情報処理学会論文誌 Vol. 52 No. 8 2365–2377 (Aug. 2011)

Rate Monotonicに基づく拡張インプリサイスタスク用リアルタイムスケジューリング

千代浩之[†1] 武田瑛[†2]
船岡健司[†2] 山﨑信行[†1]

本論文では，低ジッタと高スケジュール可能性を達成するために，準固定優先度スケジューリングを提案する．準固定優先度スケジューリングは，終端部分という第2の必須部分を有する拡張インプリサイスタスクの各々の部分を固定優先度でスケジュールする．また，本論文では，Rate Monotonic（RM）を基調とした準固定優先度スケジューリングアルゴリズム Rate Monotonic with Wind-up Part（RMWP）と RMWP++を提案する．スケジュール可能性解析では，RMWP は RM でスケジュール可能なタスクセットは必ずスケジュール可能であることを証明する．さらに，RMWP++は，タスクの実際実行時間に依存せず，最短周期タスクのジッタを 0 に抑制可能であることを証明する．シミュレーションによる評価結果では，準固定優先度スケジューリングは固定優先度スケジューリングと同程度のスケジュール成功率を発揮するだけでなく，従来のスケジューリングよりジッタを抑制したことを示した．

Real-time Scheduling Based on Rate Monotonic for Extended Imprecise Tasks[⋆1]

Hiroyuki Chishiro,[†1] Akira Takeda,[†2]
Kenji Funaoka[†2] and Nobuyuki Yamasaki[†1]

This paper proposes semi-fixed-priority scheduling to achieve both low-jitter and high schedulability. Semi-fixed-priority scheduling schedules the part of each extended imprecise task, which has a wind-up part as a second mandatory part, by fixed-priority. This paper also proposes two novel semi-fixed-priority scheduling algorithms based on Rate Monotonic (RM), called Rate Monotonic with Wind-up Part (RMWP) and RMWP++. The schedulability analysis proves that one task set is feasible by RMWP if the task set is feasible by RM. In addition, we prove that the shortest period task in RMWP++ has zero-jitter, regardless of its actual case execution time. Simulation results show that semi-fixed-priority scheduling has approximately the same success ratio as fixed-priority scheduling and lower jitter than existing scheduling.

1. 序　　論

組み込みリアルタイムシステムは，実生活の至るところで用いられているため，必要不可欠になりつつある．そのアプリケーション例として，自動車，携帯電話，ゲーム，ロボット等があげられる．特に，自律移動ロボット[1),2)] は動作する環境に応じてタスクの実際実行時間（ACET: Actual Case Execution Time）が最悪実行時間（WCET: Worst Case Execution Time）より短くなる傾向がある．高精度な動作を実現するためには，タスクのリアルタイム性を保証するだけでなく，余った CPU 時間を用いてタスクの精度を向上させる必要がある．したがって，Liu らが提案した一般計算モデル[3)] ではなく Lin らが提案したインプリサイス計算モデル[4)] に着目する．

図 1 に示すように，インプリサイス計算モデルは，1 つのタスク内に必須部分（mandatory part）と付加部分（optional part）を持つ．必須部分はリアルタイム性を保証しなければならない部分であり，必須部分が完了した場合，最低限の許容可能な精度（reward）である a を持つ結果を生成する．付加部分はリアルタイム性を保証しない部分であり，必須部分が生成した結果の精度を向上させる．インプリサイス計算モデルでは，付加部分の精度を向上させた後に，その結果を出力する処理のリアルタイム性を保証できないという問題がある．そこで，我々は結果を出力する処理に該当する第 2 の必須部分である終端部分（wind-up part）を有する拡張インプリサイス計算モデル[5),6)] を導入する．拡張インプリサイス計算モデルを用いたスケジューリングアルゴリズムには Mandatory-First with Wind-up Part（M-FWP）[5)–7)] があるが，Earliest Deadline First（EDF）[3)] に基づく動的優先度スケジューリングアルゴリズムのため，最短周期タスクのジッタが高くなってしまう[8)]．自律移動ロ

†1 慶應義塾大学
Keio University
†2 株式会社東芝
Toshiba Corporation

⋆1 This paper is based on and extends the following paper. Semi-Fixed-Priority Scheduling: New Priority Assignment Policy for Practical Imprecise Computation, *Proc. 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.339–348 (2010).

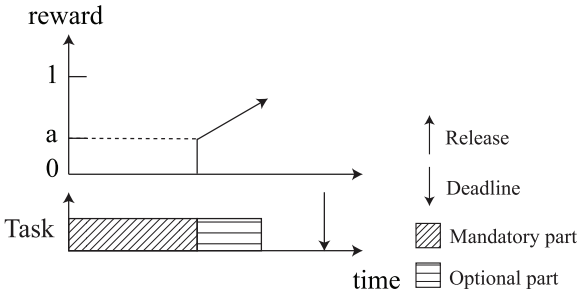


図 1　インプリサイス計算モデル
Fig. 1　Imprecise computation model.

ボットが持つタスクセットにおいて，最短周期タスクは主にモータ制御タスクである．このタスクのジッタを低く保つことは高精度な制御を実現するうえで非常に重要である．したがって，自律移動ロボットでは，M-FWP を用いることは望ましくない．これに対して，固定優先度スケジューリングアルゴリズムである Rate Monotonic（RM）[3] では，EDF より最短周期タスクのジッタが低いという利点があるが[8]，付加部分のオーバランが原因で，終端部分のデッドラインミスが発生してしまう．すなわち，RM は拡張インプリサイス計算モデルに適応できない．

本論文では，低ジッタと高スケジュール可能性を達成するために，準固定優先度スケジューリングを提案する．準固定優先度スケジューリングは，終端部分という第 2 の必須部分を有する拡張インプリサイスタスクの各々の部分を固定優先度でスケジュールする．また，RM を基調とした準固定優先度スケジューリングアルゴリズム Rate Monotonic with Wind-up Part（RMWP）と RMWP++アルゴリズムを提案する．スケジュール可能性解析では，RMWP は，RM でスケジュール可能なタスクセットは必ずスケジュール可能であることを証明する．RMWP++は，タスクの実際実行時間が最悪実行時間より短い場合においても，最短周期タスクのジッタを 0 に抑制可能であることを証明する．シミュレーションによる評価結果では，準固定優先度スケジューリングは固定優先度スケジューリングと同程度のスケジュール成功率を発揮するだけでなく，従来のスケジューリングよりジッタを抑制したことを示した．

本論文の貢献は，拡張インプリサイスタスクを低ジッタでスケジュールする準固定優先度スケジューリングおよび準固定優先度スケジューリングアルゴリズム RMWP と RMWP++を提案したことである．これらにより自律移動ロボットを高精度に制御可能になることを期待する．

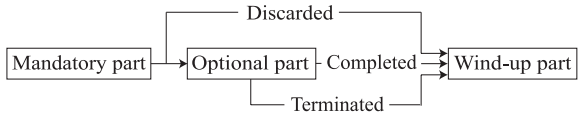


図 2　拡張インプリサイス計算モデル
Fig. 2　Extended imprecise computation model.

