# 資源制約を持つ拡張インプリサイスタスクの実時間スケジューリング

小林秀典† 山崎信行†

インプリサイス計算モデルに基づくインプリサイスタスクは，その実行に十分な時間が割り当てられない場合でも，デッドラインまでに品質の低い結果を生成することができる．しかし，インプリサイスタスクの実行は任意の時点で中断される可能性があるため，共有資源へアクセスすることができないという欠点がある．本論文では，資源の最大占有時間が既知である環境において，インプリサイスタスクが共有資源へアクセス可能なスケジューリングアルゴリズムを提案する．本アルゴリズムは，システムに存在するスラックのプロセッサバンド幅をオフラインで算出し，スケジュール可能性を判断する．オンラインでは，付加部分に割当て可能な時間を動的に算出することで，実際に要求された資源アクセスが完了できることを確認する．提案したアルゴリズムを実時間オペレーティングシステム RT-Frontier に実装し，実用性を評価する．また，資源制約を持つインプリサイスタスクを用いた検証実験を行うことで，実時間性を満たした資源アクセスが可能であることを示す．

# Real-time Scheduling of Extended Imprecise Tasks with Resource Constraints

Hidenori Kobayashi† and Nobuyuki Yamasaki†

The imprecise tasks, based on the imprecise computation model, can produce lower quality of result even when there is not enough time. However, the imprecise tasks cannot share resources, because they allow termination at any point in their optional parts. This paper proposes a scheduling algorithm that enables the imprecise tasks to share resources in systems where the maximum access duration of every resource is known. The offline part of this algorithm calculates the processor bandwidth of slack and checks the schedulability of the given task set. The online part schedules tasks in deadline order and calculates the remaining time for optional parts in order to verify whether requested accesses can be completed. We implement this algorithm on the RT-Frontier operating system to assess its practicability. Moreover, we conduct experiments using imprecise tasks with resource constraints to confirm that resources can be accessed without causing any timing violation.

## 1. はじめに

動的な実時間システムには，実行時間が大きく変動する処理が存在する[24]．そのため，つねに最悪実行時間を用いて実時間性を保証しようとすれば，プロセッサの利用率が低下してしまう．我々はインプリサイス計算モデル[17]を用いてこの問題を解決する．

本論文で扱うインプリサイス計算モデルは，必須部分（mandatory part）と付加部分（optional part）を持つものとし，このインプリサイス計算モデルに基づくタスクをインプリサイスタスクと呼ぶ．必須部分は許容最低限の品質を持つ結果を生成し，付加部分は必須部分が生成した結果の品質を向上させる．また，付加部分は最終的な結果の論理的な正しさに影響を与えてはならないとする．

インプリサイス計算モデルを用いる場合，必須部分の実時間性を事前に保証しておけば，過負荷状態に陥ったとしても，付加部分を中断することでデッドラインミスを避けることが可能である．そこで，システム設計者は，必須部分に対してのみ最悪実行時間を用いることで，プロセッサ利用率を向上させることができる．一方，プログラマには，付加部分がどの時点で中断されるかが不明であるため，付加部分において中断不可能な資源アクセスを行うことはできないという問題がある．

本論文では，インプリサイスタスクが付加部分において共有資源へアクセス可能な Slack Stealer for Op-

† 慶應義塾大学大学院
Graduate School of Keio University

tional Parts with Shared Resources（SS-OP-SR）アルゴリズムを提案する．

## 2. 拡張インプリサイス計算モデルと資源制約問題

### 2.1 拡張インプリサイス計算モデル

本研究では，従来のインプリサイス計算モデル[17]に対し，**図 1** に示す終端部分を追加した拡張インプリサイス計算モデル[13]を用いる．

従来のインプリサイス計算モデル[17]は，付加部分を中断する際に補完的な処理がまったく必要ないことを仮定している．また，従来のインプリサイス計算モデルのスケジューリングに関する研究[4),5),21)]も，この仮定を設けている．しかし，任意の時点で中断可能な Anytime Algorithm[8),9)]を利用する場合でも，実際には中断の際に何らかの補完的な処理が必要である．

たとえば，センサデータの処理結果を出力するタスクは，付加部分の中断後に最終的な結果を他のタスクに提供するための処理を必要とする．しかし，従来のインプリサイス計算モデルに基づくタスクでは，付加部分の中断後にいかなる処理も実行することはできない．そのため，下位レイヤのソフトウェアによる支援が必要である．

ところが，下位レイヤのソフトウェアはアプリケーションに関する詳細な情報を持たないことが多い．そこで，チェックポイント機構[11),25)]が多く利用されている．しかし，チェックポイントごとのオーバヘッドが大きい場合，システム全体のパフォーマンスが低下することが知られている[20)]．

この問題を解決するために，拡張インプリサイス計算モデルは終端部分を持つ．終端部分は付加部分の後に実行可能になる．ただし，その実行は中断してはならないという制約がある．そのため，下位レイヤの支援がなくても，タスクが終端部分において，計算結果を直接他のタスクに提供することが可能である．

### 2.2 本論文で扱う課題

本論文では，拡張インプリサイス計算モデル基づく拡張インプリサイスタスクの資源制約問題を扱う．拡張インプリサイスタスクには，従来のインプリサイスタスクと同様，資源のアクセス中に付加部分が中断される可能性がある．

資源アクセスが中断された場合，共有データの整合性が失われる可能性がある．さらに，中断によりロックが解放されなくなった場合には，デッドロックが生じる可能性がある．付加部分の中断後にこれらの事態に対処するためには，アクセスが中断された場所を特定しなければならない．しかし，そのためにアクセスログを生成することは効率的とはいえない．

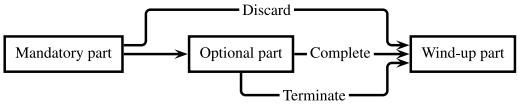


**図 1** 終端部分を有する拡張インプリサイス計算モデル
Fig. 1 Extended imprecise computation with wind-up part model.

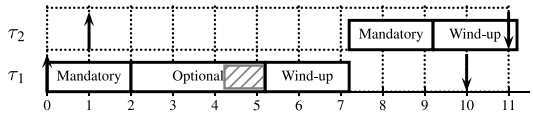


**図 2** 資源アクセスにより生じるデッドラインミスの例
Fig. 2 An example of deadline miss caused by accessing a resource.

また，ハードウェアデバイスなどの物理的な資源に対するアクセスは中断できないことが多い．このとき，デッドラインミスが生じる例を**図 2** に示す．上向きの矢印はリリースタイムを表し，下向きの矢印はデッドラインを表す．タスク $\tau_1$，$\tau_2$ はどちらも，必須部分と終端部分を合わせて 4 の時間を要求するとする．また，スケジューラは，タスク $\tau_1$ の実行を優先するとする．このとき，タスク $\tau_1$ の付加部分が 3 を超える時間を要求するとすると，両方のデッドラインを満たすためには，時刻 5 においてこの付加部分を中断しなければならない．しかし，図 2 に斜線で表した時間において，タスク $\tau_1$ が中断不可能な資源アクセスを行っていた場合，タスク $\tau_1$ の付加部分は時刻 5 よりも遅い時刻に中断されることになり，タスク $\tau_2$ の実行がデッドラインミスを起こす．

以上の問題は，資源アクセスの途中で付加部分を中断しなければならなくなったことが原因である．そこで，本研究では資源アクセスを中断しなければならない事態そのものを回避することを目標とする．

### 2.3 関 連 研 究

実時間システムにおける資源制約を扱うプロトコルとしては，Stack Resource Policy（SRP）方式[2)]，Priority Inheritance 方式，Dynamic Priority Ceiling 方式など[23)]が知られている．しかし，これらの方式は，古典的なタスクモデル[18)]のブロック時間を予測可能にすることを目的としており，中断を許す拡張インプリサイス計算モデルに対してそのまま適用することはできない．

実時間データベースを対象とした研究には，トランザクションの破棄が可能であることを前提とする手法が存在する[10),16),22)]．しかし，物理的な資源に対する

アクセスを実際に中断することは困難である．

一方，資源アクセスの中断を回避可能な手法は動的な負荷に対する柔軟性に問題がある．MARS[15] では資源を静的に割り当てる必要がある．Spring[12] では，資源アクセスの動的なプランニングを行う際に，タスクの正確な実行時間を必要とする．また，Optional Computation Window を求める方式[3] では，2 つ以上のタスクが付加部分を持つことはできない．さらに，OPT-LU アルゴリズム[1] では，付加部分の実行時間に関する上限値が既知であることを仮定しているうえに，結果の品質がステップ関数に従って向上する場合には使用できないという制限がある．

我々は拡張インプリサイス計算モデルを対象として，M-FWP アルゴリズム[13] および SS-OP アルゴリズム[14] を開発した．しかし，これらのスケジューリングアルゴリズムは，終端部分の時間制約を効率的に保証することを目的としており，資源制約を扱うことができないので，本論文において資源の共有を可能にするスケジューリングアルゴリズムを提案する．

## 3. システムに関する仮定および諸定義

**表 1** に本論文で使用する記号の定義を示す．

システムには 1 つのプロセッサおよび $K$ 種類の共有資源とそれらに排他的にアクセスを要求する $L$ 個の周期タスクが存在すると仮定する．ただし，タスクは特定の機能を実現する概念上の仕事であり，拡張インプリサイス計算モデルに基づくものであるとする．

タスクの論理的な実行をジョブと呼ぶ．たとえば，同じ仕事を 2 つの異なる時刻に実行した場合，タスクは 1 つであるがジョブは 2 つ存在したことになる．また，スケジューリングおよび時間の割当ての対象となるのはジョブである．さらに，スケジューラが実際にプロセッサをジョブに割り当てることをジョブを実行するという．

すべてのタスクに関して，ジョブの到着周期，相対デッドライン，必須部分および終端部分の最悪実行時間，資源の最大占有時間を既知の属性であるとする．一方，ジョブ $J_{ij}$ はタスク $\tau_i$ の属性に加えて，リリースタイムと（絶対）デッドラインを属性として持つとする．

ブロック時間の静的な解析を可能にするために，タスクにプリエンプションレベル[2] を付与する．プリエンプションレベルは，ジョブがプリエンプションを起こすための資格を表す静的な指標である．任意のジョブは自らのプリエンプションレベルよりも低いプリエンプションレベルを持つジョブのみをプリエンプトすることが可能である．本論文では特に，$D_i > D_j$ ならば $\pi_i < \pi_j$ を満たす値をプリエンプションレベルとして用いる．

