Vol.2013-OS-127 No.10
Vol.2013-EMB-31 No.10
2013/12/3

オーバヘッドを考慮したシミュレータによる階層型スケジューリングアルゴリズムの評価

佐藤 祐一[1]　松原 豊[1]　高田 広章[1]

概要：分散リアルタイムシステムにおいて，個別に開発・検証されたリアルタイムアプリケーションを，単一の高性能なプロセッサに統合して動作させるための階層型スケジューリングアルゴリズムが数多く提案されている．しかしながら，これらのアルゴリズムを比較評価する研究は行われていない．本論文では，スケジューリングに特化したシミュレータを用いて，複数の階層型スケジューリングアルゴリズムを比較評価する．評価対象となる階層型スケジューリングアルゴリズムは，ARINC653，ARINC653 割込み受付拡張手法，BSS アルゴリズム，時間保護アルゴリズム，周期リソースモデルの 5 種類である．評価では，実際の車載制御アプリケーションをモデル化したものを統合の対象とし，さらに，実機で動作する場合に発生するアプリケーション切替えとタスク切替えのオーバヘッドの影響を含めてシミュレーションする．今回の評価では，ARINC653 割込み受付拡張手法のスケジューリング可能性が高く，かつアプリケーション切替え回数，タスク切替え回数が最も少ないことから，自動車制御アプリケーションの統合に最も適していることが明らかになった．

キーワード：リアルタイムシステム，階層型スケジューリングアルゴリズム，時間保護

1. はじめに

代表的な分散リアルタイムシステムの 1 つである自動車制御システムには，走行性能の向上や運転支援を目的とした数多くの機能が搭載されている．現在は，ECU（Electronic Control Units；電子制御ユニット）と，ECU 上で動作するアプリケーションをサプライヤーが一体で開発し，自動車メーカがそれらを組み合わせることで，複雑な自動車制御システムを実現していることが多い．システムに新しい機能を追加する場合，センサやアクチュエータへのインタフェースと，それらを制御するマイコンが一体となった ECU を追加する必要がある．

近年，自動車制御システムの高性能化・高機能化により，自動車 1 台あたりに搭載される ECU の数が急激に増加している．その結果，ECU 搭載数の増加に伴うハードウェアコストの増加や，自動車内に ECU を追加するためのスペースが不足するという問題が発生している．

これらの問題を解決するために，現状の機能を維持したまま，ECU 数を削減する取り組みが行われている．自動車に搭載される ECU 数を削減するためのアプローチとして，個別に開発・検証された複数のリアルタイムアプリケーションを，単一の ECU 上で動作させる（アプリケーション統合）ための手法が数多く検討されている．アプリケーションを統合した際，アプリケーションで起きた障害が他のアプリケーションに影響を与えることを防ぐために，メモリ保護と時間保護の機能をもつ RTOS を用いることが望ましい．

時間保護の機能を提供する RTOS では，その保護機能を実現するために，階層型スケジューラを使用する場合が多い．階層型スケジューラとは，アプリケーションをスケジューリングするグローバルスケジューラと，アプリケーション内のタスクをスケジューリングするローカルスケジューラを階層的に配置したスケジューリング機構である．階層型スケジューラを用いたスケジューリングアルゴリズム（階層型スケジューリングアルゴリズム）は複数提案されている [1][2][3][5]．航空機の制御システムでも実際に利用されている [1]．

本論文では，これまでに提案された複数の階層型スケジューリングアルゴリズムを，実機で動作させた際のオーバヘッドを考慮して比較評価することを目的とする．評価環境には，スケジューリングシミュレータ *schesim*[6] を使用する．理論的な解析では，スケジューリング可能となるための CPU 利用率などのパラメータが悲観的になりすぎてしまう．一方，実機を用いれば正確な評価は可能だが，