本論文の構成を以下に示す．2 章では，本論文で仮定するシステムモデルと用語の定義を述べる．3 章では，リアルタイムスケジューリングアルゴリズムに関する関連研究およびそれらの問題点について述べる．4 章では，準固定優先度スケジューリングおよび準固定優先度スケジューリングアルゴリズム RMWP と RMWP++について述べる．5 章では，シミュレーションによる評価結果を述べ，最後に 6 章で結論を述べる．

2. システムモデル

本論文の対象とする拡張インプリサイス計算モデル[5),6)] を**図 2** に示す．拡張インプリサイス計算モデルは，従来のインプリサイス計算モデル[4)] に第 2 の必須部分である終端部分を追加したモデルである．拡張インプリサイス計算モデルは，従来のインプリサイス計算モデルにおける付加部分の中断または完了処理のリアルタイム性を保証可能にする．

本論文では，システムが持つプロセッサ数を 1 個とする．そして，n 個の拡張インプリサイスタスクから構成されるタスクセット $\Gamma = \{\tau_i(T_i, D_i, OD_i, m_i, o_i, w_i), i = 1, 2, ..., n\}$ がシステムに与えられる．ここで，T_i は周期，D_i は相対デッドライン，OD_i は相対付加デッドライン，m_i は必須部分の最悪実行時間，o_i は付加部分の要求実行時間，w_i は終端部分の最悪実行時間を表す．タスク τ_i の j 番目のインスタンスをジョブ $\tau_{i,j}$ と呼ぶ．付加部分の要求実行時間 o_i は，ジョブごとに変動し，その上限は未知とする．ただし，RM のような一般計算モデル用のスケジューリングアルゴリズムで拡張インプリサイスタスクをスケジュールする場合，付加部分のオーバランが原因で終端部分のデッドラインミスが発生してしまうので，OD_i と o_i を 0 に設定し，必須部分と終端部分を連続して実行する．タスクの実際実行時間は，拡張インプリサイス計算モデルの各々の部分を実行開始する時刻で最悪実行時間解析[9)] により解析可能と仮定する．また，タスクの周期 T_i は相対デッドライン D_i と等しい．タスク τ_i の CPU 利用率 $U_i = (m_i + w_i)/T_i$ とする．CPU 利用率 U_i に付加部分の要求実行時間を含まない理由は，付加部分は，タスクセットのスケジュールの成否とは

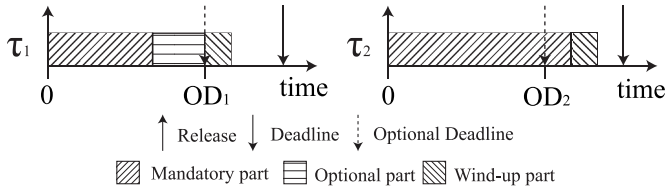


図 3　付加デッドライン
Fig. 3　Optional deadline.

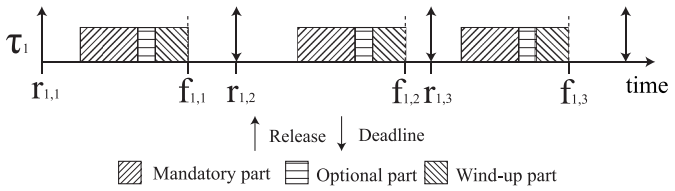


図 4　相対完了ジッタ
Fig. 4　Relative finishing jitter.

無関係だからである．システム全体の CPU 利用率 $U = \sum_{i=1}^{n} U_i$ とする．タスクは，周期の短い順に整列されているものとする．つまり，$T_1 \leq T_2 \leq ... \leq T_n$ が成り立つ．

準固定優先度スケジューリング固有のパラメータであるタスク τ_i の付加デッドライン OD_i は，付加部分が実行可能な期限および終端部分の実行開始時刻を表す．付加デッドライン以降では付加部分を実行できない．必須部分が完了していれば終端部分がデッドラインミスを発生しない時刻の範囲内で，付加部分の実行割合を向上させるために，可能な限り遅い時刻に付加デッドラインを設定する．**図 3** にタスク τ_1 と τ_2 の付加デッドライン OD_1 と OD_2 を示す．実線の上矢印は必須部分のリリース，実線の下矢印は終端部分のデッドライン，破線の下矢印は付加デッドラインを表す．タスク τ_1 は付加デッドライン OD_1 以降では付加部分を実行せず，終端部分を実行する．これに対して，タスク τ_2 は付加デッドライン OD_2 以降でも必須部分を実行する．タスク τ_2 のように付加デッドライン以降で必須部分を実行する場合，付加デッドライン時刻ではなく必須部分の完了時に終端部分を実行開始する．

本論文におけるジッタの定義は，タスクの相対完了ジッタ（RFJ: Relative Finishing Jitter）[10] とする．RFJ は 2 つの連続したジョブの完了時刻の最大偏差である．ジョブ $\tau_{i,j}$ のリリース時刻 $r_{i,j}$ と完了時刻 $f_{i,j}$ を用いて，RFJ を下式に示す．

$$RFJ_i = \max_j |(f_{i,j+1} - r_{i,j+1}) - (f_{i,j} - r_{i,j})|$$

図 4 の例を用いて，タスク τ_1 の RFJ を算出する．この場合，τ_1 の RFJ は $|(f_{1,2} - r_{1,2}) - (f_{1,1} - r_{1,1})|$ と $|(f_{1,3} - r_{1,3}) - (f_{1,2} - r_{1,2})|$ の最大値になる．自律移動ロボットでは，最短周期タスク τ_1 は主にモータ制御タスクなので，タスクセット全体の RFJ だけではなく τ_1 の RFJ のみを計測することは，高精度な制御を実現するための指標として重要である．したがって，タスク τ_1 の RFJ を Shortest Period Jitter（SPJ）と定義する．

3. 関 連 研 究

Liu らが提案した一般計算モデル[3] を対象としたスケジューリングアルゴリズムとして，RM[3] と EDF[3] がある．これらのスケジューリングアルゴリズムはタスクの実際実行時間が最悪実行時間より短い場合を考慮していない．自律移動ロボットのようにタスクの実行時間が大きく変動するシステムでは，余った CPU 時間が無駄になってしまう．

従来のインプリサイス計算モデル[4] は，タスクの実際実行時間が最悪実行時間より短い場合，余った CPU 時間を付加部分に利用可能である．タスクの最悪実行時間は実際実行時間より長く見積もってしまうので[9]，従来のインプリサイス計算モデルは一般計算モデルより実用的である．従来のインプリサイス計算モデルを用いたスケジューリングアルゴリズムとして，Optimization with Least-Utilization（OPT-LU）[11] と Mandatory-First Earliest Deadline（M-FED）[12] がある．OPT-LU は付加部分の最悪実行時間が既知であることを仮定しているので，本論文の対象とする付加部分の要求実行時間が未知である自律移動ロボットに適応できない．M-FED は EDF に基づく動的優先度スケジューリングアルゴリズムなので，SPJ が高くなってしまう[8]．

拡張インプリサイス計算モデル[5),6)] は，従来のインプリサイス計算モデルに中断または完了処理のリアルタイム性を保証する部分である終端部分を追加したモデルである．拡張インプリサイス計算モデル用のスケジューリングアルゴリズムとして，M-FED を拡張インプリサイス計算モデルに適応させた M-FWP[5),6)] がある．M-FWP は必須部分の完了したとき，付加部分の割当て可能時間を動的に算出する．付加部分の割当て可能時間がある場合，付加部分を実行する．これに対して，付加部分の割当て可能時間がない場合，終端部分を実行する．したがって，付加部分の割当て可能時間の有無に依存して，終端部分の完了時刻が変動するので，最短周期タスクのジッタである SPJ だけでなく，他のタスクの RFJ が高く

なってしまう．

拡張インプリサイスタスクを低ジッタでスケジュールするため，RM にタスクをサブタスクに分割する手法である Periodic Transformation Technique（PTT）[13] を適応させる手法がある．この手法はタスクの周期と最悪実行時間を等しい割合で分割するケースには適応可能であるが，異なる割合でタスクをサブタスクに分割するケースには適応不可能である．たとえば，1 つの拡張インプリサイスタスクを周期が 10 で最悪実行時間が 4 の 2 つのサブタスク（必須部分と終端部分に該当）に分割した場合，サブタスクの周期が 5 で最悪実行時間が 2 になる．しかしながら，必須部分と終端部分の最悪実行時間が等しいとは限らないので，拡張インプリサイスタスクに適応できたとはいえない．さらに，終端部分は付加部分の実行割合を向上させるために，付加部分のオーバランにより終端部分のデッドラインミスを発生しない範囲で，可能な限り遅く実行する要求がある．残念ながら，RM に PTT を適応させる手法では，付加部分の実行割合が過度に低くなってしまう可能性がある．したがって，RM と M-FWP の長所を組み合わせたスケジューリング手法が要求される．