タスクの付加部分の実行時間は未知であり，ジョブごとに固有の値であるとする．このとき，システムの負荷は時間とともに変動するので，評価の際の指標として，システム負荷の期待値 $u_E$ を式 (1) により定義する．

**表 1** 記号の定義
Table 1 Definition of symbols.

| 記号 | 定義 |
|---|---|
| $t_c$ | システム時刻 |
| $\tau_i$ | 周期タスク（$i = 1, 2, \ldots, L$） |
| $Z_i$ | 資源（$i = 1, 2, \ldots, K$） |
| $U_S$ | システムに存在するスラックのプロセッサバンド幅 |
| $u_E$ | システム負荷の期待値 |
| $u_M$ | 必須部分と終端部分のみによるシステム負荷の最低値 |
| $T_i$ | タスク $\tau_i$ のジョブの到着周期 |
| $D_i$ | タスク $\tau_i$ の相対デッドライン |
| $m_i$ | タスク $\tau_i$ の必須部分の最悪実行時間 |
| $o_i$ | タスク $\tau_i$ の付加部分の最悪実行時間の期待値 |
| $w_i$ | タスク $\tau_i$ の終端部分の最悪実行時間 |
| $b_i$ | タスク $\tau_i$ が付加部分において要求する資源の最大占有時間 |
| $\pi_i$ | タスク $\tau_i$ のプリエンプションレベル |
| $\mu_i(\tau_j)$ | 資源 $Z_i$ のタスク $\tau_j$ により要求される最大ユニット数 |
| $\Delta_i(\tau_j)$ | 資源 $Z_i$ のタスク $\tau_j$ による最大占有時間 |
| $J_{ij}$ | タスク $\tau_i$ の $j$ 番目のジョブ（$j = 1, 2, \ldots$） |
| $r_{ij}$ | ジョブ $J_{ij}$ のリリースタイム |
| $d_{ij}$ | ジョブ $J_{ij}$ の（絶対）デッドライン |
| $o_{ij}$ | ジョブ $J_{ij}$ が実際に付加部分を終了するのに必要な時間 |
| $R_i(t)$ | 時刻 $t$ でシステムに存在するジョブ $J_{ij}$ に割り当てられている時間（$S_i(t)$ を含む） |
| $S_i(t)$ | 時刻 $t$ でシステムに存在するジョブ $J_{ij}$ に割り当てられているスラック |
| $J_{ij}^p$ | ジョブ $J_{ij}$ よりも高い優先度を持つジョブの中で最も低い優先度を持つジョブ |
| $d_{ij}^p$ | ジョブ $J_{ij}^p$ のデッドライン |
| $J_{ij}^n$ | ジョブ $J_{ij}$ よりも低い優先度を持つジョブの中で最も高い優先度を持つジョブ |
| $d_{ij}^n$ | ジョブ $J_{ij}^n$ の（絶対）デッドライン |
| $R_i^n(t)$ | 時刻 $t$ でシステムに存在するジョブ $J_{ij}^n$ に割り当てられている時間 |
| $S_i^n(t)$ | 時刻 $t$ でシステムに存在するジョブ $J_{ij}^n$ に割り当てられているスラック |
| $\Gamma_E(t)$ | 時刻 $t$ において実行可能なジョブの集合 |
| $\Gamma_S(t)$ | $\Gamma_E(t)$ のうち付加部分を実行すべきジョブの集合 |
| $\nu_i(t)$ | 時刻 $t$ における資源 $Z_i$ の占有されていないユニット数 |
| $C_i(a)$ | 資源 $Z_i$ の占有されていないユニット数が $a$ である場合のリソースシーリング |
| $B_i$ | SRP 方式を用いた場合にタスク $\tau_i$ がブロックされうる最大時間 |
| $\Pi(t)$ | 時刻 $t$ におけるシステムシーリング |

$$u_E = \sum_{i=1}^{L} \frac{m_i + o_i + w_i}{T_i} \quad (1)$$

また，同じく評価の際の指標として，必須部分と終端部分のみによるシステム負荷の最低値 $u_M$ を式 (2) により定義する．

$$u_M = \sum_{i=1}^{L} \frac{m_i + w_i}{T_i} \quad (2)$$

タスク $\tau_i$ の実行時間が $m_i$，$b_i$，$w_i$ の和であると仮定したうえで，区間 $[t_1, t_2)$ において任意のジョブの実行を $\delta$ だけ遅らせてもデッドラインミスを生じないとき，区間 [t1, t2) には $\delta$ だけのスラックが存在すると定義する．また，この区間におけるスラックのプロセッサバンド幅を $\delta/(t_2 - t_1)$ により求めるとする．さらに，あらゆる区間におけるスラックのプロセッサバンド幅の最小値を，システムに存在するスラックのプロセッサバンド幅 $U_S$ と定義する．よって，$U_S \geq 0$ のとき，任意の区間 $[t_1, t_2)$ には必ず $(t_2 - t_1)U_S$ 以上のスラックが存在する．

本論文では，説明を簡潔にするために，ジョブはスケジューラにより明示的に削除されない限り，実行完了後もそのデッドラインまでシステムに存在すると仮定する．また，ジョブはスケジューラが割り当てた時間をその実行により消費するとし，この割り当てられている時間の残量 $R_i(t)$ が付加部分の実行中に $w_i$ 以下になることをオーバランと呼ぶ．さらに，オーバランしたジョブが付加部分の実行を継続している状態をオーバラン状態と呼ぶ．

次に，資源は横取り不能であり，等価な資源が複数存在する場合には 1 つの資源が複数ユニット存在すると見なす．また，ブロック時間の解析を容易にするために，ある資源アクセスが別の資源アクセスを内包する場合，資源の解放は獲得と逆順に行われるとし，各資源の占有時間は内包する資源の占有時間を含むとする．この仮定の下では，たとえば，資源 $Z_1$ を獲得しているジョブがさらに資源 $Z_2$ を獲得した場合，資源 $Z_2$ を資源 $Z_1$ よりも先に解放しなければならない．また，資源 $Z_1$ への競合による最大ブロック時間を解析する際には，資源 $Z_1$ の占有時間が資源 $Z_2$ の占有時間を含むので，資源 $Z_2$ の占有時間を考慮する必要がない．

資源 $Z_i$ のリソースシーリングおよびシステムシーリングを，Baker [2] と同様に，それぞれ式 (3) および式 (4) と定義する．

$$C_i(a) = \max\{\{0\} \cup \{\pi_j | a < \mu_i(\tau_j), \\ 1 \leq j \leq L\}\} \quad (3)$$

$$\Pi(t) = \max\{C_i(\nu_i(t)) | 1 \leq i \leq K\} \quad (4)$$

ここで，資源 $Z_i$ のリソースシーリングは，資源 $Z_i$ を要求することでブロックされる可能性のあるすべてのジョブのプリエンプションレベルの最大値である．したがって，リソースシーリングは資源の利用可能なユニット数の関数である．また，全ユニットが利用可能である場合のリソースシーリングは 0 であると考える．さらに，全リソースシーリングの最大値をシステムシーリングとする．このとき，SRP 方式においてタスク $\tau_i$ がブロックされうる最大時間 $B_i$ は式 (5) により定義される．

$$B_i = \max\{\{0\} \cup \{\Delta_k(\tau_j) | \pi_j < \pi_i, C_k(0) \geq \pi_i, \\ 1 \leq j \leq L, 1 \leq k \leq K\}\} \quad (5)$$

## 4. SS-OP-SR アルゴリズム

実時間システムにおいて，プロセッサの利用率を向上させるためには，デッドラインを指標としたオンラインスケジューリングを行うことが望ましい．また，開始時刻が予測できない資源アクセスが存在する場合，オンラインで資源アクセスを制御する必要がある．SS-OP-SR アルゴリズムでは，オンラインで付加部分に割当て可能な時間を，スラックのプロセッサバンド幅 $U_S$ を基に算出する．ここで，仮定より，システムに存在するスラックのプロセッサバンド幅は動的に変動しない．また，相対デッドラインが周期より短い場合，スラックのプロセッサバンド幅の算出には，準多項式（pseudo-polynomial）オーダの計算量が必要であり [23]，これをオンラインで行うのは困難である．そのため，SS-OP-SR アルゴリズムでは，オフラインでスラックのプロセッサバンド幅を算出し，オンラインで実際のスラックの割当ておよびスケジューリングを行う．

1 つのスケジューリングアルゴリズムをオフラインアルゴリズムとオンラインアルゴリズムから構成することは，利便性を低下させる可能性があるが，この構成により，オンラインでのスラックの割当てに必要な計算量を $O(1)$ に抑えることができる．このとき，ジョブの管理に選択木 [7] を用いることで，オンラインアルゴリズムの計算量を $O(\lceil \log_2 L \rceil)$ に抑えることができる．また，組み込みシステムでは，オフラインアルゴリズムをターゲット上に実装する必要はないため，オンラインアルゴリズムのみにより構成する場合と比べて，使用するメモリ量を小さくできるという利点もある．

**Procedure SlackBw**:
$i < j$ ならば $\pi_i \geq \pi_j$ とする;
$c_i = m_i + b_i + w_i$;
$\lambda_i(\ell) = 1 + \left\lfloor \frac{\ell - D_i}{T_i} \right\rfloor$;
$\sigma_i(\ell) = \sum_{k=1}^{i} \lambda_k(\ell) c_k + \lambda_i(\ell) B_i$;
$\zeta = \max \left\{ D_L, \frac{\sum_{i=1}^{L} \left(1 - \frac{D_i}{T_i}\right) c_i}{1 - U} \right\}$;
$\Omega_i = \left\{ mT_i + D_i \;\middle|\; m = 0, 1, \ldots, \left\lfloor \frac{\zeta - D_i}{T_i} \right\rfloor \right\}$;
$U = \sum_{i=1}^{L} \frac{c_i}{T_i}$
**if** $U \geq 1$ **then**
  $U_S := 1 - U$;
**else**
  $U_S := \min \left\{ \frac{\ell - \sigma_i(\ell)}{\ell} \;\middle|\; \ell \in \Omega_i, 1 \leq i \leq L \right\}$;
**endif**

