1 に定義した記号を用いて図 3 に表す．

まず，事象 1 において，各ジョブに割り当て可能なスラック (S_i) を二つの仮想的なプロセッサを考える事により計算する．ただし，実際のプロセッサの速度を 1 であるとし，これらの仮想プロセッサの速度を U_e と U_o であるとする．このうち速度が U_e のプロセッサは必須部分と終端部分を実行するものであり，速度が U_o のプロセッサは付加部分を実行するものである．

仮想プロセッサ上でのスケジュールは Earliest Deadline First (EDF) アルゴリズムに基づいて構築する．その際，各部分のリリースタイム及びデッドラインは元のジョブのものと等しい

表 1　図 3 における記号の定義

記号	意味
R_i	ジョブ J_i が消費可能な実行時間
S_i	ジョブ J_i が保持するスラックの量
E_i	ジョブ J_i が必須部分を終了するまでは S_i と同義，それ以降は R_i と同義
$J_{p(i)}$	d_i 以前のデッドラインを持つジョブ J_k ($k \neq i$) のうち最も遅いデッドラインを持つジョブ
$J_{n(i)}$	d_i 以降のデッドラインを持つジョブ J_k ($k \neq i$) のうち最も早いデッドラインを持つジョブ

以下のいずれの事象も生じない間は，実行可能状態にあるジョブの内，最もデッドラインの早いジョブ J_i を実行し，その際に消費した時間と同量の時間を R_i から減少させる．

事象 1: ジョブ J_i が実行可能になった場合，$R_i = m_i$ とする．次に，S_i を式 (3) より求め，ジョブ $J_{n(i)}$ が存在する場合には $E_{n(i)} = E_{n(i)} - S_i$ とする．

事象 2: ジョブ J_i が必須部分を終了した場合，$R_i = R_i + S_i$ とする．

事象 3: ジョブ J_i が付加部分を終了した場合，あるい $R_i = 0$ で付加部分を中断した場合，$R_i = R_i + w_i$ とする．

事象 4: ジョブ J_i が終端部分を終了した場合，ジョブ $J_{n(i)}$ が存在するならば，$E_{n(i)} = E_{n(i)} + R_i$ とする．

図 3　SS-OP アルゴリズム

表 2　図 4 のジョブパラメータ

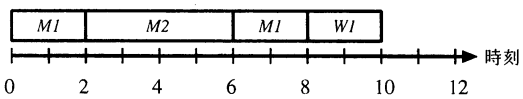
ジョブ	r_i	d_i	m_i	w_i
J_1	0	12	2	1
J_2	2	10	2	0

と仮定する．また，必須部分及び終端部分の実行時間は実際の最悪実行時間の $1/U_e$ 倍であるとする．一方，付加部分の最悪実行時間に関する情報は全く与えられていないので，EDF により割り当て可能な時間の最大値と常に等しいとする．

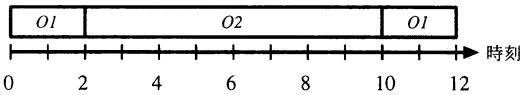
上記の条件で表 2 に示す二つのジョブのスケジュールを作成した場合の例を図 4 に示す．ただし，図中ではジョブ J_i の必須部分における実行を M_i，付加部分における実行を O_i，終端部分における実行を W_i として表した．

ここで注目すべきは，図 4(b) に示したスケジュールにおいて各付加部分に割り当てられた実行時間の量である．仮想プロセッサ上のスケジュールは EDF に基づいて構築されるために，仮想プロセッサ上で付加部分の利用率の総和が 1 を超える事はない．したがって，この速度 U_o の仮想プロセッサ上で割り当て可能な時間を基に，その U_o 倍の時間を実プロセッサで割り当てれば，付加部分のプロセッサ要求量は実プロセッサ上の任意の区間 $[t_1, t_2)$ において $U_o(t_2 - t_1)$ を超える事が無い．

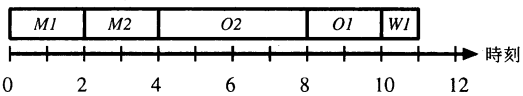
同様に図 4(a) においては，必須部分及び終端部分の利用率の総和が 1 を超えることはない．これは，仮定から実プロセッサにおけるこれらの部分のプロセッサ要求量が任意の区間 $[t_1, t_2)$ で $U_e(t_2 - t_1)$ を超える事は無いためである．そのため，仮想プロセッサ上では EDF により安全にスケジュール可能である．逆に，実プロセッサ上では，これらの U_e 倍の実行時間を安全に割り当てる事が出来る．ただし，必須部分及び終端部分の実