4. 準固定優先度スケジューリング

準固定優先度スケジューリングは，拡張インプリサイスタスクの各々の部分の優先度を固定するが，必須部分から付加部分または付加部分から終端部分に実行部分を遷移する際に，優先度を変更する．拡張インプリサイスタスクの各々の部分の優先度を固定することで，準固定優先度スケジューリングは固定優先度スケジューリングで用いられている手法やスケジュール可能性解析等を応用可能になる．

準固定優先度スケジューリングは，**図 5** に示すように 1 つの拡張インプリサイスタスク τ_i を 2 つのサブタスク τ_i^m と τ_i^w に分割する．各々のサブタスク τ_i^m と τ_i^w の周期はともに T_i，最悪実行時間はそれぞれ m_i と w_i，最初のジョブの開始時刻はそれぞれ 0 と OD_i，相対デッドラインはそれぞれ D_i と $D_i - OD_i$ である．また，タスク τ_i^m と τ_i^w は同時に実行できず，実行順序は τ_i^m，τ_i^w となる．

図 6 に Liu らが提案した一般スケジューリング[3] と我々が提案する準固定優先度スケジューリングを示す．タスク τ_i の時刻 t における残り実行時間 $R_i(t)$ を，一般スケジューリングは破線，準固定優先度スケジューリングは実線で表す．付加部分の実行は準固定優先度スケジューリングのみなので省略する．RM や EDF のような一般スケジューリングでは，τ_i のリリース時刻で $R_i(t)$ に $m_i + w_i$ を設定する．これに対して，準固定優先度スケジューリングでは，τ_i のリリース時刻で $R_i(t)$ に m_i を設定する．そして，付加デッドライン時刻 OD_i で $R_i(t)$ に w_i を設定する．図 6 では，タスク τ_i の必須部分は付加デッドライン OD_i より早く完了しているが，付加デッドライン OD_i 以降で必須部分が完了した場合，必須部分が完了した時刻で $R_i(t)$ に w_i を設定する．

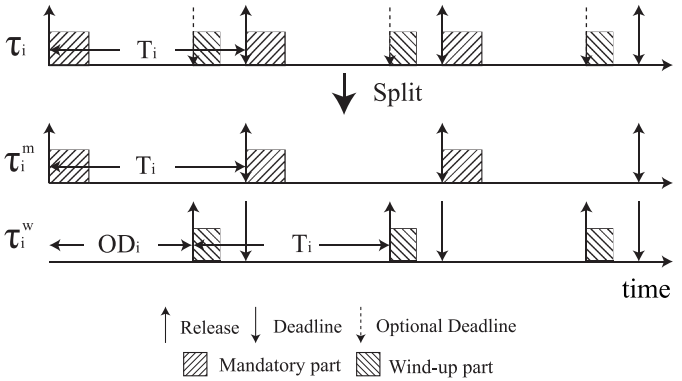


図 5　1 つの拡張インプリサイスタスクを 2 つのサブタスクに分割
Fig. 5　Split one extended imprecise task into two subtasks.

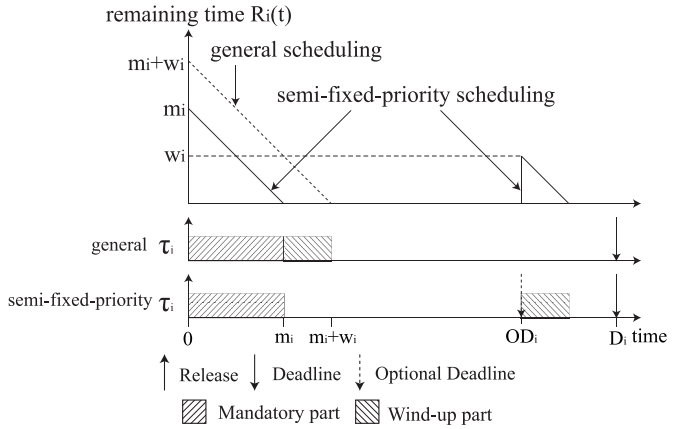


図 6　一般スケジューリングと準固定優先度スケジューリング
Fig. 6　General scheduling and semi-fixed-priority scheduling.

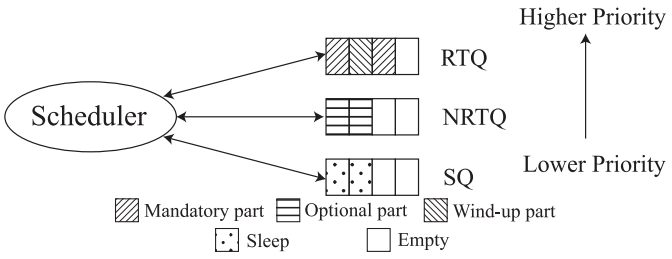


図 7　タスクキュー
Fig. 7　Task queue.

4.1　RMWP アルゴリズム

本節では，低ジッタと高スケジュール可能性を達成するために，準固定優先度スケジューリングアルゴリズム RMWP を提案する．RMWP は Real-Time Queue（RTQ）と Non-Real-Time Queue（NRTQ）および Sleep Queue（SQ）の 3 つのキューを管理する．**図 7** で示すように，RTQ では必須部分または終端部分が実行可能状態のタスク，NRTQ では付加部分が実行可能状態のタスクを管理し，それぞれのキュー内でタスクを RM でスケジュールする．ただし RTQ にあるタスクは NRTQ にあるタスクより優先度が高く，RTQ にタスクがない場合のみ NRTQ にあるタスクを実行する．あるタスク τ_i の必須部分と終端部分が同時に実行可能状態にはならない．付加デッドライン OD_i で必須部分の実行が完了しない場合，図 3 の τ_2 のように終端部分を開始せず，必須部分を続けて実行する．この場合，付加デッドラインではなく必須部分が完了した時刻で，終端部分が実行可能状態になる．タスクの付加部分または終端部分が完了した場合，そのタスクは SQ に格納される．

図 8 に RMWP アルゴリズムを示す．RMWP のスケジューラのイベントは 7 種類あり，各々の条件が成立するときに該当する処理を実行する．RMWP のスケジューラのイベントが発生する条件は以下のとおりである．(1) タスク τ_i が実行可能状態になる．(2) タスク τ_i が必須部分を完了する．(3) タスク τ_i が付加部分を完了する．(4) 付加デッドライン時刻 OD_i を経過する．(5) タスク τ_i が終端部分を完了する．(6) RTQ にタスクがある．(7) RTQ にタスクがなく，NRTQ にタスクがある．

4.1.1　RMWP の付加デッドライン

付加デッドラインは，必須部分が完了していれば終端部分がデッドラインミスを発生しない時刻の範囲内で，付加部分の実行割合を向上させるために可能な限り遅い時刻に設定される．付加デッドラインにより終端部分の実行可能範囲を制限することでジッタを抑制する．

(1)	When task τ_i is ready, set remaining time $R_i(t)$ to m_i, dequeue τ_i from SQ, and enqueue τ_i to RTQ. If task τ_i is the highest priority task in RTQ, preempt the current task.
(2)	When task τ_i completes its mandatory part:
(a)	If optional deadline OD_i expired, set remaining time $R_i(t)$ to w_i.
(b)	Otherwise set remaining time $R_i(t)$ to o_i, dequeue τ_i from RTQ and enqueue τ_i to NRTQ. If there are one or multiple tasks in RTQ or NRTQ which have higher priority than task τ_i, preempt τ_i.
(3)	When task τ_i completes its optional part, dequeue τ_i from NRTQ and enqueue τ_i to SQ.
(4)	When optional deadline OD_i expires:
(a)	If task τ_i is in RTQ and does not complete its mandatory part, do nothing.
(b)	If task τ_i is in NRTQ, terminate and dequeue τ_i from NRTQ, set remaining time $R_i(t)$ to w_i, and enqueue τ_i to RTQ. If τ_i is the highest priority task in RTQ, preempt the current task.
(c)	If task τ_i is in SQ, dequeue τ_i from SQ, set remaining time $R_i(t)$ to w_i and enqueue τ_i to RTQ.
(5)	When task τ_i completes its wind-up part, dequeue τ_i from RTQ and enqueue τ_i to SQ.
(6)	When there are one or multiple tasks in RTQ, perform RM in RTQ.
(7)	When there is no task in RTQ and there are one or multiples tasks in NRTQ, perform RM in NRTQ.