**図 3** SlackBw 手続きの疑似コード
Fig. 3 Procedure SlackBw in pseudo-code.

SS-OP-SR アルゴリズムでは，資源アクセスによるデッドロックを回避し，ジョブごとに 2 つ以上の優先度逆転を生じさせないために，SRP 方式におけるプリエンプションの制御規則を利用する．よって，最も早いデッドラインを持つタスクのプリエンプションレベルがシステムシーリングよりも高い場合にのみ，プリエンプションを許可する．さらに，SS-OP-SR アルゴリズムでは，資源の要求時にアクセス制御を行うことにより資源アクセスの完全性を保証する．このとき，ジョブのブロック時間を 1 つの競合する資源の占有時間に抑えることができ，その最大値は SRP 方式における最大ブロック時間 $B_i$ と同一になる．本論文ではこの証明を 4.4 節に示す．

### 4.1 オフラインアルゴリズム

システムに存在するスラックのプロセッサバンド幅を求め，スケジュール可能性を判断するために，オフラインアルゴリズムを規則 1 により定義する．

**規則 1** **図 3** に示す SlackBw 手続きによりスラックのプロセッサバンド幅 $U_S$ を算出する．この結果，$U_S > 0$ である場合にのみ，与えられたタスクセットを受諾する．

SlackBw 手続きでは，$U_S$ を算出するために，$m_i$，$b_i$，$w_i$ の和として得られる時間 $c_i$ をタスク $\tau_i$ に対して予約し，SRP 方式における最大ブロック時間 $B_i$ を含んだプロセッサ要求量 $\sigma_i(\ell)$ を解析する．SS-OP-SR アルゴリズムでは，SRP 方式と同様にプリエンプションレベルを基にしたプリエンプション制御を行うので，ジョブの最大ブロック時間は SRP 方式における最大ブロック時間 $B_i$ と等しい．そこで，SlackBw 手続きでは，資源の競合による最大ブロック時間として SRP 方式の最大ブロック時間 $B_i$ を用いる．また，プロセッサ要求量の解析区間の上限の算出は Zheng らの方法[26] による．

この解析の結果，$U_S \leq 0$ である場合には，SS-OP-SR アルゴリズムは実時間性を保証することができないので，与えられたタスクセットを受諾しない．ただし，$U_S$ はあらゆる区間におけるスラックのプロセッサバンド幅の最小値であるので，$U_S \leq 0$ であっても，区間によっては SS-OP-SR アルゴリズムで利用できないスラックが存在することがある．また，資源アクセスの完全性を保証するために，SlackBw 手続きは付加部分の実行時間を資源の最大占有時間と等しいと仮定して $U_S$ を算出するので，受諾しないタスクセットにもスケジュール可能なものが含まれることがある．すなわち，$U_S \leq 0$ であっても，タスク $\tau_i$ の付加部分を $b_i$ よりも短い時間しか実行しなければスケジュール可能な場合がある．しかし，このとき，付加部分における資源アクセスが割当て可能な時間内に完了することを保証できないので，オーバランによるデッドラインミスが生じる可能性がある．そのため，SS-OP-SR アルゴリズムは $U_S \leq 0$ となるタスクセットを受諾しない．

### 4.2 オンラインアルゴリズム

ジョブをスケジューリングし，スラックとオフラインで予約した時間を割り当てるために，オンラインアルゴリズムを規則 2 から規則 6 により定義する．

**規則 2** 任意の 2 つのジョブ $J_{ij}$，$J_{kl}$ の優先度は以下の手順により決定する．

(2-1) $d_{ij} < d_{kl}$ ならば，ジョブ $J_{ij}$ にジョブ $J_{kl}$ より高い優先度を与える．
(2-2) $d_{ij} = d_{kl}$ かつ $D_i < D_k$ ならば，ジョブ $J_{ij}$ にジョブ $J_{kl}$ より高い優先度を与える．
(2-3) $d_{ij} = d_{kl}$ かつ $D_i = D_k$ であり，さらに $i < k$ である場合には，ジョブ $J_{ij}$ にジョブ $J_{kl}$ より高い優先度を与える．

**規則 3** ジョブ $J_{ij}$ がシステムに到着したとき，以下の手順を行う．

(3-1) **図 4** に示す AllocTime 手続きを実行し，ジョブ $J_{ij}$ に対しオフラインで予約した時間およびスラックを割り当てる．
(3-2) **図 5** に示す ChkPreempt 手続きにより，実行するジョブを選択する．

**規則 4** ジョブ $J_{ij}$ を 1 単位時間だけ実行したとき，$R_i(t)$ を 1 単位時間減少させる．さらに，ジョブ $J_{ij}$ がこの実行中に $\Gamma_S(t)$ に属し，$S_i(t) > 0$ であったならば，同時に $S_i(t)$ を 1 単位時間減少させる．

**規則 5** 実行中のジョブ $J_{ij}$ がオーバランしたとき，その付加部分を中断し，終端部分を開始する．

**Procedure AllocTime**($J_{ij}$):
$\varepsilon$ はローカル変数;
$\varepsilon := r_{ij}$;
**if** $\exists J_{ij}^{p}$ **then**
$\varepsilon := \max\left\{\varepsilon, d_{ij}^{p}\right\}$;
**endif**
**if** $\exists J_{ij}^{n}$ **then**
$\varepsilon := \max\left\{\varepsilon, d_{ij}^{n} - \frac{S_i^n(t_c)}{U_S}\right\}$;
**endif**
$S_i(t_c) := \max\{0, d_{ij} - \varepsilon\}U_S$;
$R_i(t_c) := m_i + b_i + w_i + S_i(t_c)$;
**if** $\exists J_{ij}^{n}$ **then**
$R_i^n(t_c) := R_i^n(t_c) - S_i(t_c)$;
$S_i^n(t_c) := S_i^n(t_c) - S_i(t_c)$;
**endif**

**図 4** AllocTime 手続きの疑似コード
Fig. 4 Procedure AllocTime in pseudo-code.

**Procedure ChkPreempt**($J_{ij}$):
実行中のジョブを $J_{kl}$ とする;
$\Gamma_E(t_c)$ で最も優先度の高いジョブを $J_{mn}$ とする；
**if** $J_{ij} = J_{mn}$ and $\pi_i > \Pi(t_c)$ **then**
ジョブ $J_{ij}$ の実行を開始;
**else**
ジョブ $J_{kl}$ の実行を継続;
**endif**

**図 5** ChkPreempt 手続きの疑似コード
Fig. 5 Procedure ChkPreempt in pseudo-code.

**Procedure Reclaim**($J_{ij}$):
$\varphi$ はローカル変数;
**if** $\exists J_{ij}^{n}$ **then**
$R_i^n(t_c) := R_i^n(t_c) + R_i(t_c)$;
$S_i^n(t_c) := S_i^n(t_c) + R_i(t_c)$;
**endif**
$\varphi := d_{ij} - \frac{R_i(t_c)}{U_S}$;
**if** $\varphi \leq t_c$ **then**
ジョブ $J_{ij}$ をシステムから削除;
**else**
$d_{ij} := \varphi$;
**endif**
$R_i(t_c) := 0$;
$S_i(t_c) := 0$;

**図 6** Reclaim 手続きの疑似コード
Fig. 6 Procedure Reclaim in pseudo-code.

**規則 6** 実行中のジョブ $J_{ij}$ が終了したとき，以下の手順を行う．
(6-1) **図 6** に示す Reclaim 手続きを実行する．
(6-2) **図 7** に示す SlctResume 手続きにより，次に実行するジョブを選択する．

上記の規則の動作をジョブの実行順序の決定とスラックの管理の観点から以下に示す．

（1） 実行順序の決定

ジョブの実行順序はブロック時間に影響を与える．また，動的にジョブをスケジューリングする場合，計算

**Procedure SlctResume**($J_{ij}$):
$\Gamma_E(t_c)$ で最も遅くに実行したジョブを $J_{kl}$ とする;
$\Gamma_E(t_c)$ で最も優先度の高いジョブを $J_{mn}$ とする;
**if** $\Gamma_E(t_c) \neq \emptyset$ **then**
**if** $\pi_m > \Pi(t_c)$ **then**
ジョブ $J_{mn}$ の実行を開始;
**else**
ジョブ $J_{kl}$ の実行を再開;
**endif**
**endif**

**図 7** SlctResume 手続きの疑似コード
Fig. 7 Procedure SlctResume in pseudo-code.

量の問題から優先度逆転をすべて排除することは難しい．したがって，ジョブごとに 2 つ以上の優先度逆転が生じないことを保証する SRP 方式が，現実的には資源制約によるブロック時間を最も短くすることができるといえる．そこで，最大ブロック時間を SRP 方式と同一に抑えるために，プリエンプションレベルと規則 2 により定まる優先度を用いてプリエンプションの是非を判断し，実行順序を決定する．

規則 3 の ChkPreempt 手続きでは，システムに到着したジョブ $J_{ij}$ によるプリエンプションの是非を判断する．同手続きは，ジョブ $J_{ij}$ の優先度が実行可能なジョブの中で最も高く，そのプリエンプションレベル $\pi_i$ がシステムシーリング $\Pi(t_c)$ よりも高い場合にのみ，ジョブ $J_{ij}$ の実行を新たに開始する．

規則 6 の SlctResume 手続きでも同様に，最高優先度のジョブ $J_{mn}$ のプリエンプションレベル $\pi_m$ がシステムシーリング $\Pi(t_c)$ よりも高い場合にのみ，ジョブ $J_{mn}$ の実行を新たに開始する．これは，$\Gamma_E(t_c)$ で最も遅く実行したジョブ $J_{kl}$ をジョブ $J_{mn}$ がプリエンプトすることになるためである．それに対し，ジョブ $J_{kl}$ の実行を再開する場合には，プリエンプションは発生しないので，ジョブ $J_{kl}$ のプリエンプションレベルはシステムシーリングより低くてもかまわない．