(a) 速度 U_e の仮想プロセッサにおける EDF スケジュール



(b) 速度 U_o の仮想プロセッサにおける EDF スケジュール



(c) 速度 1 の実プロセッサにおける SS-OP スケジュール

図 4　スケジュール例

行時間はあらかじめ必須利用率を用いて予約されているので，SS-OP スケジュールを構築する観点からは重要ではない．

以上より，これらの二つの仮想プロセッサ上のプロセッサ要求量を基にし，プロセッサを割り当てれば，実プロセッサ上でもシステムの利用率が 1 を超える事が無い．したがって，付加部分を他のジョブの必須部分や終端部分よりも先に実行してもデッドラインミスを起こすことは無い．またこの際，最終的な SS-OP スケジュールにおける実行順と仮想的なプロセッサ上における実行順は，部分単位で異なっていても良い．これは，要求プロセッサの定義から，リリースタイム以降でデッドラインよりも前であれば，どの時点でジョブを実行してもプロセッサ要求量が変化しないためである．

次に，あるジョブが到着する前に，より遅いデッドラインを持つジョブがスラックを消費してしまうことにより，実プロセッサ上でのスラックの割り当て量が速度 U_o の仮想プロセッサ上のものに従えない場合を考える．例えば図 4(c) において，仮にジョブ J_2 が到着する前にジョブ J_1 が 2 のスラックを消費してしまっていた場合，ジョブ J_2 は 4 のスラックを利用することが出来ない．しかし，速度 U_o の仮想プロセッサ上で，付加部分の実行時間は任意に変更可能である．ゆえに，図 4(b) の EDF スケジュールにおいて，実プロセッサ上で消費されたスラック量を基にジョブ J_2 に割り当てるスラックを 6 に変更することで，再度スケジュール可能になる．

上記を定式化すると，実際にジョブ J_i に対して割り当て可能なスラック量は式 (3) の様になる．

$$S_i = U_o \times F(d_i, \max\{d_{p(i)}, t_E, r_i\}) \tag{3}$$

ただし，

$$F(x,y) = \begin{cases} 0 & x \leqq y \text{ の場合} \\ x - y & \text{それ以外} \end{cases} \tag{4}$$

であり，t_E は最もデッドラインの早いジョブがスラックを消費することの出来る最も早い時刻を表す．

時刻 t_E を算出するためには，初期値がゼロの変数を必要な場合に更新していく方法が容易である．SS-OP では，最もデッドラインの早いジョブ J_E がプリエンプトされた場合及び付加部分を終了した場合に，t_E を式 (5) の様に更新する．

$$t_E = \max\{d_E, t_E\} - \frac{R_E}{U_o} \tag{5}$$

なお，この式では，スラックを全て消費した場合に t_E が到達する時刻から，スラックの残量を考慮する事で新しい t_E の値を求めているが，これは消費したスラック量を基に t_E の値を更新する事と同等である．

スラックの計算以降に必要な操作は二つである．まず，新たに実行可能になったジョブ J_i に対してスラックを割り当てる際に，ジョブ $J_{n(i)}$ が存在するならば，ジョブ $J_{n(i)}$ に割り当てられていたスラックを減少させる事が必要である（事象 1）．また，その後に，スラックを全て消費せずに終了したジョブ J_i が存在した場合，消費されなかったスラックをジョブ $J_{n(i)}$ に再度割り当てることが必要である（事象 4）．

3.3　インプリサイス計算支援機構の実現方法

インプリサイス計算を実現するためには，付加部分における実行時間を与えられた量以下に制限すること，付加部分を中断する際に必要なコンテキストの操作を行うこと，さらにスケジューリングに必要な情報をスケジューラに伝えることが必要である．