図 8　RMWP アルゴリズム
Fig. 8　RMWP algorithm.

付加デッドライン以降で必須部分を実行した場合，終端部分がデッドラインミスを発生する可能性がある．しかしながら，図 3 の τ_2 のようにデッドラインミスを発生しない場合もある．タスク τ_k の付加デッドラインを設定するために，τ_k が τ_k より高優先度のタスク τ_i $(i < k)$ に干渉される最悪干渉時間 I_k^i を求める．

定理 1（最悪干渉時間）．タスク τ_k が τ_k より高優先度のタスク τ_i（$i < k$）に干渉される最悪干渉時間 I_k^i は多くとも下式である．

$$I_k^i = \left\lceil \frac{T_k}{T_i} \right\rceil (m_i + w_i) \tag{1}$$

証明．干渉時間が最悪になる場合は，タスク τ_i の付加デッドラインが 0 の場合である．この場合，RMWP は RM と同じスケジュールを生成する．**図 9** に最悪干渉時間のケースを示す．この場合，干渉時間は式 (1) になる．また，式 (1) より干渉時間が多くなることは

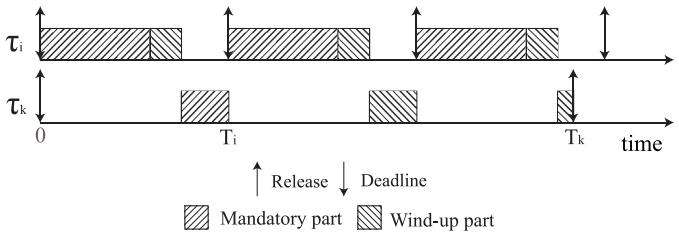


図 9　最悪干渉時間のケース
Fig. 9　Case of worst case interference time.

ない．□

そして，定理 1 を用いて，付加デッドラインを以下の定理に示す．

定理 2（付加デッドライン）．タスク τ_k の付加デッドライン OD_k が下式の場合，τ_k の必須部分が付加デッドライン OD_k までに完了していれば，τ_k の終端部分はデッドラインミスを発生しない．

$$OD_k = D_k - w_k - \sum_{i=1}^{k-1} I_k^i \tag{2}$$

証明．式 (2) のタスク τ_k の付加デッドライン OD_k は，相対デッドライン D_k から終端部分の最悪実行時間 w_k と定理 1 の最悪干渉時間 I_k^i の総和を減算する．したがって，τ_k の必須部分が付加デッドライン OD_k までに完了していれば，τ_k の終端部分がデッドラインミスを発生することはない．□

ここで，M-FWP のような動的優先度スケジューリングアルゴリズムで付加デッドラインを静的に算出するためには，最悪干渉時間を算出しなければならない．しかしながら，M-FWP はジョブごとに優先度が変動するので，何番目のジョブで干渉時間が最悪になるのかを解析することは困難である．RMWP は準固定優先度でタスクをスケジュールすることで，定理 2 より付加デッドラインを静的に算出可能にした．

4.1.2　RMWP のスケジュール可能性解析

まず，RMWP は RM でスケジュール可能なタスクセットは必ずスケジュール可能であることを以下の定理に示す．

定理 3（RMWP のスケジュール可能性）．RMWP は，タスクセット $\Gamma = \{\tau_i, i = 1, 2, ..., n\}$ が RM でスケジュール可能ならば必ずスケジュール可能である．

証明．対偶を示すことにより定理を証明する．つまり，RMWP は，タスクセット Γ がスケジュール不可能ならば RM でも必ずスケジュール不可能であることを示す．定理 2 より，タスク τ_i の付加デッドライン OD_i までに必須部分が完了していれば，終端部分がデッドラインミスを発生することはない．また，τ_i の終端部分がデッドラインミスを発生する場合は，必須部分が付加デッドライン以降で実行するときのみである．この場合，τ_i は付加部分を実行せず，必須部分と終端部分を連続して実行する．RM でも同様に，τ_i の必須部分と終端部分を連続して実行するので，必ず τ_i の終端部分がデッドラインミスを発生する．対偶が真なので定理は証明された．□

次に，RMWP のスケジュール可能上限を示す．

定理 4（RMWP のスケジュール可能上限）．タスク数が n の場合，RMWP のスケジュール可能上限 U_{lub} は下式である．

$$U_{lub} = n(2^{1/n} - 1) \tag{3}$$

証明．定理 3 より RMWP は，RM でスケジュール可能なタスクセットは必ずスケジュール可能である．全タスクの付加デッドラインが 0 の場合，定理 1 より干渉時間は最悪になり，RMWP は RM と同じスケジュールを生成する．したがって，RMWP のスケジュール可能上限は RM[3] と等しく，式 (3) となる．□

4.1.3　RMWP のスケジュール例

図 10 に RMWP と RM のスケジュール例を示す．タスクセットは $\Gamma = \{\tau_1 = (10, 10, 7, 3, 1, 3), \tau_2 = (15, 15, 1, 3, 1, 2)\}$ である．付加デッドラインは，定理 2 を用いて算出した．RMWP では，タスク τ_2 より高優先度タスク τ_1 の終端部分を必須部分の完了時刻ではなく，付加デッドライン時刻で開始する．τ_1 の必須部分の完了時刻が付加デッドライン時刻より早い場合，τ_2 は優先して実行できる．したがって，RM で発生する τ_2 のデッドラインミスを回避可能である．また，区間 $[14, 15)$ と $[26, 27)$ で，τ_1 の付加部分を実行可能である．RMWP でスケジュール失敗，RM でスケジュール成功するタスクセットは，定理 3 より存在しないだけでなく，余った CPU 時間を付加部分に利用できるので，RMWP は RM より優れている．

4.2　RMWP++アルゴリズム

自律移動ロボットでは，最短周期タスクは主にモータ制御タスクであるが，RMWP では最短周期タスクの実際実行時間が最悪実行時間より短い場合，ジッタが発生してしまう．そこで，最短周期タスクのジッタをつねに 0 にするために，RMWP を拡張した RMWP++

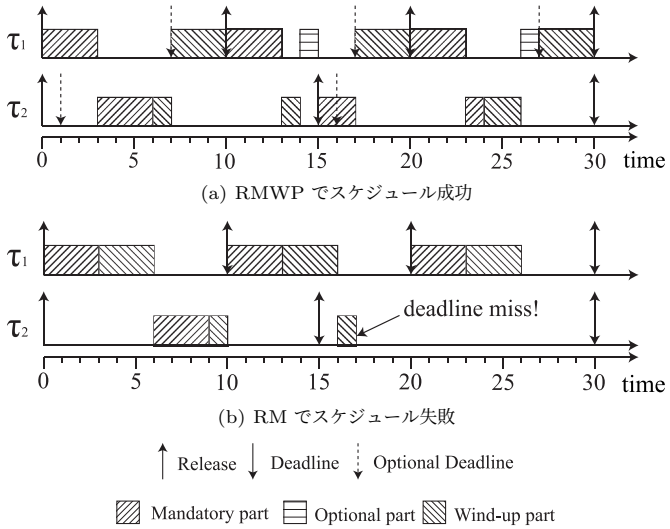


図 10　RMWP と RM のスケジュール例
Fig. 10　Example of schedule generated by RMWP and RM.

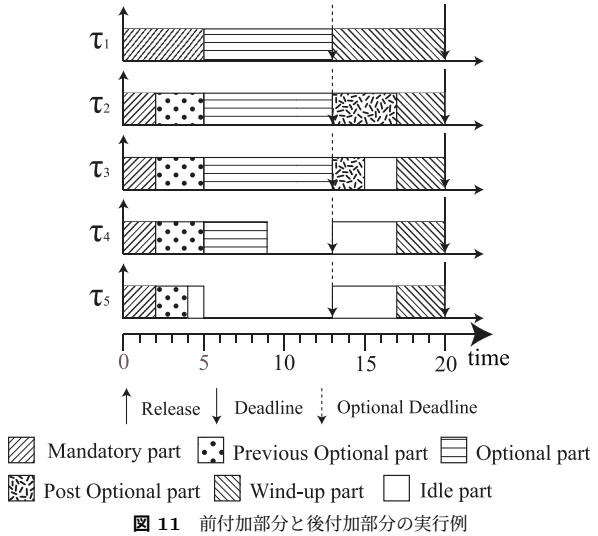


図 11　前付加部分と後付加部分の実行例
Fig. 11　Example of executing previous and post optional parts.