（2） スラックの管理

システムに到着したジョブには，規則 3 の AllocTime 手続きが，オフラインで予約された時間に加えて，システムのスラックを割り当てる．

AllocTime 手続きがジョブ $J_{ij}$ に割り当てるスラックは，その実行可能な区間 $[r_{ij}, d_{ij})$ から算出する．ただし，この区間内で，ジョブ $J_{ij}$ よりも優先度の高いジョブが実行可能な区間は除外する．また，ジョブ $J_{ij}$ の到着前に，ジョブ $J_{ij}$ より低い優先度のジョブによりスラックが消費された区間も同様に除外する．この結果，ジョブ $J_{ij}$ に対してスラックを割当て可能な場合，スラックの算出区間は $[\hat{\varepsilon}, d_{ij})$ となる．ただし，$\hat{\varepsilon}$ は AllocTime 手続きにおけるローカル変数 $\varepsilon$ の最終

的な値であるとする．このとき，ジョブ $J_{ij}$ に対して，$(d_{ij} - \hat{\varepsilon})U_S$ だけのスラックを割り当てる．

ジョブに割り当てられたスラックは，付加部分において実際に要求される時間より少ない場合がある．規則 4 は，この場合に生じるオーバランを検出するために，実行を開始したジョブに割り当てられた時間を減少させる．また，後述の資源アクセス制御のために，付加部分の実行においては予約された時間よりもスラックを先に消費させる．

一方，割り当てられた時間をすべて消費する前にジョブが終了する場合がある．そこで，規則 6 の Reclaim 手続きは，プロセッサの利用率を向上させるために 2 つの処理を行う．1 つは，終了したジョブ $J_{ij}$ が保持する未使用の時間を直接ジョブ $J_{ij}^n$ に割り当てることである．もう 1 つは，ジョブ $J_{ij}$ によりスラックが消費された区間を AllocTime 手続きから特定できるようにすることである．Reclaim 手続きのローカル変数 $\varphi$ を用いると，スラックが消費された区間は $[r_{ij}, \varphi)$ と表すことができる．そこで，$\varphi > t_c$ である場合，この区間のみをスラックの算出区間から除外するために，ジョブ $J_{ij}$ のデッドラインを $\varphi$ に変更する．

### 4.3 資源アクセス制御

タスク $\tau_i$ の付加部分では，オンラインアルゴリズムにより，$b_i$ 以上の時間を消費することができる．しかし，これだけでは付加部分における資源アクセスの完全性を保証できないので，資源アクセスをオンラインで制御する．

資源アクセスを制御する手続きを**図 8**，**図 9**，**図 10** に示す．このうち，Down 手続きおよび TryDown 手続きはジョブ $J_{ij}$ が資源 $Z_k$ を $n$ ユニット要求する操作であり，Up 手続きは逆にジョブ $J_{ij}$ が資源 $Z_k$ を $n$ ユニット解放する操作である．

Down 手続きおよび TryDown 手続きでは，資源アクセスの完全性を保証するために，確実に資源アクセスを完了できる場合を除き，資源要求を拒否する．$\Gamma_S(t)$ に属するジョブ $J_{ij}$ が資源アクセスを完了するためには，アクセスの開始時点で，その資源の占有時間以上の時間を保持していることが必要である．ただし，時間 $R_i(t)$ に含まれる終端部分の最悪実行時間 $w_i$ を付加部分で消費することはできない．また，スラック $S_i(t)$ は新たなジョブの到着とともに減少する可能性がある．ゆえに，$\Gamma_S(t)$ に属すジョブ $J_{ij}$ が付加部分で確実に消費可能な時間は $R_i(t) - S_i(t) - w_i$ となる．そこで，この値が要求された資源の占有時間よりも大きい場合に資源の獲得を許可する．また，資源の獲得に成功するためには，割り当てられた時間に含まれるスラックは少ない方が望ましいことから，オンラインアルゴリズムの規則 4 は，予約された時間よりもスラックを先に消費させる．

```
Procedure Down(J_ij, Z_k, n):
    ξ_i(t_c) = R_i(t_c) − S_i(t_c) − w_i;
    if J_ij ∉ Γ_S(t_c) or ξ_i(t_c) ≥ Δ_k(τ_i) then
        ジョブ J_ij は資源 Z_k を n ユニット獲得;
    else
        ジョブ J_ij は付加部分を中断し，終端部分を開始;
    endif
```

**図 8** Down 手続きの疑似コード
Fig. 8 Procedure Down in pseudo-code.

```
Procedure TryDown(J_ij, Z_k, n):
    ξ_i(t_c) = R_i(t_c) − S_i(t_c) − w_i;
    if J_ij ∉ Γ_S(t_c) or ξ_i(t_c) ≥ Δ_k(τ_i) then
        ジョブ J_ij は資源 Z_k を n ユニット獲得;
    endif
```

**図 9** TryDown 手続きの疑似コード
Fig. 9 Procedure TryDown in pseudo-code.

```
Procedure Up(J_ij, Z_k, n):
    Γ_E(t_c) 内で最も優先度の高いジョブを J_gh とする;
    J_ij は保持する資源 Z_k を n ユニット解放;
    if J_gh ≠ J_ij and π_g > Π(t_c) then
        ジョブ J_ij からジョブ J_gh への実行の切替え;
    endif
```

**図 10** Up 手続きの疑似コード
Fig. 10 Procedure Up in pseudo-code.

Down 手続きと TryDown 手続きの違いは，想定する資源アクセスの性質および資源の獲得に失敗した場合の動作にある．Down 手続きは実行に不可欠な資源へのアクセスを想定し，資源の獲得に失敗したジョブの付加部分を中断する．それに対し，TryDown 手続きは補助的な資源へのアクセスを想定し，資源の獲得に失敗した場合でも実行の継続を許す．

資源を解放する Up 手続きは必ず成功する．資源の解放時には，システムシーリングが下がる場合がある．システムシーリング $\Pi(t_c)$ が最高優先度のジョブ $J_{gh}$ のプリエンプションレベル $\pi_g$ よりも低くなるとき，ジョブ $J_{gh}$ の実行に必要な資源はすべて利用可能な状態になる．そこで，このとき，ジョブ $J_{gh}$ へ実行を切り替える．

### 4.4 スケジュール可能性

資源アクセスにともなうブロック時間に関して，補題 1 が成り立つ．

**補題 1** SS-OP-SR アルゴリズムが生成するスケジュールにおいて，任意のタスク $\tau_i$ がブロックされうる最大時間は SRP 方式における最大ブロック時間 $B_i$ と等しい．

（証明）SS-OP-SR アルゴリズムにおけるプリエンプ

ションレベルはSRP方式の基準を満たす．さらに，いかなるジョブ $J_{ij}$ も $\pi_i > \Pi(t)$ が成立する時刻 $t$ まで実行を開始することはない．また，すべての資源は必ず解放される．よって，Baker[2] より，補題1が成り立つ． □

提案したオンラインアルゴリズムは $U_S > 0$ である場合のみを対象とするため，補題2が成り立つ．

**補題2** SS-OP-SRアルゴリズムにおいてReclaim手続きを実行しない場合，実際に消費されるスラックのプロセッサバンド幅は $U_S$ を超えることがない．

(証明) Reclaim手続きを実行しない場合，AllocTime手続きのみがジョブにスラックを割り当てる．また，システムに最初に到着したジョブに割り当てられるスラックのプロセッサバンド幅は明らかに $U_S$ を超えない．

そこで，すでに割り当てられているスラックのプロセッサバンド幅が $U_S$ 以下である任意の区間 $[t_1, t_2)$ において，ジョブ $J_{ij}$ が到着した場合を考える．ただし，$t_1 \le r_{ij} < d_{ij} \le t_2$ であるとする．

まず，AllocTime手続きにより $S_i(t) = 0$ となる場合，区間 $[t_1, t_2)$ において消費されるスラックの量は変化しない．

次に，$S_i(t) > 0$ となる場合をジョブ $J_{ij}^n$ の有無により分けて考える．ジョブ $J_{ij}^n$ が存在しない場合，区間 $[t_1, t_2)$ におけるスラックの要求量 $h(t_1, t_2)$ は，区間 $[t_1, t_2)$ においてすでに割り当てられているスラックのプロセッサバンド幅が $U_S$ 以下であるという仮定より，

$$\begin{aligned} h(t_1, t_2) &= h(t_1, \hat{\varepsilon}) + (d_{ij} - \hat{\varepsilon})U_S \\ &\le (\hat{\varepsilon} - t_1)U_S + (d_{ij} - \hat{\varepsilon})U_S \\ &\le (t_2 - t_1)U_S \end{aligned} \tag{6}$$

を満たす．ただし，$\hat{\varepsilon}$ はAllocTime手続きにおけるローカル変数 $\varepsilon$ の最終的な値である．一方，ジョブ $J_{ij}^n$ が存在する場合，同様に $\hat{\varepsilon}$ を用いて，

$$\begin{aligned} S_i(t) &= (d_{ij} - \hat{\varepsilon})\, U_S \\ &\le \left( d_{ij} - \left( d_{ij}^n - \frac{S_i^n(t)}{U_S} \right) \right) U_S \\ &\le S_i^n(t) \end{aligned} \tag{7}$$

であるから，区間 $[t_1, t_2)$ において消費されるスラックの量は変化しない．

以上より，つねに

$$h(t_1, t_2) \le (t_2 - t_1)U_S \tag{8}$$

が成り立つので，区間 $[t_1, t_2)$ において消費されるスラックのプロセッサバンド幅は $U_S$ を超えない． □

補題1および補題2を利用して，スケジュール可能性を定理1に示す．

**定理1** $U_S > 0$ のとき，SS-OP-SRアルゴリズムによりスケジュール可能である．

(証明) $i < j$ ならば $\pi_i \ge \pi_j$ とする．このとき補題1および補題2から，Reclaim手続きを実行しない場合には，任意の区間 $[t_1, t_2)$ におけるプロセッサ要求量に関して，

$$\begin{aligned} &\forall i, 1 \le i \le L, \\ &\sum_{k=1}^{i} \left( 1 + \left\lfloor \frac{t_2 - t_1 - D_k}{T_k} \right\rfloor \right) (m_k + b_k + w_k) \\ &\quad + \left( 1 + \left\lfloor \frac{t_2 - t_1 - D_i}{T_i} \right\rfloor \right) B_i + (t_2 - t_1)U_S \\ &\qquad\qquad \le t_2 - t_1 \end{aligned} \tag{9}$$