まず，付加部分が割り当てられた時間内に終了できない場合を検出するために，ハードウェアタイマを用いてジョブの実行時間（R_i）を管理する．カーネルはジョブの実行を開始する際にタイマをセットし，そのジョブに与えられた実行時間が全て消費された時点で割り込みを生じるようにする．なお，図 3 において，終端部分が実行可能になるまで，予約した実行時間（w_i）をジョブの実行時間（R_i）に加算しないのは，付加部分の実行時間をこのように管理するためである．与えられた時間以内にジョブが実行を終了した場合，あるいはそれ以前にコンテキストスイッチが生じた場合，スケジューラはタイマを停止する．この時スケジューラは，タイマのカウンタの値を基にそれまで実行していたジョブが利用可能な実行時間を更新する．一方，タイマからの割り込みが先に生じた場合，スケジューラは付加部分を中断し，終端部分を実行可能にする．

付加部分を中断した後に終端部分を実行するためには，ジョブのコンテキストを適切に操作する必要がある．一般的に，異なるコンテキストの処理を行なう手法としては，シグナルハンドラがある．しかし，シグナルハンドラを終端部分の実行に用いた場合，一つの付加部分を中断するために合計で二回のコンテキストスイッチが必要になる．すなわち，付加部分の中断時にシグナルハンドラへコンテキストを変更し，終端部分の終了時にコンテキストを必須部分の先頭へ変更する必要がある．

一方，RT-Frontier では上記のコンテキストの変更を一回に減少させる手法を用いている．この手法は，終端部分を終了した場合には必ず次のサイクルで必須部分を実行するという点に

```
 1: save_context();            /* コンテキストの保存 */
 2: wfunc();                   /* 終端部分 */
 3: for (;;) {
 4:     end_periodic_job();    /* 各サイクルの終了 */
 5:     mfunc();               /* 必須部分 */
 6:     end_mandatory();
 7:     ofunc();               /* 付加部分 */
 8:     end_optional();
 9:     wfunc();               /* 終端部分 */
10: }
```

図 5　インプリサイスタスクを実現するための構造

着目したものである．より具体的には，付加部分を正常に終了した場合と同部分を中断した場合の両方における終端部分以降の実行パスは同一である．この事を利用すると，インプリサイス計算を図 5 に示した構造に従って実行させることが出来る．ただし，左端の数字は行数を表すものである．

図 5 における最初のサイクルは，1 行目でコンテキストの保存のみを行ない，4 行目のシステムコールにより終了する．ここで，本来のコードには 2 行目の終端部分を実行させないための制御が存在するが，紙面の関係上省略した．次のサイクル以降では，付加部分が中断されない限り，必須部分（5 行目），付加部分（7 行目），終端部分（9 行目）と連続的に実行し，必ず 4 行目で各サイクルを終了する．一方，付加部分を中断する必要が生じた際には，スケジューラはコンテキストを 1 行目で保存したものにより上書きする．これにより，このジョブは中断された後，2 行目の終端部分から実行を開始し，正常に付加部分を終了した場合と同様にサイクルを終了する．ゆえに，終端部分を終了した後のコンテキストを変更する必要はない．

最後に，図 5 には，ジョブがスケジューラに必要な情報を伝達するために，必要な二つのシステムコールを併せて示した．図 3 における事象の内，必須部分の終了及び付加部分の終了は，サイクルの終了と同様にスケジューラにより判断する事が出来ない．そこで，これら事象をシステムコールを介してスケジューラに伝達することにした．図 5 では，6 行目に示した `end_mandatory()` がスケジューラに必須部分の終了を伝達するためのシステムコールであり，8 行目に示した `end_optional()` が付加部分の終了を伝達するシステムコールである．これらのシステムコールの呼び出しは必ずプログラム上の正しい場所で行なわれる必要がある．そこで上記の構造をライブラリ化して提供する事で，アプリケーションプログラムを容易に構築できるようにした．

4. 評　　価

本稿で述べたインプリサイス計算支援機構をコストならびにパフォーマンスの観点から評価を行なった．計測は *Responsive Processor* [8] 上で行ない，コンパイルには gcc 3.3.3 を用いた．

4.1　コストに関する計測結果

第一に，カーネル及びインプリサイス計算を実装するために必要なライブラリのコードサイズを計測した結果を表 3 に示す．なお，表 3 には比較のために EDF を用いた場合のサイズを示したが，この場合にインプリサイス計算を実現することは出来ないのでライブラリのコードサイズは 0 である．