[1] 名古屋大学大学院情報科学研究科
Graduate School of Information Science, Nagoya University

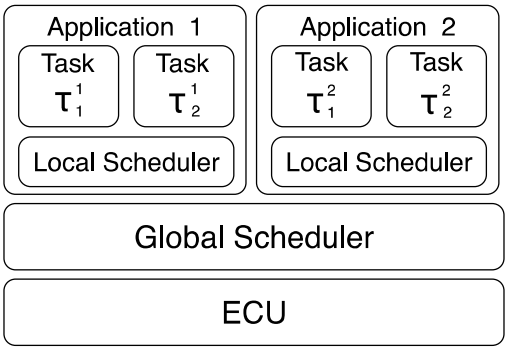

図 1 階層型スケジューラ

すべてのアルゴリズムを実装するのは難しい．そこで，ある程度の精度で，かつ実機を使用せずに動作を簡易に確認するためには，シミュレータを使用することが望ましい．本論文では，実機で発生する処理オーバヘッドの影響を含めてシミュレーションできるように *schesim* を拡張し，階層型スケジューリングアルゴリズムを比較評価する．

本論文の構成は，以下のとおりである．まず，第 2 章で本論文で想定するシステムの構成と，評価対象とする階層型スケジューリングアルゴリズムの概要，評価で考慮するオーバヘッドを説明する．第 3 章では，*schesim* の拡張方法について述べる．第 4 章では，比較評価に用いる指標を説明し，*schesim* を用いたシミュレーションによる比較評価の結果を述べる．第 5 章で，関連研究を述べる．最後に，第 6 章で，本論文のまとめと今後の展望について述べる．

2. システムモデル

2.1 システムの構成

図 1 に示すように，自動車制御システムで動作する複数のリアルタイムアプリケーションを，1 つの高性能な ECU 上に統合することを想定する．各々独立に動作するコンピュータシステム上でデッドラインを満たして動作するアプリケーションを，高性能な 1 つのコンピュータシステム（統合 ECU と呼ぶ）上で動作させる．統合 ECU はシングルプロセッサである．アプリケーション間の同期通信はないものとする．アプリケーションは複数のタスクから構成されており，各アプリケーションおよびタスクがアクセス可能なメモリ領域は MPU（Memory Protection Unit）などのアクセス制御機構により保護（メモリ保護）される．さらに，各アプリケーションが実行される時間は，階層型スケジューリングにより保護される．

2.2 スケジューリングアルゴリズム

対象とする階層型スケジューリングアルゴリズムは，以下の 5 種類である．

- ARINC653
- ARINC653 割込み受付拡張手法
- BSS（Bandwidth Sharing Server）アルゴリズム
- 時間保護アルゴリズム
- 周期リソースモデル（PRM：Periodic Resource Model）

ARINC653 とは，米国 Aeronautical Radios（ARINC）社の APplication/EXecutive（APEX）ワーキンググループが規定した航空機システム IMA（Integrated Modular Avionics）向け基盤ソフトウェアの標準規格である [1]．ARINC653 では，IMA で高い安全性，信頼性を実現するために，パーティションと呼ばれる単位で各アプリケーションが利用するメモリとプロセッサ時間を保護するための API を規定している．ARINC653 の規定する時間保護機能では，まずシステムの構築段階でシステム周期とパーティションごとの実行時間を決める．次に，システム周期内における各パーティションの実行順序と実行開始時間，実行時間を定めたスケジューリングテーブルをシステム全体で 1 つ生成する．システム動作時は，生成したスケジューリングテーブルに従ってパーテションを，順番に実行することで，プロセッサ時間を保護する．

ARINC653 割込み受付拡張手法（以降，ARINC 拡張手法と呼ぶ）は，ARINC653 をベースとし，高い応答性を要求される割込み処理だけを，実行中のパーティションの処理に優先して即座に実行することで，割込み処理の応答性を向上させる手法である．即座に実行される割込み処理は，特権割込み処理と呼ぶ．特権割込み処理によって実行が中断されたパーティションの実行時間は，特権割込み処理が動作した時間分だけ延長される．各アプリケーションのプロセッサ時間を保護するために，スケジューリングテーブルには，パーティションの合計実行時間がシステム周期の時間を超過しないように，特権割込み処理の最大時間分の余裕時間を設定する．特権割込み処理は特権モードで実行されるため，保護領域の設定を必要とせず，アプリケーションの切替え処理が必要ない．