アルゴリズムを提案する．RMWP++は最短周期タスクのジッタを 0 に抑制するために，前付加部分（previous optional part）と後付加部分（post optional part）を図 2 の拡張インプリサイス計算モデルに追加する．タスク τ_i の前付加部分と後付加部分は，それぞれ必須部分と終端部分の実際実行時間と最悪実行時間の差分である $m_i - am_i$ と $w_i - aw_i$ を τ_i の付加部分に割り当てるための実行部分である．すなわち，タスク τ_i の前付加部分と後付加部分の実行時間をそれぞれ必須部分と終端部分の残り実行時間 $R_i(t)$ に設定する．これにより，付加部分のオーバランによる終端部分のデッドラインミスを回避する．ここで，前付加部分または後付加部分の実行区間内で，タスク τ_i の付加部分の要求実行時間 o_i を満たした場合，何もしない時間であるアイドル部分（idle part）を実行する．また，前付加部分と後付加部分の優先度は必須部分と終端部分の優先度と等しいため，これらが実行可能状態のタスクは，図 7 の RTQ に格納される．RMWP++は，全タスクの実際実行時間が最悪実行時間とつねに等しい場合，図 8 で示した RMWP と同じ動作を行うため，生成されるスケジュールは等しくなる．また，RMWP++が図 8 に追加するスケジューラのイベントは以下のとおりである．

- イベント（2）において，タスク τ_i の必須部分を完了した場合，イベント（2.a）と（2.b）の条件判定を行う前に以下を行う．タスク τ_i の必須部分の実際実行時間 am_i が最悪実行時間 m_i より短い場合，$m_i - am_i$ の時間を前付加部分に割り当てる．
- イベント（2.a）と（4.b）および（4.c）において，タスク τ_i の終端部分の最悪実行時間 w_i を残り実行時間 $R_i(t)$ に設定する前に以下を行う．タスク τ_i の実際実行時間 aw_i が最悪実行時間 w_i より短い場合，$w_i - aw_i$ の時間を後付加部分に割り当てる．

4.2.1　前付加部分と後付加部分の実行例

図 11 にタスクの前付加部分と後付加部分の実行例，**表 1** に実行する各々のタスクのパラメータを示す．ここで，タスク τ_1〜τ_5 は 1 つのタスクセットではなく，独立して実行する．タスク τ_1 は区間 $[0,5)$ で必須部分，区間 $[5,13)$ で付加部分，区間 $[13,20)$ で終端部分を実行する．タスク τ_i は $am_1 = m_1$ かつ $aw_1 = w_1$ なので，前付加部分と後付加部分を両方とも実行しない．タスク τ_2 は区間 $[0,2)$ で必須部分，区間 $[2,5)$ で前付加部分，区間 $[5,13)$ で付加部分，区間 $[13,17)$ で後付加部分を実行する．タスク τ_2 は τ_1 と比較して，必須部分と終端部分の実際実行時間が短い時間を付加部分の実行時間に費やすことが可能になる．タスク τ_3 は区間 $[0,2)$ で必須部分，区間 $[2,5)$ で前付加部分，区間 $[5,13)$ で付加部分，

表 1　各々のタスクのパラメータ
Table 1　Parameters of each task.

	T_i	D_i	OD_i	am_i	m_i	o_i	aw_i	w_i
τ_1	20	20	13	5	5	8	7	7
τ_2	20	20	13	2	5	15	3	7
τ_3	20	20	13	2	5	13	3	7
τ_4	20	20	13	2	5	7	3	7
τ_5	20	20	13	2	5	2	3	7

区間 $[13, 15)$ で後付加部分，区間 $[15, 17)$ でアイドル部分，区間 $[17, 20)$ で終端部分を実行する．タスク τ_3 は区間 $[15, 17)$ でアイドル部分を実行している理由は，時刻 15 で付加部分の要求実行時間 $o_3 = 13$ を満たしたためである．タスク τ_4 は区間 $[0, 2)$ で必須部分，区間 $[2, 5)$ で前付加部分，区間 $[5, 9)$ で付加部分，区間 $[13, 17)$ でアイドル部分，区間 [17.20) で終端部分を実行する．タスク τ_4 は時刻 9 で付加部分の要求実行時間 $o_4 = 7$ を満たしたため，付加デッドライン時刻 $OD_4 = 13$ までスリープし，区間 $[13, 17)$ でアイドル部分を実行する．タスク τ_5 は区間 $[0, 2)$ で必須部分，区間 $[2, 4)$ で前付加部分，区間 $[4, 5)$ でアイドル部分，区間 $[13, 17)$ でアイドル部分，区間 $[17, 20)$ で終端部分を実行する．タスク τ_5 は時刻 4 で付加部分の要求実行時間 $o_5 = 2$ を満たしたため，区間 $[4, 5)$ でアイドル部分を実行する．その後，付加デッドライン時刻 $OD_5 = 13$ までスリープし，区間 $[13, 17)$ でアイドル部分を実行する．

4.2.2　RMWP++に関する解析

RMWP++は必須部分と終端部分の最悪実行時間と実際実行時間の差分でそれぞれ前付加部分と後付加部分を実行することで，以下の定理を導く．

定理 5（RMWP++における SPJ）．RMWP++でスケジュールするタスクセットにおける最短周期タスク τ_1 のジッタである SPJ は，実際実行時間に依存せず，つねに 0 である．

証明．最短周期タスク τ_1 の実際実行時間が最悪実行時間と等しい場合，終端部分の相対完了時刻は等しい．また，タスク τ_1 の実際実行時間が最悪実行時間より短い場合，図 11 で示すように，タスク τ_1 の終端部分の最悪実行時間 w_1 と実際実行時間 aw_1 の差である $w_1 - aw_1$ を後付加部分に割り当てるので，τ_i の終端部分の相対完了時刻は実際実行時間に依存せず，つねに等しい．□

全タスクの実際実行時間が最悪実行時間とつねに等しい場合，RMWP++は RMWP と同じスケジュールを生成し，この場合において，タスクの干渉時間が最悪になる．したがって，定理 2 による付加デッドラインの算出と，定理 4 によるスケジュール可能上限の解析は RMWP++にも適応可能である．

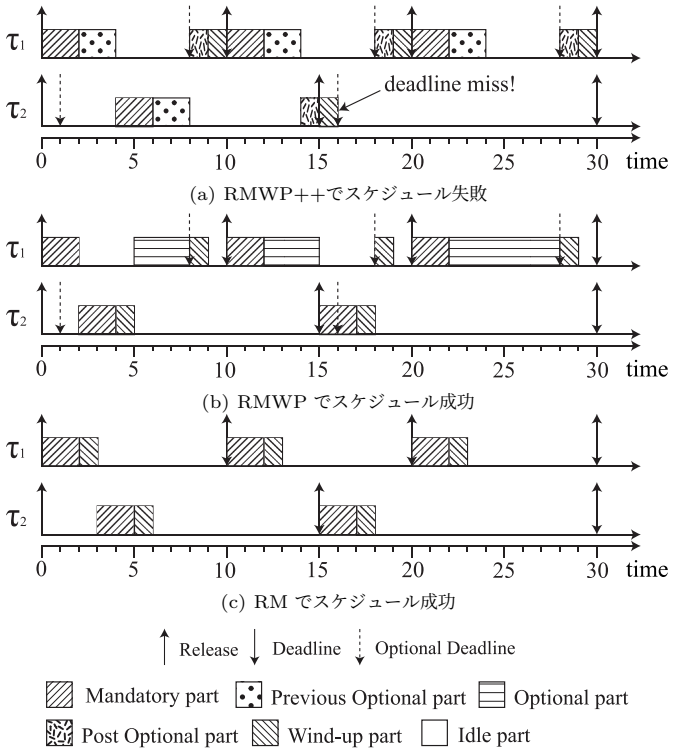


図 12　RMWP++と RMWP および RM のスケジュール例
Fig. 12　Example of schedule generated by RMWP++, RMWP and RM.