表 3　コードサイズの比較

スケジューリング	サイズ (byte)			
アルゴリズム	text	data	bss	合計
SS-OP	42144	376	5424	47944
EDF	39748	376	5416	45540

表 4　SS-OP スケジューラにおける最悪オーバヘッド

事象	オーバヘッド (μs)
リリース	$0.30n + 7.50$
必須部分の終了 (end_mandatory())	3.00
付加部分の中断	9.15
付加部分の終了 (end_optional())	3.10
終端部分の終了 (end_periodic_job())	$0.30n + 8.51$

n：ジョブの数

表 5　実験に用いたジョブのパラメータ

パラメータ	範囲	単位
周期 (T_i)	$[20, 50]$	$1ms$
必須部分の最悪実行時間 (m_i)	$[T_i - 10, T_i + 10]$	$100\mu s$
終端部分の最悪実行時間 (w_i)	$[1, 10]$	$100\mu s$
付加部分の実行時間	$[4m_i - 30, 4m_i + 30]$	$100\mu s$

次に，時間粒度が $0.05\mu s$ であるタイマを用いて，SS-OP スケジューラのオーバヘッドを計測した．システムに存在するジョブが n であるときに，各事象の最悪の場合に対して 100 回づつ計測した結果を表 4 に示す．ただし，これらの値はコンテクストスイッチのオーバヘッド（$13.45\mu s$）を含まないものである．

4.2　パフォーマンスに関する計測結果

付加部分の実行時間が一定でないタスクを含むシステムでは，計算結果の誤差を単一の誤差関数を用いて算出する事は出来ない．そこで，代わりに付加部分がどれだけ実行されたかを測定することにより，インプリサイス計算の生成する結果の精度を概算した．

実験に用いた負荷は，ジョブの時間属性を表 5 に示す範囲の一様分布から選択することにより人工的に合成した．表 5 において他のパラメータを用いて表した範囲は，そのパラメータを先に選択したことを示す．また，これらのうち，付加部分の実行時間以外は同じタスクに属するジョブ間で共通としたが，付加部分の実行時間はジョブ毎に必ず新しく選択した．

実験では，上記の様に合成した全タスクの一番目のジョブを時刻 0 で同時（in phase）にリリースした．各実験は 5 秒間行い，同一のタスクセットによる実験を 5 回づつ行なった．この結果を図 7 及び図 6 に示す．図 7 は必須利用率を横軸としたものであり，図 6 は実質の平均プロセッサ要求率を横軸としたものである．また，いずれの図でも各点はそれぞれのタスクセットにおける付加部分の実行割合の平均を示す．

4.3　考　　察

まず，インプリサイス計算を実装するためのライブラリ及び

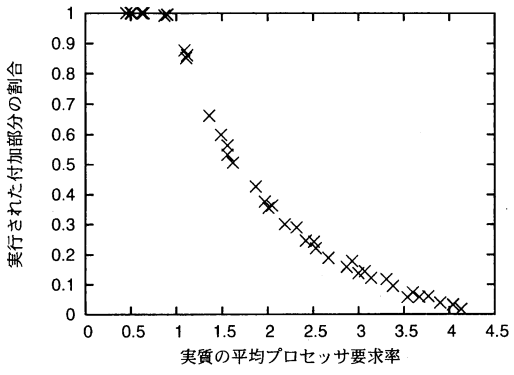


図 6　平均プロセッサ要求量に対する付加部分の実行割合

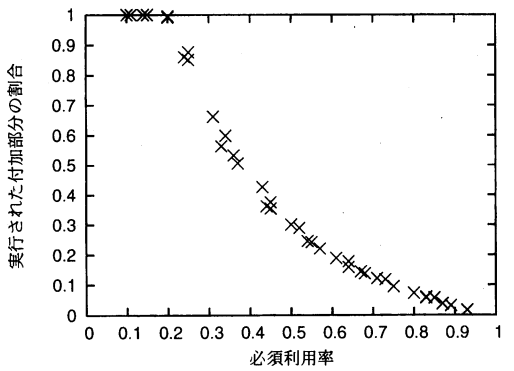


図 7　必須負荷に対する付加部分の実行割合

SS-OP スケジューラの EDF スケジューラに対するコードサイズの増加量はおよそ $2.4KB$ であると見積もることが出来る．これは，オペレーティングシステム全体の大きさの 6% 程度を占める．ただし，data 及び bss セクションの増加量は $8B$ と非常に小さく抑えることが出来ている．したがって，text セクションを ROM に格納することが出来る場合には，RAM の使用量への実質的な影響は無いと言えるであろう．