BSS アルゴリズム [2]（以降，BSS と呼ぶ）は，グローバルスケジューリングとローカルスケジューリングの両方に EDF（Earliest Deadline First）スケジューリングを用いる場合と，ローカルスケジューリングに固定優先度ベースのスケジューリングを用いる場合がある．統合前に EDF スケジューリングによって全てのタスクがスケジューリング可能で，かつ統合後のローカルスケジューリングアルゴリズムとして EDF スケジューリングを採用している場合に限り，各アプリケーションの全てのタスクは，統合後もスケジューリング可能であることを理論的に保証する．すなわち，このアルゴリズムは，各タスクの相対デッドラインがわかれば適用できるため，多くのアプリケーションに適用できる．

時間保護アルゴリズム [3]（以降，TPA と呼ぶ）は，グローバルスケジューリングに EDF スケジューリング，ローカルスケジューリングに固定優先度ベースのケジューリン

グを用いたBSSアルゴリズムの場合は，統合後にタスクがデッドラインを満たすことは保証されないという問題を解消するために，実行中の低優先度タスクより遅い絶対デッドラインを持つ高優先度タスクの起動時刻を遅延させることで，低優先度タスクがデッドラインをミスしないよう改良された手法である．この意味で，ローカルスケジューリングアルゴリズムは厳密な固定優先度ベースのスケジューリングとは異なるが，時間保護アルゴリズムは，統合前に固定優先度ベースでスケジューリング可能であり，統合後のローカルスケジューリングアルゴリズムとしてEDFスケジューリングを採用した場合に，各アプリケーションのタスクがスケジューリング可能であることが理論的に証明されている．

周期リソースモデル [4]（以降，PRMと呼ぶ）は，アプリケーション内の各タスクがデッドラインまでに要求するプロセッサ時間（リソース）を求め，その要求を上回るリソースを周期的に供給することで，各アプリケーションのスケジューリング可能性を保証するアルゴリズムである．文献 [4] で提案されたリソースモデルは，RM（Rate Monotonic）スケジューリングまたはEDFスケジューリングを用いてスケジューリング可能であるアプリケーションに対し，統合後もそれぞれのアプリケーションのタスクがスケジューリング可能であることを保証する．

2.3 アプリケーション統合における処理オーバヘッド

リアルタイム性を保証する階層型スケジューリングアルゴリズムの提案においては，実機で発生する処理オーバヘッドは無視していることが多い．しかしながら，実機でリアルタイムシステムを動作させる場合，RTOSによるオーバヘッドが必ず発生し，その影響は無視できない．例えば，実行中の低優先度タスクが高優先度タスクによって中断される場合，実行中の低優先度タスクのコンテキストをレジスタからメモリへ書き出し，高優先度タスクのコンテキストをメモリからレジスタへ書き戻す処理（コンテキストスイッチという）が必要である．他にも，割込みの出入口処理にかかる時間や，タスクのスケジューリングにかかる時間などがオーバヘッドとして存在する．階層型スケジューリングアルゴリズムがリアルタイム性を保証することを理論的に証明する際，先述のオーバヘッドは無視されることが多い．

さらに，階層型スケジューラを用いるシステムでは，アプリケーションのメモリ保護と時間保護を実現するために，プロセッサで実行するアプリケーションを切り替えるための処理オーバヘッドが存在する．例えば，MPUの設定を切替える処理は，アプリケーションの切替えが発生するたびに行われる．