が成り立つ．

ここで，Reclaim手続きによる影響を考える．ある時刻 $t_f$ において終了したジョブ $J_{ij}$ に関して，Reclaim手続きを実行する前は $R_i(t_f) = R$, $S_i(t_f) = S$, $d_{ij} = d$ であったとする．このとき，区間 $[d - S/U_S, d)$ に存在するスラックが未消費であったと見なすことができる．よって，$R = S$ である場合には，Reclaim手続きを実行しても式(9)が成り立つ．また，表1における $R_i(t)$ および $S_i(t)$ の定義より，$R < S$ であることはない．

$R > S$ である場合，Reclaim手続き以降に実行されるジョブは，区間 $[t_f, d)$ において最大で $R - S$ の時間を余分にスラックとして消費する可能性がある．しかし，$R > S$ が成り立つためには，ジョブ $J_{ij}$ の終了以前に，予約した時間をすべて消費せずに終了したジョブが1つ以上存在し，その未消費の時間の総和は $R - S$ 以上であるはずである．よって，Reclaim手続きを実行した場合にも式(9)が成り立つ．

ゆえに，$U_S > 0$ のとき，任意の区間において消費される時間の総和はその区間の長さを超えることがないので，スケジュール可能である． □

以上より，SS-OP-SRアルゴリズムを用いたシステムのスケジュール可能性は，スラックのプロセッサバンド幅 $U_S$ が0より大きいことを確認することで判断可能である．

### 4.5 実 行 例

**表2** に示すタスクセットを用いて，SS-OP-SRアルゴリズムの動作例を示す．ただし，すべてのタスクはシステムに1ユニット存在する資源 $Z_1$ を共有するとする．また，資源アクセスは付加部分の最後のみで行われ，その占有時間は2であるとする．すなわち，ジョブ $J_{ij}$ は付加部分を $o_{ij} - 2$ だけ実行した時点で資源 $Z_1$ を要求する．さらに，タスク $\tau_1$, $\tau_3$ は資源要求にTryDown手続きを使用し，タスク $\tau_2$ はDown

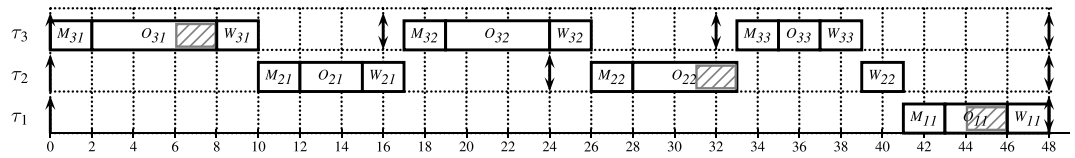


**図 11** SS-OP-SR アルゴリズムによる実行例
Fig. 11 An example schedule created by the SS-OP-SR algorithm.

**表 2** タスクセット
Table 2 Task set.

| $\tau_i$ | $m_i$ | $b_i$ | $w_i$ | $T_i$ | $D_i$ | $\pi_i$ | $r_{i1}$ | $o_{ij}$ |
|---|---|---|---|---|---|---|---|---|
| $\tau_1$ | 2 | 2 | 2 | 48 | 48 | 1 | 0 | 3 |
| $\tau_2$ | 2 | 2 | 2 | 24 | 24 | 2 | 0 | 5 |
| $\tau_3$ | 2 | 2 | 2 | 16 | 16 | 3 | 0 | 6 |

**表 3** タスクの実行に割り当てられた時間とスラックの残量
Table 3 Remaining time and slack allocated to task execution.

| $t_c$ | $R_1(t_c)$ | $S_1(t_c)$ | $R_2(t_c)$ | $S_2(t_c)$ | $R_3(t_c)$ | $S_3(t_c)$ |
|---|---|---|---|---|---|---|
| 0 | 12 | 6 | 8 | 2 | 10 | 4 |
| 6 | 12 | 6 | 8 | 2 | 4 | 0 |
| 10 | 12 | 6 | 8 | 2 | 0 | 0 |
| 15 | 12 | 6 | 3 | 0 | 0 | 0 |
| 16 | 10 | 4 | 2 | 0 | 8 | 2 |
| 17 | 10 | 4 | 0 | 0 | 9 | 3 |
| 23 | 10 | 4 | 0 | 0 | 3 | 0 |
| 24 | 6 | 0 | 10 | 4 | 2 | 0 |
| 31 | 6 | 0 | 5 | 1 | 0 | 0 |
| 32 | 6 | 0 | 4 | 0 | 6 | 0 |
| 44 | 4 | 0 | 0 | 0 | 0 | 0 |

手続きを使用するとする．

このとき，オフラインアルゴリズムは，SlackBw 手続きによりスラックのプロセッサバンド幅 $U_S$ を 0.25 と算出し，表 2 のタスクセットを受諾する．

実際にオンラインアルゴリズムにより生成されるスケジュールを**図 11** に示す．図中の $M_{ij}$，$O_{ij}$，$W_{ij}$ はそれぞれジョブ $J_{ij}$ の必須部分，付加部分，終端部分の実行を表す．また，**表 3** に，スケジューリング後に各タスクが保持する時間とスラックの残量を示す．表 3 の各時刻における動作を以下に示す．

(1) システム時刻 0 では，到着したすべてのジョブに対し，AllocTime 手続きを実行する．これにより，ジョブ $J_{31}$，$J_{21}$，$J_{11}$ に，それぞれ区間 $[0,16)$，$[16,24)$，$[24,48)$ に存在するスラックを割り当てる．

(2) システム時刻 6 では，ジョブ $J_{31}$ が資源 $Z_1$ を要求する．ジョブ $J_{31}$ が付加部分で消費可能なスラック以外の時間は 2 であり，これは資源 $Z_1$ の占有時間と等しいので，ジョブ $J_{31}$ は資源を獲得する．

(3) システム時刻 10 では，終了したジョブ $J_{31}$ に対し，Reclaim 手続きを実行する．ただし，ジョブ $J_{31}$ には未消費の時間がないので他のジョブに割り当てられた時間に変更はない．

(4) システム時刻 15 では，ジョブ $J_{21}$ が資源 $Z_1$ を Down 手続きにより要求する．しかし，ジョブ $J_{21}$ の付加部分に残されたスラック以外の時間は 1 しかないため，資源の獲得に失敗する．そこで，ジョブ $J_{21}$ の付加部分をこの時点で中断する．

(5) システム時刻 16 では，到着したジョブ $J_{32}$ に対し，区間 $[24,32)$ に存在するスラックを割り当て，ジョブ $J_{11}$ が保持するスラックを同量だけ減少させる．

(6) システム時刻 17 では，終了したジョブ $J_{21}$ が保持する 1 の時間を，Reclaim 手続きによりジョブ $J_{32}$ の保持する時間に加算する．

(7) システム時刻 23 では，ジョブ $J_{32}$ が資源 $Z_1$ の獲得に失敗する．しかし，ジョブ $J_{32}$ は TryDown 手続きを用いたので，システム時刻 24 まで付加部分の実行を継続することができる．

(8) システム時刻 24 では，到着したジョブ $J_{22}$ に対し，区間 $[32,48)$ に存在するスラックを割り当て，ジョブ $J_{11}$ が保持するスラックを同量だけ減少させる．

(9) システム時刻 31 では，ジョブ $J_{22}$ が資源 $Z_1$ の獲得に成功する．ここで，仮にジョブ $J_{22}$ が付加部分でスラックよりも予約された時間を先に消費した場合を考える．この場合，ジョブ $J_{22}$ がシステム時刻 31 で保持する 5 の時間にはスラックが 3 だけ含まれるので，資源の獲得を許可するべきではない．すなわち，ジョブ $J_{22}$ が消費可能なスラックはジョブ $J_{33}$ の実行により減少するので，仮に資源の獲得を許可した場合には，オーバランを避けるために $J_{22}$ の資源アクセスを中断しなければならなくなる．

(10) システム時刻 32 では，到着したジョブ $J_{33}$ に予約された時間を割り当てる．

(11) システム時刻 44 では，ジョブ $J_{11}$ が資源 $Z_1$ の獲得に成功する．ここで，仮にジョブ $J_{11}$ の付加部分の実行時間 $o_{11}$ が実際には 4 であったとする．このときに $o_{11}$ を誤って 3 と見積もり，静的に資源 $Z_1$ を割り当てていたとすると，システム時刻 46 でジョブ $J_{11}$ の資源アクセスを中断しなければならなくなる．また，アクセスを中断できない場合，ジョブ $J_{11}$

がデッドラインミスを起こす．それに対し，資源アクセス制御をオンラインで行う SS-OP-SR アルゴリズムでは，システム時刻 45 における資源要求を拒否するので，デッドラインミスを起こすことがない．

## 5. 評　　価

SS-OP-SR アルゴリズムの実用性を評価するために，レスポンシブプロセッサ[27] をターゲットとした RT-Frontier カーネルに本手法を実装し，資源アクセス制御に関するオーバヘッドおよびカーネルのコードサイズを計測した．さらに，資源制約を持つ拡張インプリサイスタスクを用いた実験により実時間性の検証を行った．

コンパイルには，GNU[6] のツールである gcc（バージョン 3.4.3）および binutils（バージョン 2.15）を使用し，gcc の最適化オプションには-O2 を指定した．また，レスポンシブプロセッサのプロセッシングユニットの動作速度は 100 MHz に設定し，オーバヘッドの計測には，レスポンシブプロセッサが内蔵する分解能 0.05 $\mu$s のタイマを用いた．

### 5.1 実用性の評価

比較対象には，実用性が高い SRP 方式を適用した Earliest Deadline First（EDF）アルゴリズム[18] を用いる．以降の表中では，これを EDF+SRP と表記する．

資源アクセス制御に関するオーバヘッドを**表 4** に示す．このうち，資源の解放時にプリエンプションが生じる場合には，解放を行うシステムコールの発行からプリエンプトしたジョブに実行が移るまでの時間をオーバヘッドとし，それ以外の場合には各操作に対応するシステムコールに要する時間をオーバヘッドとした．また，各操作のオーバヘッドは 1,000 回ずつ計測した．その際，最悪に近い状況を実現するために，各操作の直前に命令キャッシュおよびデータキャッシュをフラッシュした．

EDF アルゴリズムに SRP 方式を適用した場合には資源の獲得に失敗することがないので，各操作のオーバヘッドを単純に比較することはできない．そこで，1 回の資源アクセスに成功する場合に必要な最大オーバヘッドの総和を比較する．表 4 より，2 つの手法間で資源を獲得する際のオーバヘッドの差が最も大きいのは，SS-OP-SR アルゴリズムで Down 手続きを用いる場合である．また，資源の解放時のオーバヘッドの差はプリエンプションが生じる場合に最も大きい．したがって，資源アクセスごとの最大オーバヘッドの増加は $(26.20 + 67.85) - (21.35 + 66.40) = 6.30\,\mu$s（約 7%）にとどまることが分かる．