4.2.3　RMWP++のスケジュール例

図 12 に RMWP++と RMWP および RM のスケジュール例，**表 2** にスケジュールするタスクセットのパラメータを示す．RMWP++と RMWP の付加デッドラインは，定理 2 を用いて算出した．RMWP++では，タスクの必須部分と終端部分の最悪実行時間と実際実行時間の差分をそれぞれ前付加部分と後付加部分に割り当てるため，タスク τ_2 はデッドラインミスを発生してしまう．これに対して，RMWP では，タスクは前付加部分と後付加部分を実行しないため，タスク τ_2 のデッドラインミスを回避可能である．また，RM ではタス

表 2　タスクセットのパラメータ
Table 2　Parameters of taskset.

	T_i	D_i	OD_i	am_i	m_i	o_i	aw_i	w_i
τ_1	10	10	8	2	4	6	1	2
τ_2	15	15	1	2	4	10	1	2

クの実際実行時間が最悪実行時間より短いことにより，タスクの完了時刻が早まることで，スケジュールが成功する．このようなスケジュール例があるため，定理 3 は RMWP++に適応不可能であり，RM でスケジュール可能なタスクセットは必ずしも RMWP++でスケジュール可能ではない．しかしながら，定理 5 より，最短周期タスクのジッタをつねに 0 に抑制可能である．したがって，RMWP++は自律移動ロボットで高精度な動作を可能にする．

5. シミュレーション評価

本章では，準固定優先度スケジューリングの有効性を 5 種類の評価指標を用いて多角的に評価する．評価指標の詳細は 5.1 節で述べる．評価に用いるアルゴリズムは RMWP，RMWP++，RM，M-FWP である．評価には 1,000 個のタスクセットを用いる．k 番目のタスクセットのシミュレーション時間はタスクセット内の全タスクの周期の最小公倍数であるハイパーピリオド H_k とする．本論文の対象とする自律移動ロボットでは，大きく異なる周期のタスクが存在する．したがって，タスク τ_i の周期 T_i は 1 ms から 30 ms まで 1 ms おきで設定する．CPU 利用率 U_i は 0.02 から 0.25 まで 0.01 おきで設定する．システム全体の CPU 利用率 U は 0.3 から 1 まで 0.05 おきで測定する．U_i および U は必須部分と終端部分のみを含む．各々の U_i は CPU 利用率を 2 つに分割して，それぞれを必須部分と終端部分に割り当てる．タスクの実際実行時間と最悪実行時間の比率である ACET/WCET は $[0.25, 1.0]$ と 1.0 の 2 種類とする．これらとは別に，付加部分の要求実行時間 o_i の CPU 利用率 o_i/T_i を 0.1，0.2，0.3 の 3 種類を基準値として設定し，各々の基準値の ± 0.05 の範囲で，一様分布により生成する．つまり，o_i の CPU 利用率の範囲はそれぞれ $[0.05, 0.15]$，$[0.15, 0.25]$，$[0.25, 0.35]$ となる．タスクごとに o_i を設定するので，タスクセット内のタスク数が多いほど，付加部分の CPU 利用率の合計は多くなる．評価結果にはそれぞれ RMWP-10，RMWP-20，RMWP-30 のように表記する．ただし，o_i が 0 の場合，評価結果には RMWP と表記する．また，ジョブ $\tau_{i,j}$ の要求実行時間を $o_{i,j}$ と表記する．

5.1 評価指標

評価指標として用いる，必須部分と終端部分のスケジュール成功率（Success Ratio），付加部分の実行割合（Reward Ratio），コンテキストスイッチの頻度（Switch Ratio），RFJ を各々のタスクの周期で正規化した割合（RFJ Ratio），SPJ を最短周期タスク τ_1 の周期で正規化した割合（SPJ Ratio）をそれぞれ下式に示す．

$$\text{Success Ratio} = \frac{\#\text{ of successfully scheduled task sets}}{\#\text{ of scheduled task sets}}$$

$$\text{Reward Ratio} = \frac{\sum_k \sum_i \frac{T_i}{H_k} \sum_j \frac{o_{i,j}}{o_i}}{\#\text{ of tasks in successfully scheduled task sets}}$$

$$\text{Switch Ratio} = \frac{\sum_k \frac{\#\text{ of context switches}}{H_k}}{\#\text{ of successfully scheduled task sets}}$$

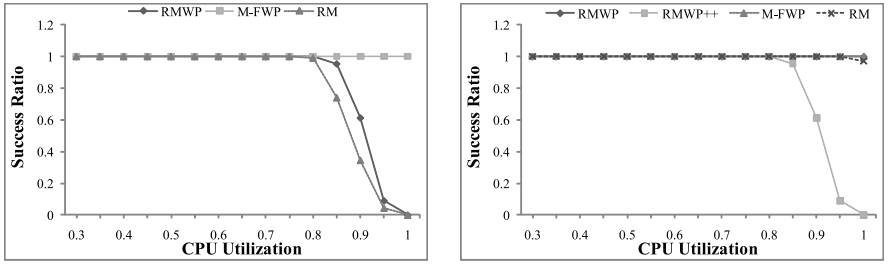
$$\text{RFJ Ratio} = \frac{\sum_k \sum_i \frac{RFJ_i}{T_i}}{\#\text{ of tasks in successfully scheduled task sets}}$$

$$\text{SPJ Ratio} = \frac{\sum_k \frac{RFJ_1}{T_1}}{\#\text{ of successfully scheduled task sets}}$$

Success Ratio が 0 の場合の CPU 利用率では，他の評価指標による評価結果はない．システム全体のオーバヘッドの評価である Switch Ratio をスケジュールが成功したタスクセットの割合とする理由は，拡張インプリサイス計算モデルは必須部分と終端部分がデッドラインミスを発生してはいけないハードリアルタイムシステムを対象としているからである．また，Reward Ratio と RFJ Ratio および SPJ Ratio に関する評価も同様に，スケジュールが成功したタスクセット内のタスク数を用いた．タスクセット全体のジッタの評価である RFJ Ratio だけでなく，SPJ Ratio を評価する理由は，本論文の対象とする自律移動ロボットで高精度な動作を実現するためには，タスクセット内で最短周期および許容するジッタが低いタスクであるモータ制御タスクのジッタが重要になるからである．また，ACET/WCET=1.0 の場合の評価結果に関して，RMWP++は RMWP と同じスケジュールを生成するので，RMWP++の評価結果は省略する．

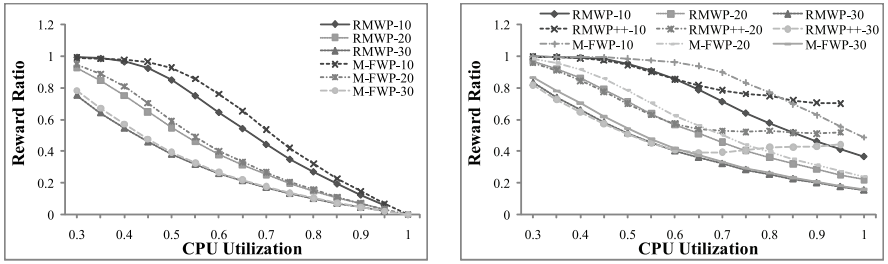
5.2 評価結果

図 13 に Success Ratio の評価結果を示す．RMWP-10 と RMWP-20 および RMWP-30 の Success Ratio は RMWP の Success Ratio，M-FWP-10 と M-FWP-20 および M-FWP-30 の Success Ratio は M-FWP の Success Ratio と同じ評価結果なので省略する．図 13 (a) の評価結果に関して，M-FWP の Success Ratio はつねに 1 を保っている．これに対して，



(a) ACET/WCET = 1.0　(b) ACET/WCET = [0.25,1.0]

図 13　Success Ratio
Fig. 13　Success ratio.