次に，SS-OP スケジューラによるオーバヘッドであるが，表 4 にはジョブの数に依存する操作が二つ存在する．リリース時のオーバヘッドは，カーネルが EDF 順に保持する実行キューにジョブを追加することによるものであり，終端部分の終了時のオーバヘッドは待機キューにジョブを追加する事によるものである．これらのオーバヘッドがジョブ数の一次関数になっている理由は，ジョブを線形的なキューにより管理したためである．これは，ツリー構造などを利用すれば改善することが可能である．さらに，必須部分及び付加部分の終了時に必要なオーバヘッドが他のものよりも小さいことから，付加部分を中断する必要がない場合にはインプリサイス計算によるオーバヘッドが小さく抑えられるという利点がある．

最後に，付加部分の実行割合を計測した結果から，SS-OP がインプリサイス計算の特徴を利用し，実質的な負荷の調整に成功していることが分かる．まず，図 6 より，全ジョブのプロセッサ要求量の和が 1 を超えた時点で付加部分を中断するようになっている．プロセッサ要求量の和が大きくなるにつれて実行される付加部分の割合は小さくなり，必須利用率が 1 付近ではほぼ全ての付加部分が破棄されていることが図 7 より分かる．

5. ま と め

インプリサイス計算のスケジューリング及び実装の観点から，実時間オペレーティングシステム RT-Frontier におけるインプリサイス計算支援機構について述べた．

本機構は独自の計算モデル及び SS-OP アルゴリズムからなり，付加部分の後に必須部分と同等の終端部分を実行する事を可能にするものである．これにより，プログラマは下位レイヤのソフトウエアに頼っていた付加部分中断後の処理を明示的にアプリケーションに組み入れる事が出来る．このことは，アプリケーションの下位レイヤへの依存性を低下させ，その移植性を向上させるものであると考える．また，終端部分を導入した場合でも，コンテキスト操作によるオーバヘッドを従来のモデルによるものと同等に抑える実装が可能である事を示した．

今後は，ジョブ間の資源共有及び順序制約を扱える様に SS-OP アルゴリズムを拡張する予定である．また，付加部分の平均実行時間がある程度予測可能な場合に，計算結果の品質を制御する機構を追加していきたいと考えている．

謝辞　本論文の研究は，文部科学省の科学技術振興調整費の支援による．また本研究の一部は，科学技術振興機構 CREST の支援による．

文　献

[1] K. Lin, S. Natarajan and J.-S. Liu: "Imprecise Results: Utilizing Partial Computations in Real-Time Systems", Proceedings of the IEEE 8th Real-Time Systems Symposium, pp. 210–217 (1987).

[2] H. Kobayashi and N. Yamasaki: "An Integrated Approach for Implementing Imprecise Computations", IEICE Transactions on Information and Systems, **E86-D**, 10, pp. 2040–2048 (2003).

[3] D. L. Hull and J. W. S. Liu: "ICS: A System for Imprecise Computations", Proceedings of the AIAA Computing in Aerospace, pp. 371–374 (1993).

[4] R. Bettati, N. S. Bowen and J. Y. Chung: "On-Line Scheduling for Checkpointing Imprecise Computation", Proceedings of the Fifth Euromicro Workshop on Real-Time Systems, pp. 238–243 (1993).

[5] W. Feng and J. W.-S. Liu: "Algorithms for Scheduling Real-Time Tasks with Input Error and End-to-End Deadlines", IEEE Trans. Softw. Eng., **23**, 2, pp. 93–106 (1997).

[6] H. Aydin, P. Mejia-Alvarez, R. Melhem and D. Mossé: "Optimal Reward-Based Scheduling of Periodic Real-Time Tasks", Proceedings of the 20th IEEE Real-Time Systems Symposium, pp. 79–89 (1999).

[7] M. Spuri and G. Buttazzo: "Scheduling Aperiodic Tasks in Dynamic Priority Systems", The Journal of Real-Time Systems, **10**, 2, pp. 179–210 (1996).

[8] N. Yamasaki: "Responsive Processor for Parallel/Distributed Real-Time Control", Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1238–1244 (2001).