**表 4** 資源アクセス制御に関するオーバヘッド
Table 4 Overheads of resource access control.

| 手法 | 操作 | 最大値 |
|---|---|---|
| SS-OP-SR | TryDown（成功） | 24.80 |
| | TryDown（失敗） | 19.80 |
| | Down（成功） | 26.20 |
| | Down（失敗） | 38.25 |
| | Up（プリエンプションなし） | 21.80 |
| | Up（プリエンプションあり） | 67.85 |
| EDF+SRP | 獲得（成功） | 21.35 |
| | 解放（プリエンプションなし） | 21.95 |
| | 解放（プリエンプションあり） | 66.40 |

($\mu$s)

**表 5** RT-Frontier カーネルのコードサイズ
Table 5 Sizes of RT-Frontier kernel.

| 手法 | text | data | bss | 合計 |
|---|---|---|---|---|
| SS-OP-SR | 26,640 | 424 | 264 | 27,328 |
| SS-OP | 25,968 | 416 | 264 | 26,648 |
| EDF+SRP | 24,604 | 424 | 256 | 25,284 |
| EDF | 24,068 | 416 | 256 | 24,740 |

(byte)

次に，カーネルのコードサイズを**表 5** に示す．コードサイズに関しては，増加量を相対的に評価するために，SS-OP アルゴリズムと資源アクセス制御を行わない EDF アルゴリズムを実装したカーネルに対する計測も行った．

表 5 より，SS-OP-SR アルゴリズムの SS-OP アルゴリズムに対するコードサイズの増加量は 27,328 − 26,648 = 680 バイト（約 2.6%）である．これは，EDF アルゴリズムに SRP 方式を追加したときの増加量である 25,284 − 24,740 = 544 バイト（約 2.2%）に匹敵する．コードサイズの増加量が少なく抑えられた理由は，ジョブに割り当てた時間の管理と付加部分の中断に関する処理が SS-OP-SR アルゴリズムと SS-OP アルゴリズムで共通するためである．また，SS-OP-SR アルゴリズムによる増加量が SRP 方式による増加量よりも大きいのは，SS-OP-SR アルゴリズムが 2 通りの資源要求手続きを持つことおよび資源要求を拒否する場合があることによる．

### 5.2 実時間性の検証

実時間性を確認するために，Nyström らにより示されたセンサデータを処理するシステム[19] を仮想した検証実験を行った．

#### 5.2.1 実 験 方 法

システムにはセンサからデータを読み出すタスク，読み出されたデータから二次的な情報を抽出するタスク，さらにこの抽出された情報を利用するアプリケーショ

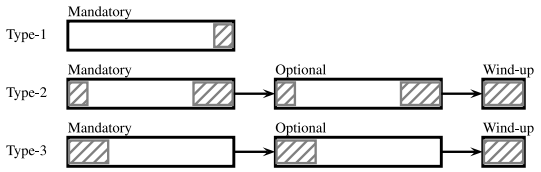


**図 12** タスクタイプの構造
Fig. 12 Structures of task types.

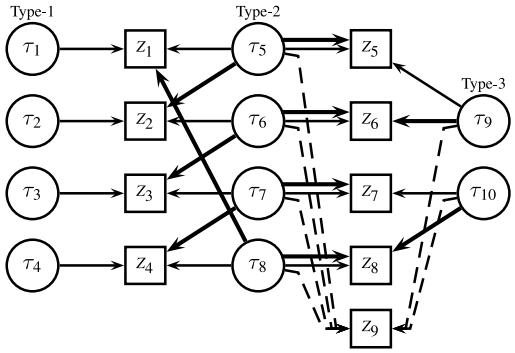


**図 13** タスクの資源制約
Fig. 13 Resource constraints of tasks.

ンタスクが存在する．本論文では，これらのタスクタイプを順に Type-1，Type-2，Type-3 とする．**図 12** にタスクタイプの構造を示す．各部分の最初または最後にある斜線は資源の占有を表す．また，SS-OP-SR アルゴリズムを用いる場合，資源要求はすべて Down 手続きを用いて行う．

実験では，タスクの数を 10，資源の数を 9 とした．**図 13** にタスクの資源制約を示す．タスクから資源へ向かう細い実線の矢印は必須部分におけるアクセスを表す．同様に，太い実線の矢印および破線の矢印はそれぞれ付加部分および終端部分におけるアクセスを表す．同一部分において，2 つの資源 $Z_i$，$Z_j$ にアクセスする場合，$i < j$ であれば，資源 $Z_i$ をアクセスした後に資源 $Z_j$ をアクセスする．また，すべての資源は 1 ユニットずつ存在し，いかなる資源アクセスも別の資源アクセスを内包することはない．さらに，すべての資源アクセスは中断できないとする．

資源の占有時間およびタスクの時間属性に関するパラメータを**表 6** および **表 7** に示す．このうち，範囲により表したものは，その範囲の一様分布から値を選択したことを意味する．また，表 7 における必須部分と終端部分の最悪実行時間はそれぞれの実際の実行時間と等しいとする．さらに，過負荷時に生じるオーバランの影響を評価するために，必須部分の最悪実行時間と付加部分の実行時間をそれぞれパラメータ $\alpha$ と $\beta$ を用いて表した．実験に用いたパラメータ $\alpha$，$\beta$ とシステム負荷の最低値 $u_M$，期待値 $u_E$ の関係を**表 8** に示す．

**表 6** 資源の占有時間
Table 6 Access duration of resources.

| | $Z_1, Z_2, Z_3, Z_4$ | $Z_5, Z_6, Z_7, Z_8$ | $Z_9$ |
|---|---|---|---|
| 占有時間 | [500, 1000] | [1000, 2000] | 1000 |

($\mu$s)

**表 7** タスクの時間属性
Table 7 Time attributes of tasks.

| | $\tau_1, \tau_2, \tau_3, \tau_4$ (Type-1) | $\tau_5, \tau_6, \tau_7, \tau_8$ (Type-2) | $\tau_9, \tau_{10}$ (Type-3) |
|---|---|---|---|
| $T_i$ | [9000, 11000] | [100000, 200000] | |
| $D_i$ | $T_i$ | | |
| $m_i$ | $0.1T_i$ | $\alpha T_i$ | |
| $w_i$ | 0 | 1000 | |
| $r_{i1}$ | 0 | | |
| $o_{ij}$ | 0 | $[(\beta - 0.01)T_i, (\beta + 0.01)T_i]$ | |

($\mu$s)

**表 8** パラメータ $\alpha$，$\beta$ とシステム負荷の最低値 $u_M$，期待値 $u_E$ の関係
Table 8 Parameters $\alpha$ and $\beta$ vs. minimal and expected system workloads $u_M$ and $u_E$.

| $\alpha$ | $\beta$ | $u_M$ | $u_E$ |
|---|---|---|---|
| 0.05 | 0.04 | 0.74 | 0.98 |
| | 0.05 | | 1.04 |
| | 0.06 | | 1.10 |
| | 0.07 | | 1.16 |
| 0.08 | 0.04 | 0.92 | 1.16 |
| | 0.05 | | 1.22 |
| | 0.06 | | 1.28 |
| | 0.07 | | 1.34 |

比較対象には，SS-OP アルゴリズムを変更して用いた．変更した SS-OP アルゴリズムは，まず $b_i$ を 0 として扱う点が SS-OP-SR アルゴリズムと異なる．また，資源アクセス制御には SRP 方式をそのまま使用し，資源アクセス中にオーバランが生じた場合でも，そのジョブがアクセスを完了するまで実行を継続させることにした．これは，実時間オペレーティングシステム RT-Frontier においてロックを用いた排他制御を行うためである．すなわち，占有されていたハードウェア資源を他のジョブに対して利用可能な状態にするためには，資源を占有していたジョブがロックを解放する必要がある．また，ロックを解放するためには，実際にハードウェアがリクエストを完了するのを待たなければならない場合がある．そのため，この場合にはオーバランしたジョブの実行を継続させることが適当である．以降では，この変更した SS-OP アルゴリズムを従来のものと区別するために MOD-SS-OP アルゴリズムと呼ぶ．

**表 9** MOD-SS-OP アルゴリズムを用いた場合の資源アクセス制御に関するオーバヘッド
Table 9 Overheads of resource access control under MOD-SS-OP algorithm.

| 手法 | 操作 | 最大値 |
|---|---|---|
| MOD-SS-OP | 獲得（成功） | 21.20 |
| | 解放（プリエンプションなし） | 21.70 |
| | 解放（プリエンプションあり） | 68.55 |

($\mu$s)

資源アクセス制御に必要なオーバヘッドは，タスクセットを生成した後に各部分の最悪実行時間に加算した．ここで，MOD-SS-OP アルゴリズムを用いた場合のオーバヘッドは，事前に表 4 と同様に計測を行い，**表 9** に示す結果を得た．表 9 より，MOD-SS-OP アルゴリズムにおける資源アクセス制御のオーバヘッドは，獲得に成功し，解放時にプリエンプションが生じる場合に最大になることが分かる．また，表 4 より，SS-OP-SR アルゴリズムにおいても，この場合にオーバヘッドが最大になることが分かる．そこで，この場合のオーバヘッドをそれぞれ資源アクセスの回数分だけ加算した．

実験では，表 8 に示した 8 つの場合ごとに，100 の異なるタスクセットをそれぞれ 10 秒間実行し，デッドラインミスを起こしたタスクセットの割合を計測した．ただし，デッドラインミスが生じた場合，その時点でタスクセットの実行を中断した．さらに，デッドラインミスの原因となるオーバランが生じる条件を推考するために，オーバランの平均頻度およびオーバラン状態にあった時間の割合を計測した．また，SS-OP-SR アルゴリズムが必要以上に付加部分の実行を制限していないことを確認するために付加部分の平均実行割合を計測した．ただし，ジョブ $J_{ij}$ が付加部分を $\hat{o}_{ij}$ だけ実行した場合，付加部分の実行割合は $\hat{o}_{ij}/o_{ij}$ になるとする．また，平均実行割合の算出においては，実験中に実行を終了しなかったジョブは考慮しないことにする．