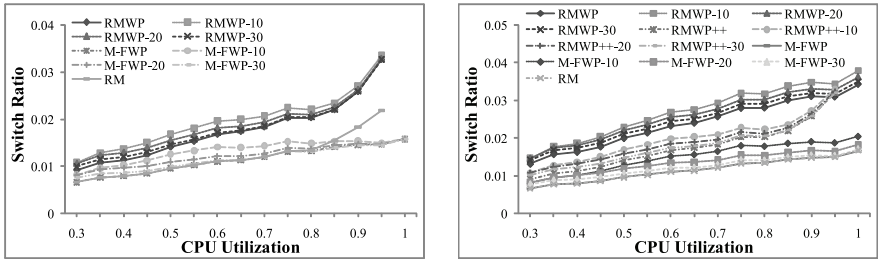


(a) ACET/WCET = 1.0　(b) ACET/WCET = [0.25,1.0]

図 14　Reward Ratio
Fig. 14　Reward ratio.

RM の Success Ratio と RMWP の Success Ratio は CPU 利用率が 0.85 以上になると低下する．RMWP の Success Ratio が RM の Success Ratio より高い理由は，図 10 のように RMWP でスケジュール成功，RM でスケジュール失敗となるタスクセットがあるからである．図 13 (b) の評価結果では，RMWP と M-FWP および RM の Success Ratio は 1 となるが，RMWP++の Success Ratio は図 13 (a) の RMWP と同様に CPU 利用率が 0.85 以上になると低下する．この理由は，RMWP++は必須部分と終端部分の最悪実行時間と実際実行時間の差分をそれぞれ前付加部分と後付加部分に割り当てるからである．

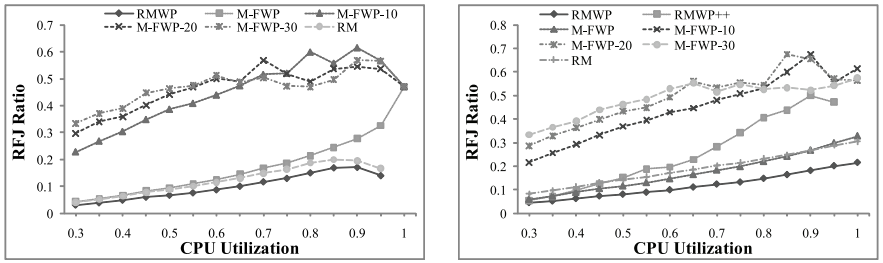
図 14 に Reward Ratio の評価結果を示す．図 14 (a) の評価結果に関して，M-FWP-10，M-FWP-20，M-FWP-30 の Reward Ratio はそれぞれ RMWP-10，RMWP-20，RMWP-30 の Reward Ratio より，それぞれ高くなっている．RMWP は付加デッドラインを静的に設定するが，M-FWP は動的に付加部分の割当て可能時間を算出するので，RMWP の Reward Ratio は M-FWP の Reward Ratio より低くなったと考えられる．図 14 (b) の評価結果に関して，RMWP++-10，RMWP++-20，RMWP++-30 の Reward Ratio はそれぞれ RMWP-10，RMWP-20，RMWP-30 の Reward Ratio より高い結果を示している．さらに，CPU 利用率が 0.85 以上では RMWP++-10，RMWP++-20，RMWP++-30 の Reward Ratio はそれぞれ M-FWP-10，M-FWP-20，M-FWP-30 の Reward Ratio より高くなる．この理由は，RMWP++は前付加部分や後付加部分により，周期の短いタスクの付加部分の実行割合が増加したからであると考えられる．



(a) ACET/WCET = 1.0　(b) ACET/WCET = [0.25,1.0]

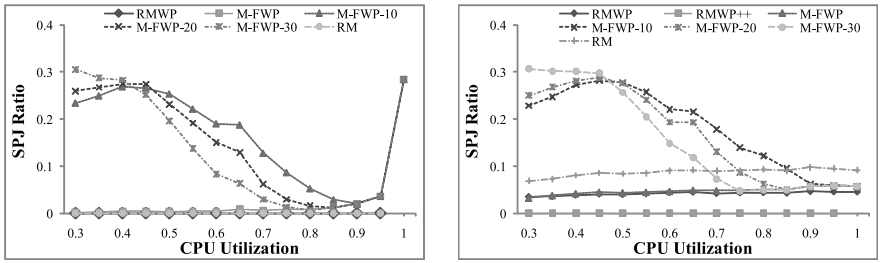
図 15　Switch Ratio
Fig. 15　Switch ratio.

図 15 に Switch Ratio の評価結果を示す．図 15 (a) に関して，RMWP-10，RMWP-20，RMWP-30 の Switch Ratio の評価結果を考慮すると，付加部分の実行割合が高いほど RMWP の Switch Ratio は低くなる．この理由は，付加部分の実行割合が高いほどコンテキストスイッチをせずに，タスクの必須部分と付加部分および終端部分を連続して実行する場合が多いからである．M-FWP も RMWP と同様に付加部分の実行割合が高いほど Switch Ratio が低くなる．付加部分を実行した場合，RMWP の Switch Ratio は RM の Switch Ratio より約 1.5 倍，M-FWP の Switch Ratio は RM の Switch Ratio より約 1.2 倍高い．したがって，拡張インプリサイス計算モデル用のスケジューリングアルゴリズムの Switch Ratio は，一般計算モデル用のスケジューリングアルゴリズムの Switch Ratio より高い傾向がある．また，RMWP の Switch Ratio は M-FWP の Switch Ratio より約 1.3 倍高い．この理由は，M-FWP ではタスクの付加部分が完了した場合，続けて終端部分を実行する



(a) ACET/WCET = 1.0　(b) ACET/WCET = [0.25,1.0]

図 16　RFJ Ratio
Fig. 16　RFJ ratio.



(a) ACET/WCET = 1.0　(b) ACET/WCET = [0.25,1.0]

図 17　SPJ Ratio
Fig. 17　SPJ ratio.

が，RMWP はタスクの付加部分が付加デッドライン時刻までに完了した場合，付加デッドライン時刻まで終端部分の実行を遅らせることで，コンテキストスイッチが M-FWP より多く発生するからである．図 15 (b) に関して，興味深いことに，RMWP++は RMWP より低い Switch Ratio を示している．この結果から，前付加部分と後付加部分は，Reward Ratio を向上させるだけでなく，Switch Ratio を低くする利点があると考えられる．

図 16 に RFJ Ratio の評価結果を示す．RMWP-10 と RMWP-20 および RMWP-30 の RFJ Ratio は RMWP の RFJ Ratio と同じ評価結果なので省略する．図 16 (a) に関して，M-FWP-10 と M-FWP-20 および M-FWP-30 の RFJ Ratio は大幅に高いが，RM の RFJ Ratio は M-FWP の RFJ Ratio より低く，RMWP の RFJ Ratio は RM の RFJ Ratio よりさらに低い．M-FWP は，3 章で述べたように，付加部分の割当て可能時間の有無により終端部分の完了時刻が変動し，RFJ Ratio が高くなってしまう．これに対して，RMWP は付加部分の割当て可能時間に依存せず，RFJ Ratio を低く維持できる．図 16 (b) に関して，RMWP++の RFJ Ratio は RMWP の SPJ Ratio より高くなっている．この理由として，RMWP++は定理 5 より最短周期タスクのジッタを 0 に抑制可能にしたことにより，他のタスクのジッタが高くなるというトレードオフが発生したと考えられる．