本研究では，実際のECU上でシステムを動作させたときに発生すると考えられるオーバヘッドのうち特に大きいと思われる以下のオーバヘッドを含めてシミュレーションする．

- タスクのコンテキストスイッチ
- 割込みの出入り口処理
- アプリケーション切替え

各オーバヘッドのシミュレーション方法は3章で説明する．評価で使用する各オーバヘッドの具体的な時間は4章で説明する．

3. スケジューリングシミュレータの拡張

3.1 スケジューリングシミュレータ *schesim*

schesim[6] は，スクリプト言語Rubyで実装されたスケジューリングシミュレータである．シミュレータ内部では離散的なシステム時刻を管理し，システム時刻を更新するタイミングでアプリケーションやタスクのスケジューリング，RTOSの提供するサービス（セマフォ管理など）を実行する．*schesim* では，システムに含まれる各タスクや割込みの処理をモデル化したものを入力として，システム全体のスケジューリングをシミュレートする．タスクの処理に制御構造を記述できるため，タスクの実行時間変動や，タスク間同期通信もモデル化できる．加えて，シナリオファイルによって，システム時刻を指定したAPIの実行が行えるため，非周期処理のシミュレーションも可能である．さらに，シミュレーション結果を可視化するツール [7] や，CPU利用率などの統計情報を計算するツールが用意されており，シミュレーション結果の分析を容易にできる．

3.2 オーバヘッドのシミュレーション方法

オーバヘッドを *schesim* 上でシミュレーションするために，タスク開始（再開）時，タスク終了（タスク切替えによる中断を含む）時，アプリケーション開始時，アプリケーション終了（アプリケーション切替えによる中断を含む）時に呼ばれるフック関数を追加した．具体的には，以下の4種類の関数を定義した．

1. PreTaskHook ：タスク実行開始（再開）時
2. PostTaskHook ：タスク実行終了（中断）時
3. PreAppHook ：アプリケーション実行開始時
4. PostAppHook ：アプリケーション実行終了（中断）時

1と2をタスクフック，3と4をアプリケーションフックと呼ぶ．

タスクフックとアプリケーションフックの内部で各種オーバヘッドの時間分だけシステム時刻を進めることで，オーバヘッドによる時間経過を表現できる．例えば，図2に示すように，2つのタスク τ_1，τ_2 を考える．τ_1 は低優先度，実行時間10，起動オフセット0のタスクであるとし，τ_2 は高優先度で，実行時間2，起動オフセット6のタスクであるとする．PreTaskHookおよびPostTaskHookの実行時間をそれぞれ1とする．

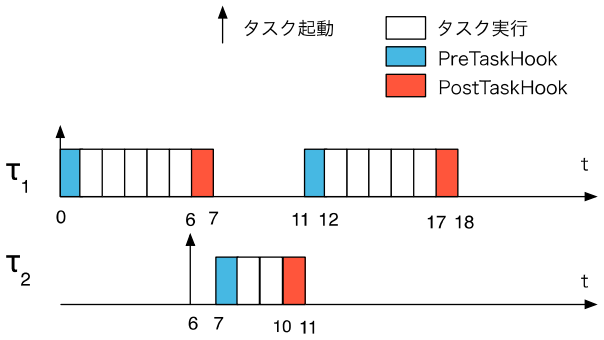

図 2 オーバヘッドを考慮したタスクスケジューリングの例

表 1 各フックで行う処理

関数	処理内容
PreTaskHook	実行するタスクのコンテキストをメモリから読み出し，レジスタへ書き込む処理，または割込みの入り口処理.
PostTaskHook	終了（中断）するタスクのコンテキストをレジスタから読み出し，メモリへ書き込む処理，または割込みの出口処理.
PreAppHook	実行するアプリケーションがアクセス可能なメモリ領域の設定を更新する処理.
PostAppHook	処理なし.