### 5.2.2 実験結果

実験に用いたタスクセットのうち，オフラインアルゴリズムにより受諾されなかったタスクセットの割合を**表 10** に示す．これより，MOD-SS-OP アルゴリズムを用いた場合，すべてのタスクセットを受諾したことが分かる．それに対し，SS-OP-SR アルゴリズムでは $u_M$ が 0.92 の場合に，9%から 12%のタスクセットを受諾することができなかったことが分かる．これは，SS-OP-SR アルゴリズムのみが付加部分に対して資源の最大占有時間を予約することが原因である．

実際に実験においてデッドラインミスを起こしたタスクセットの割合を**表 11** に示す．MOD-SS-OP アルゴリズムを用いた実験ではデッドラインミスを生じた場合が存在するのに対し，SS-OP-SR アルゴリズムを用いた実験ではすべての場合にデッドラインを満たすことが確認できた．MOD-SS-OP アルゴリズムのデッドラインミスの原因は，いずれの場合にもオーバランであると考えられる．まず，$u_M$ が 0.74 である場合，MOD-SS-OP アルゴリズムを用いた実験では，システム負荷の期待値 $u_E$ が 0.98 と 1.04 の場合にまったくデッドラインミスが生じていないのに対し，$u_E$ が 1.10 と 1.16 の場合には 10%以上のタスクセットにおいてデッドラインミスが生じている．これは，システムが過負荷になるにつれて多くのオーバランを生じるようになったためであると考えられる．それに対し，$u_M$ が 0.92 である場合には，すべての $u_E$ の値に対してデッドラインミスを起こしたタスクセットが存在している．また，その割合の変化には $u_M$ が 0.74 の場合と逆の傾向が見られる．すなわち，$u_M$ が 0.92 の場合には，$u_E$ が小さい場合に多くのデッドラインミスが生じている．これは，$u_M$ の値が大きい場合，$u_E$ の値にかかわらず多くの付加部分が中断されているためであると考えられる．定常的な過負荷状態にシステムがある場合，実行時間の長い付加部分における資源アクセスの方が中断される可能性が低い．そのため，$u_E$ の値が大きい方がオーバランによるデッドラインミスも生じにくいと考えられる．

表 10 と表 11 を総合すると，MOD-SS-OP アルゴ

**表 10** 受諾されなかったタスクセットの割合
Table 10 Ratios of task sets that were not accepted.

| $u_M$ | $u_E$ | MOD-SS-OP | SS-OP-SR |
|---|---|---|---|
| 0.74 | 0.98 | 0 | 0 |
| | 1.04 | 0 | 0 |
| | 1.10 | 0 | 0 |
| | 1.16 | 0 | 0 |
| 0.92 | 1.16 | 0 | 0.12 |
| | 1.22 | 0 | 0.09 |
| | 1.28 | 0 | 0.11 |
| | 1.34 | 0 | 0.11 |

**表 11** デッドラインミスを起こしたタスクセットの割合
Table 11 Ratios of task sets that caused a deadline miss.

| $u_M$ | $u_E$ | MOD-SS-OP | SS-OP-SR |
|---|---|---|---|
| 0.74 | 0.98 | 0 | 0 |
| | 1.04 | 0 | 0 |
| | 1.10 | 0.12 | 0 |
| | 1.16 | 0.13 | 0 |
| 0.92 | 1.16 | 0.54 | 0 |
| | 1.22 | 0.48 | 0 |
| | 1.28 | 0.42 | 0 |
| | 1.34 | 0.38 | 0 |

**表 12** オーバランの平均頻度
Table 12 Average overrun rates.

| $u_M$ | $u_E$ | MOD-SS-OP | SS-OP-SR |
|---|---|---|---|
| 0.74 | 0.98 | 0.19 | 0 |
| | 1.04 | 43.99 | 0 |
| | 1.10 | 77.76 | 0 |
| | 1.16 | 95.96 | 0 |
| 0.92 | 1.16 | 190.53 | 0 |
| | 1.22 | 174.40 | 0 |
| | 1.28 | 166.43 | 0 |
| | 1.34 | 166.84 | 0 |

(times/10s)

リズムにおいてデッドラインミスを起こしたタスクセットの割合は，SS-OP-SR アルゴリズムにおいて受諾されなかったタスクセットの割合よりも大きい．したがって，実時間性を満たしたスケジューリングが可能であったタスクセットの割合は SS-OP-SR の方が高い．ゆえに，SS-OP-SR アルゴリズムの方が資源制約を持つタスクセットの実時間性を満たす能力が高いといえる．

次に，オーバランの平均頻度を**表 12** に示す．表 12 より，オーバランは MOD-SS-OP アルゴリズムを用いた実験においてのみ生じていることが分かる．また，オーバランの平均頻度の変化は表 11 と同様の傾向を示している．すなわち，オーバランの平均頻度は，$u_M$ が 0.74 の場合，システム負荷の期待値 $u_E$ が大きい場合に高くなっている．一方，$u_M$ が 0.92 の場合，オーバランの平均頻度は，$u_E$ 大きい場合に低くなる傾向にある．これは，表 11 に対する考察でも述べたように，$u_M$ が 0.74 の場合には $u_E$ が大きいほど中断される付加部分が多いためであり，$u_M$ が 0.92 の場合には $u_E$ の値が大きいほど資源アクセスが中断される可能性が低いことによると考えられる．

表 11 と表 12 を総合すると，$u_M$ が 0.74，$u_E$ が 1.04 の場合にはオーバランが生じているのにデッドラインミスは生じていないことが分かる．これは，実験にあたって，資源アクセス制御に必要なオーバヘッドを最悪の場合を基に予約したためであると考えられる．すなわち，実際の資源アクセス制御を最悪の場合よりも小さいオーバヘッドで完了した場合，未消費の時間をオーバランしたジョブの実行に使用することができるようになるため，デッドラインミスを起こすことがなかったと考えられる．

**図 14** に，MOD-SS-OP アルゴリズムを用いた実験において，$u_M$ が 0.74，$u_E$ が 1.04 以上の場合にオーバラン状態にあった時間の割合を示す．$u_M$ が 0.74 の場合を対象としたのは，デッドラインミスが生じる場合と生じない場合を容易に比較することができるためである．図 14 から，$u_E$ が同一ならば，スラックのプロセッサバンド幅 $U_S$ が小さい場合にオーバラン状態にあった時間の平均割合が高いことが分かる．また，スラックのプロセッサバンド幅 $U_S$ が同一ならば，$u_E$ が大きい場合，すなわち付加部分の実行時間が長い場合にオーバラン状態にあった時間の平均割合が高いことが分かる．これらの結果は表 11 および表 12 に対する考察を支持する．すなわち，付加部分の中断頻度が上昇するとオーバランの頻度が上昇し，その結果としてオーバラン状態にあった時間の割合が高くなり，デッドラインミスの割合が増加したと考えられる．

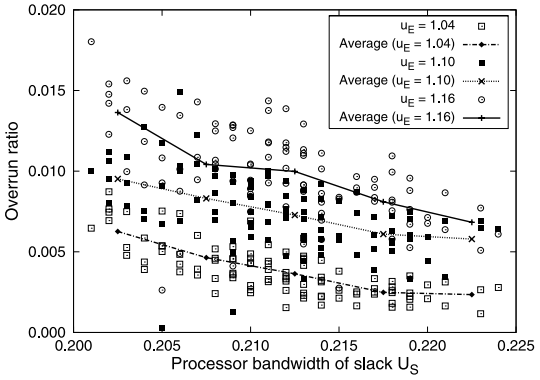


**図 14** MOD-SS-OP アルゴリズムを用いた実験においてオーバラン状態にあった時間の割合（$u_M = 0.74$）
Fig. 14 Ratios of the length of overrun to the length of the experiment under MOD-SS-OP algorithm ($u_M = 0.74$).

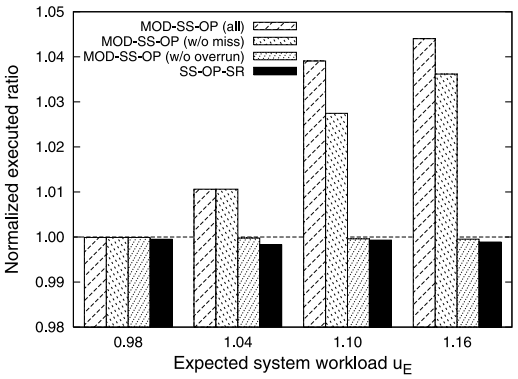


**図 15** 資源制約がない場合により正規化した付加部分の平均実行割合（$u_M = 0.74$）
Fig. 15 Average ratios of executed optional parts normalized by the ratios obtained in experiments without any resource constraints ($u_M = 0.74$).