図 17 に SPJ Ratio の評価結果を示す．RMWP-10 と RMWP-20 および RMWP-30 の SPJ Ratio は RMWP の SPJ Ratio と同じ評価結果なので省略する．図 17 (a) に関して，RMWP と RM の SPJ Ratio はつねに 0 であるが，M-FWP の SPJ Ratio は 0 ではなく，ジッタが発生してしまう．特に，付加部分を実行する M-FWP-10 と M-FWP-20 および M-FWP-30 の SPJ Ratio は大幅に高い．M-FWP-10 と M-FWP-20 および M-FWP-30 の SPJ Ratio は，CPU 利用率が高くなるほど低くなる傾向にあるが，CPU 利用率が 1 になると，大幅に増加する．M-FWP は，CPU 利用率が低い場合は付加部分を実行するが，CPU 利用率が高い場合は付加部分をほとんど実行せず，EDF と同様のスケジュールを生成する．EDF は CPU 利用率が 1 になると SPJ Ratio が大幅に高くなる性質があるため，このような評価結果になったと考えられる．図 17 (b) に関して，定理 5 より，RMWP++の SPJ Ratio はつねに 0 を示している．これに対して，図 17 (a) と異なり，RMWP と RM の SPJ Ratio は 0 より高くなってしまう．したがって，RMWP++は最短周期タスクのジッタを実際実行時間に依存せず，つねに 0 に抑制することが可能である．

M-FWP の Success Ratio と Reward Ratio は RMWP や RMWP++の Success Ratio と Reward Ratio より高いので，スケジュール成功率と付加部分の実行割合が高いほど有効なアプリケーションに適している．これに対して，RMWP や RMWP++の RFJ Ratio と SPJ Ratio は M-FWP で付加部分を実行した場合である M-FWP-10 と M-FWP-20 および M-FWP-30 の RFJ Ratio および SPJ Ratio と比較して，大幅に低い．本論文では，モータ制御のように許容するジッタが低いタスクを持つ自律移動ロボットを対象としている．RMWP や RMWP++は付加部分の割当て可能時間によりタスクのジッタを変動させないだけでなく，M-FWP で付加部分を実行した場合である M-FWP-10 と M-FWP-20 および M-FWP-30 よりジッタが大幅に低い．したがって，準固定優先度スケジューリングにより自律移動ロボットでは安定した制御を行うことができると考えられる．また，RMWP は定理 3 より RM でスケジュール可能なタスクセットは必ずスケジュール可能であること，余った CPU 時間を付加部分に利用可能であることが優れている．さらに，RMWP++は最

短周期タスクのジッタを実際実行時間にかかわらず，0 にすることが可能である．したがって，準固定優先度スケジューリングアルゴリズム RMWP と RMWP++は M-FWP や RM より自律移動ロボットに適している．

6. 結　　論

本論文では，低ジッタと高スケジュール可能性を達成するために，準固定優先度スケジューリングおよび準固定優先度スケジューリングアルゴリズム RMWP と RMWP++を提案した．RMWP は付加デッドラインを設定することで，終端部分の実行可能範囲を制限し，ジッタを抑制した．スケジュール可能性解析では，RMWP は RM でスケジュール可能なタスクセットを必ずスケジュール可能であることを証明した．また，RMWP++は，タスクの実際実行時間が最悪実行時間より短い場合においても，最短周期タスクのジッタを 0 に抑制可能であることを証明した．この性質は，実際に自律移動ロボットを動作するうえで非常に重要である．シミュレーションによる評価結果では，RMWP は RM より高いスケジュール成功率であること，RMWP++はタスクの実際実行時間に依存せず，最短周期タスクのジッタを 0 に抑制可能であることを示した．また，そのトレードオフとして，コンテキストスイッチの頻度が，RMWP や RMWP++は RM や M-FWP より多く発生した．本論文で対象とする自律移動ロボットでは，モータ制御のように許容するジッタが低いタスクを持ち，その付加部分の実際実行時間はジョブごとに変動する．また，タスクの最悪実行時間は実際実行時間より長く見積もってしまう傾向がある．実際に自律移動ロボットで高精度な制御を達成するためには，スケジュール成功率を向上させるだけでなく，ジッタを抑制することも重要である．したがって，準固定優先度スケジューリングは自律移動ロボットに適している．

今後の課題を以下に示す．準固定優先度スケジューリングアルゴリズム RMWP と RMWP++を拡張インプリサイス計算モデル用のリアルタイム OS である RT-Frontier [14] に実装する．また，実用性を検証するために，RMWP や RMWP++のスケジューラやタスクのコンテキストスイッチのオーバヘッドを考慮した評価を行う．

謝辞　本研究の一部は科学技術振興機構 CREST によるものである．また，本研究の一部は，文部科学省グローバル COE プログラム「環境共生・安全システムデザインの先導拠点」に依るものであることを記し，謝意を表す．

参 考 文 献

1) Kanehiro, F., Hirukawa, H. and Kajita, S.: OpenHRP: Open Architecture Humanoid Robotics Platform, *The International Journal of Robotics Research*, Vol.23, No.2, pp.155–165 (2004).
2) Ahn, H.S., Beak, Y.M., Sa, I.-K., Kang, W.S., Na, J.H. and Choi, J.Y.: Design of reconfigurable heterogeneous modular architecture for service robots, *IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems*, pp.1313–1318 (2008).
3) Liu, C. and Layland, J.: Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *J. ACM*, Vol.20, No.1, pp.46–61 (1973).
4) Lin, K., Natarajan, S. and Liu, J.: Imprecise Results: Utilizing Partial Computations in Real-Time Systems, *Proc. 8th IEEE Real-Time Systems Symposium*, pp.210–217 (1987).
5) Kobayashi, H. and Yamasaki, N.: An Integrated Approach for Implementing Imprecise Computations, *IEICE trans. information and systems*, Vol.86, No.10, pp.2040–2048 (2003).
6) Kobayashi, H., Yamasaki, N. and Anzai, Y.: Scheduling Imprecise Computations with Wind-up Parts, *Proc. 18th International Conference on Computers and Their Applications*, pp.232–235 (2003).
7) Kobayashi, H.: REAL-TIME SCHEDULING OF PRACTICAL IMPRECISE TASKS UNDER TRANSIENT AND PERSISTENT OVERLOAD, Ph.D. Thesis, Keio University (2006).
8) Buttazzo, G.C.: Rate Monotonic vs. EDF: Judgment Day, *Real-Time Systems*, Vol.29, No.1, pp.5–26 (2005).
9) Yamamoto, K., Ishikawa, Y. and Matsui, T.: Portable Execution Time Analysis Method, *Proc. 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pp.267–270 (2006).
10) Buttazzo, G.C.: *HARD REAL-TIME COMPUTING SYSTEMS: Predictable Scheduling Algorithms and Applications*, 2nd edition, Springer (2004).
11) Aydin, H., Melhem, R., Mosse, D. and Mejfa-Alvarez, P.: Optimal Reward-Based Scheduling of Periodic Real-Time Tasks, *Proc. 20th IEEE Real-Time Systems Symposium*, pp.79–89 (1999).
12) Baruah, S.K. and Hickey, M.E.: Competitive On-line Scheduling of Imprecise Computations, *IEEE Trans. Comput.*, Vol.47, pp.1027–1033 (1996).
13) Sha, L., Lehoczky, J.P. and Rajkumar, R.: Solutions for Some Practical Problems in Prioritized Preemptive Scheduling, *Proc. 7th IEEE Real-Time Systems Symposium*, pp.181–191 (1986).

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).

(平成 22 年 11 月 30 日受付)
(平成 23 年 5 月 14 日採録)

千代　浩之（学生会員）

2008 年慶應義塾大学理工学部情報工学科卒業．2010 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，分散ミドルウェア等の研究に従事．

武田　瑛

2007 年慶應義塾大学理工学部情報工学科卒業．2009 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，株式会社東芝勤務．

船岡　健司（正会員）

2006 年慶應義塾大学理工学部情報工学科卒業．2008 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．2008 年株式会社東芝勤務の傍ら，同大学大学院理工学研究科開放環境科学専攻博士課程に在籍．2009 年同大学院理工学研究科開放環境科学専攻博士課程修了．博士（工学）．現在，株式会社東芝勤務．

山﨑　信行（正会員）

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，准教授）．リアルタイムシステム，プロセッサアーキテクチャ，並列分散処理，システム LSI，ロボティクス等の研究に従事．日本ロボット学会，IEEE 各会員．