青色と赤色で示されている部分が，それぞれ PreTaskHook と PostTaskHook が実行されている時間である．システム時刻 t が 6 のとき τ_2 がリリースされるので，タスク切替えが発生するが，τ_1 の PostTaskHook が終了するまで τ_2 は実行されない．アプリケーション切替え時における PreAppHook と PostAppHook の振る舞いも，図 2 の τ_1 と τ_2 をアプリケーションに置き換えることで同様に表現できる．

2.3 節で述べたオーバヘッドと，タスクフックとアプリケーションフックで表現する処理の対応関係を，表 1 に示す．現時点では，以下の点において，実機での動作と違いがある．

- タスクの移動と再開，終了と中断を区別していない
- アプリケーション終了時における時間保護のための処理オーバヘッドを 0 としている

4. 階層型スケジューリングアルゴリズムの比較評価

4.1 評価指標

本研究では，以下の観点から階層型スケジューリングアルゴリズムの比較評価を行う．

- スケジューリング可能性
- アプリケーションの切替え回数
- タスク・割込みの切替え回数

スケジューリング可能性は，シミュレーション中にデッドラインミスが発生するかどうかで判断する．デッドラインミスがシミュレーション中に発生しなかった場合，スケジューリング可能であると判断し，シミュレーション中にデッドラインミスが 1 回でも発生した場合，スケジューリング不可能であると判断する．さらに，シミュレーション中に発生したアプリケーション切替えの回数，タスク切替えの回数を計測し，比較評価する．アプリケーション切替えとは，あるアプリケーションから別のアプリケーションに実行が切り替わることを指す．具体的には，メモリ保護領域の設定が更新された場合に，アプリケーション切り替えが発生したとみなす．タスク切替えとは，実行中のタスクまたは割込み（処理単位とよぶ）と異なる ID を持つ処理単位に実行が切り替わることを指す．

4.2 評価実験

評価には，実際に利用されている車載制御アプリケーションを使用する．今回の評価ではエンジンの制御アプリケーション（以下，EG と呼ぶ）と，モータジェネレータの制御アプリケーション（以下，MG と呼ぶ）を用いる．これらのアプリケーションを *schesim* に入力できる形にモデル化する際，EG および MG はエンジンやモータの回転数によってタスクの処理負荷とデッドラインが変動するため，処理負荷別に EG は 2 種類，MG は 3 種類のアプリケーションモデルを作成した（以降，EG の低負荷モデルを EG-L，EG の高負荷モデルを EG-H，MG の低負荷モデルを MG-L，中負荷モデルお MG-M，高負荷モデルを MG-H と呼ぶ）．

EG のアプリケーションモデルは，EG-L が 39 個のタスクと 30 個の割込みで構成され，EG-H が 33 個のタスクと 23 個の割込みで構成されている．タスクおよび割込みには周期的に実行されるものと，非周期に実行されるものがあり，タスクや割込みの中には実行時間が変動するものが含まれている．MG のアプリケーションモデルは，MG-L と MG-M が 8 個のタスクと 8 個の割込みで構成され，MG-H がタスク 8 個のタスクと 9 個の割込みで構成されている．タスクは全て周期タスクであり，タスクや割込みの中には実行時間が変動するものが含まれている．EG，MG 共に固定優先度ベースのスケジューリングによってスケジューリングされる．

各アプリケーションが統合前に動作していた ECU の性能から，統合 ECU の性能を仮定した．各アプリケーションの CPU 利用率は，EG が 33%，MG が 50%とした．各切替えのオーバヘッドは，表 2，表 3 の値を使用する．この値は，開発中のプロトタイプシステムを使って測定した値から計算した．アイドルとは，実行するタスクあるいはアプリケーションが存在せず，システムがアイドル状態であることを指す．タスク切替えのオーバヘッドは，切り替わる前のタスク（割込み）の PostTaskHook の実行時間と，

表 2 処理単位の切替えパターンとオーバヘッド

切替えパターン	時間 [us]
タスク→タスク	1.68
タスク→割込み	1.84
タスク→アイドル	0
割込み→タスク	1.84
割込み→割込み	1.84
割込み→アイドル	0
アイドル→タスク	0.84
アイドル→割込み	0.92