**図 15** に，同じく $u_M$ が 0.74 である実験において計測した付加部分の平均実行割合を，資源制約がない場合により正規化して示す．資源制約がない場合に

より正規化したのは，MOD-SS-OP アルゴリズムと SS-OP-SR アルゴリズムが同一のスケジュールを生成することが理由である．平均実行割合は 3 種類の異なる方針に基づいて算出した．まず，図中に SS-OP-SR および MOD-SS-OP (all) として示した値は，実験に使用したすべてのタスクセットを対象として算出した平均実行割合である．すなわち，MOD-SS-OP (all) にはデッドラインミスを生じたスケジュールにおける付加部分の実行割合が含まれる．次に，MOD-SS-OP (w/o miss) により示した値はデッドラインミスを起こさなかったタスクセットのみを対象として算出した平均実行割合である．したがって，このとき，オーバランによる実行も，デッドラインミスが生じない限り，有効であると考える．最後に，MOD-SS-OP (w/o overrun) により示した値はデッドラインミスを起こさなかったタスクセットのみを対象とし，オーバラン状態において付加部分の実行に使用された時間を除いて算出した平均実行割合である．この場合，スケジューラにより割り当てられた時間のみを使用して実行した付加部分のみを有効な実行と考える．

図 15 より，$u_E$ が 0.98 のとき，すべての平均実行割合がほぼ 1 になっている．これは，表 12 より，MOD-SS-OP アルゴリズムにおいてもほとんどオーバランが生じていないためであると考えられる．それに対し，$u_E$ が 1.04 以上のとき，MOD-SS-OP (all) と MOD-SS-OP (w/o miss) の値が 1 を超えている．これは，MOD-SS-OP (w/o miss) においても，スケジューラが割り当てた時間よりも多くの時間を付加部分の実行に使用したことを意味する．前述のように，これが可能であるのは，本実験において資源アクセス制御のオーバヘッドを最悪の場合を基に予約したためであると考えられる．すなわち，実際の資源アクセス制御が予約した時間よりも短い時間内に完了した場合，この未消費の時間をオーバランした付加部分の実行に使用すればデッドラインミスを起こすことがない．一方，MOD-SS-OP (w/o overrun) の値が 1 を下回っているのは，資源制約によりジョブに割当て可能なスラックが減少したためであると考えられる．また，オーバラン状態における実行を除外して算出した MOD-SS-OP (w/o overrun) の値が 1 を下回ったことから，MOD-SS-OP (all) の値が 1 を超えていた第 1 の理由はオーバランであったことと考えられる．

SS-OP-SR アルゴリズムと MOD-SS-OP アルゴリズムを比較すると，図 15 において SS-OP-SR はいずれの場合にも MOD-SS-OP (w/o overrun) を下回っている．これは，Down 手続きにより付加部分が中断されたことによると考えられる．すなわち，SS-OP-SR アルゴリズムを用いた場合，割り当てられたスラックをすべて消費せずに終了するジョブが存在する．また，このジョブが終了した時点で他に実行可能なジョブが存在しなければ，その未消費のスラックはシステムをアイドル状態にすることで消費されてしまう．したがって，タスクセットが MOD-SS-OP アルゴリズムを用いてもスケジュール可能である場合，MOD-SS-OP アルゴリズムの方が SS-OP-SR アルゴリズムより高いパフォーマンスを達成可能であるといえる．ただし，MOD-SS-OP アルゴリズムでは，オーバランによるスケジュールへの影響を制御することができない．また，SS-OP-SR アルゴリズムの正規化した平均実行割合はつねに 0.99 を超えており，SS-OP-SR アルゴリズムが必要以上に付加部分の実行を制限しているとはいえない．したがって，確実に実時間性を満たす必要があるシステムでは SS-OP-SR アルゴリズムの方が有用であるといえる．

以上の結果より，資源制約により生じるオーバランを回避することで実時間性を保証する SS-OP-SR アルゴリズムの有効性を確認することができた．

## 6. むすび

本論文では，実時間システムにおいて，拡張インプリサイスタスクが共有資源へアクセス可能な SS-OP-SR アルゴリズムを提案した．本アルゴリズムは，資源制約を持つ拡張インプリサイスタスクを用いた実験において，付加部分の中断が必要な過負荷状態に陥った場合にも，デッドラインミスを 0%に抑えることができた．また，その資源アクセス制御による最悪時のオーバヘッドは，EDF アルゴリズムに SRP 方式を適用した場合と比較して，約 7%の増加にとどまることを示した．さらに，拡張インプリサイスタスクの実行機構を持つ RT-Frontier カーネルに対するコードサイズの増加は約 2.6%にとどまることを示した．

本研究で提案した SS-OP-SR アルゴリズムを用いることにより，実行時間が変動する場合にもプロセッサの利用率を低下させることなく，資源制約を持つ処理の時間制約を満たすことが可能になる．また，本アルゴリズムは実行時間の予測が不正確な実時間システムにおいても有用であると考えられる．そのため，環境に不確定要素を多く含む動的な実時間システムを構築する際に特に有効であると考えられる．

今後は，より広範囲のアプリケーションを対象とするために，周期インプリサイスタスクと非周期インプリサイスタスクの両方による資源共有を可能にする検

討を行う．また，インプリサイスタスクが生成する結果の品質を予測可能にすることで，システム設計者が結果の品質を制御可能にする．

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

## 参 考 文 献

1) Aydin, H., Melhem, R., Mossé, D. and Mejía-Alvarez, P.: Optimal Reward-Based Scheduling for Periodic Real-Time Tasks, *IEEE Trans. Comput.*, Vol.50, No.2, pp.111–130 (2001).
2) Baker, T.P.: Stack-Based Scheduling of Realtime Processes, *Real-Time Systems*, Vol.3, No.1, pp.67–99 (1991).
3) Balbastre, P., Ripoll, I. and Crespo, A.: Schedulability Analysis of Window-Constrained Execution Time Tasks for Real-Time Control, *Proc. 14th Euromicro Conference on Real-Time Systems*, pp.11–18 (2002).
4) Baruah, S. and Hickey, M.: Competitive On-Line Scheduling of Imprecise Computations, *IEEE Trans. Comput.*, Vol.47, No.9, pp.1027–1032 (1998).
5) Chung, J.-Y., Liu, J.W.S. and Lin, K.-J.: Scheduling Periodic Jobs That Allow Imprecise Results, *IEEE Trans. Comput.*, Vol.39, No.9, pp.1156–1174 (1990).
6) GNU Project. http://www.gnu.org
7) Han, S., Park, M. and Cho, Y.: On the Real-Time Job Management in Dynamic Priority Scheduling under the Stack Resource Policy, *Proc. ISCA 18th International Conference*, pp.308–311 (2003).
8) Hansen, E.A. and Zilberstein, S.: Monitoring and Control of Anytime Algorithms: A Dynamic Programming Approach, *Artificial Intelligence*, Vol.126, pp.139–157 (2001).
9) Horsch, M.C. and Poole, D.: An Anytime Algorithm for Decision Making under Uncertainty, *Proc. 14th Conference on Uncertainty in Artificial Intelligence*, pp.246–255 (1998).
10) Huang, J., Stankovic, J.A., Ramamritham, K. and Towsley, D.: On Using Priority Inheritance in Real-Time Databases, *Real-Time Systems*, Vol.4, No.3, pp.243–268 (1992).
11) Hull, D., Feng, W. and Liu, J.-S.: Enhancing the Performance and Dependability of Real-Time Systems, *Proc. IEEE International Computer Performance and Dependability Symposium*, pp.174–182 (1995).
12) Humphrey, M. and Stankovic, J.: Predictable Threads for Dynamic, Hard Real-Time Environments, *IEEE Trans. Parallel and Distributed Systems*, Vol.10, No.3, pp.281–295 (1999).
13) Kobayashi, H. and Yamasaki, N.: An Integrated Approach for Implementing Imprecise Computations, *IEICE Trans. Inf. Syst.*, Vol.E86-D, No.10, pp.2040–2048 (2003).
14) Kobayashi, H. and Yamasaki, N.: RT-Frontier: A Real-Time Operating System for Practical Imprecise Computation, *Proc. 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp.255–264 (2004).
15) Kopetz, H., Damm, A., Koza, C., Mulazzani, M., Schwabl, W., Senft, C. and Zainlinger, R.: Distributed Fault-Tolerant Real-Time Systems: The MARS Approach, *IEEE Micro*, Vol.9, No.1, pp.25–40 (1989).
16) Kuo, T.-W., Liang, M.-C. and Shu, L.: Abort-Oriented Concurrency Control for Real-Time Databases, *IEEE Trans. Comput.*, Vol.50, No.7, pp.660–673 (2001).
17) Lin, K., Natarajan, S. and Liu, J.-S.: Imprecise Results: Utilizing Partial Computations in Real-Time Systems, *Proc. IEEE 8th Real-Time Systems Symposium*, pp.210–217 (1987).
18) Liu, C.L. and Layland, J.W.: Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, *Journal of the Association for Computing Machinery*, Vol.20, No.1, pp.46–61 (1973).
19) Nyström, D., Tešanović, A., Norström, C., Hansson, J. and Bånkestad, N.-E.: Data Management Issues in Vehicle Control Systems: A Case Study, *Proc. 14th Euromicro Conference on Real-Time Systems*, pp.249–256 (2002).
20) Punnekkat, S., Burns, A. and Davis, R.: Analysis of Checkpointing for Real-Time Systems, *Real-Time Systems*, Vol.20, pp.83–102 (2001).
21) Shih, W.-K. and Liu, J.W.S.: On-Line Scheduling of Imprecise Computations to Minimize Error, *SIAM Journal on Computing*, Vol.25, No.5, pp.1105–1121 (1996).
22) Shu, L., Young, M. and Rajkumar, R.: An Abort Ceiling Protocol for Controlling Priority Inversion, Technical Report SERC-TR-157-P, Purdue University (1994).
23) Stankovic, J.A., Spuri, M., Ramamritham, K. and Buttazzo, G.C.: *Deadline Scheduling for Real-Time Systems*, Kluwer Academic Publishers (1998).
24) Wegener, J. and Mueller, F.: A Comparison of Static Analysis and Evolutionary Testing for the Verification of Timing Constraints, *Real-Time Systems*, Vol.21, pp.241–268 (2001).

25) Zhang, Y. and Chakrabarty, K.: Dynamic Adaptation for Fault Tolerance and Power Management in Embedded Real-Time Systems, *ACM Trans. Embedded Computing Systems*, Vol.3, No.2, pp.336–360 (2004).
26) Zheng, Q. and Shin, K.G.: On the Ability of Establishing Real-Time Channels in Point-to-Point Packet-Switched Networks, *IEEE Trans. Comm.*, Vol.42, No.2/3/4, pp.1096–1105 (1994).
27) 山﨑信行：並列分散リアルタイム制御用レスポンシブプロセッサ，日本ロボット学会誌，Vol.19, No.3, pp.352–361 (2001).

(平成 17 年 4 月 28 日受付)
(平成 17 年 9 月 19 日採録)

**小林　秀典**（学生会員）

1977 年生．2000 年慶應義塾大学理工学部情報工学科卒業．2002 年同大学大学院理工学研究科開放環境科学専攻修士課程修了．現在，同博士課程に在籍．リアルタイムシステム，オペレーティングシステム等の研究に従事．

**山﨑　信行**（正会員）

1966 年生．1991 年慶應義塾大学理工学部物理学科卒業．1996 年同大学大学院理工学研究科計算機科学専攻博士課程修了．博士（工学）．同年電子技術総合研究所入所．1998 年 10 月慶應義塾大学理工学部情報工学科助手．同専任講師を経て 2004 年 4 月より同助教授．現在，産業技術総合研究所特別研究員を兼務．並列分散処理，リアルタイムシステム，システム LSI，ロボティクス等の研究に従事．