表 3 アプリケーション切替えパターンとオーバヘッド

切替えパターン	時間 [us]
アプケーション→アプリケーション	2.772
アプリケーション→アイドル	0
アイドル→アプリケーション	0

切り替わった後のタスク（割込み）の PreTaskHook の実行時間の合計が表 2 と対応するように *schesim* に実装した．アプリケーション切替えのオーバヘッドは，PreTaskHook の実行時間を 2.772us，PostTaskHook の実行時間を 0 として実装した．

schesim 上で各階層型スケジューリングアルゴリズムを利用して，EG と MG を統合し，*schesim* で 200000us だけ動作させた．統合プロセッサ上で動作させるアプリケーションモデルは，EG と MG から 1 種類ずつ選択し，合計 6 種類の組み合わせで評価した．シミュレーションは，オーバヘッドを考慮していない状態と，オーバヘッドを考慮した状態の両方で行った．

階層型スケジューリングアルゴリズムとして ARINC 拡張手法を選択した場合，統合対象となるアプリケーションの処理の一部を，特権割込みとして登録する必要がある．本論文では，ARINC653 によってスケジューリングを行う際にデッドラインミスの要因となったタスクや割込みを特権割込み処理とした．

4.3 評価結果

4.3.1 スケジューリング可能性

スケジューリング可能性の評価結果を表 4 と表 5 に示す．EG モデルの処理負荷によらず，同じ結果が得られたため，MG モデルの種別と評価結果をまとめた．「◯」はスケジューリング可能であったことを示し，「−」はシミュレーション中にデッドラインミスが発生し，スケジューリング不可能であったことを示している．

ARINC653 および PRM は，オーバヘッドを考慮したシミュレーションの結果，統合対象にどのようなアプリケーションモデルを選択した場合でも，スケジューリング不可能であった．BSS は MG-H と MG-L を統合に用いた場合にスケジューリング不可能となった．TPA は BSS と比べ，MG-L を統合に用いた場合にスケジューリング可能となった．全ての組み合わせがスケジューリング可能となったのは，ARINC 拡張手法のみであった．

表 4 スケジューリング可能性評価結果（オーバヘッドなし）

アルゴリズム	MG-L	MG-M	MG-H
ARINC653	◯	◯	◯
ARINC 拡張手法	◯	◯	◯
BSS	◯	◯	◯
TPA	◯	◯	◯
PRM	◯	◯	◯

表 5 スケジューリング可能性評価結果（オーバヘッドあり）

アルゴリズム	MG-L	MG-M	MG-H
ARINC653	-	-	-
ARINC 拡張手法	◯	◯	◯
BSS	-	◯	-
TPA	◯	◯	-
PRM	-	-	-

表 6 アプリケーション切替え回数

アルゴリズム		ARINC 拡張手法	BSS	TPA
回数	OH[*1]なし	1867	3327	3331
	OH[*1]あり	1867	3361	3355

4.3.2 アプリケーション切替え回数

アプリケーション切替え回数の評価結果を表 6 に示す．評価には，各アルゴリズムで EG-H と MG-M を統合した場合のシミュレーション結果を用いた．ARINC653 および PRM は，オーバヘッドを考慮したシミュレーションでスケジューリング不可能だったため，評価対象から除外した．表 6 から，ARINC 拡張手法が最も切替え回数が少なく，BSS と TPA を比較すると，オーバヘッドを考慮していない状態では BSS のほうが回数が少ないが，オーバヘッドを考慮すると TPA の方が回数が少なかった．ARINC 拡張手法はオーバヘッドを考慮したシミュレーションでも，アプリケーション切替え回数が変化しなかった．BSS および TPA は，オーバヘッドを考慮するとアプリケーション切替え回数が増加した．

4.3.3 タスク切替え回数

タスク切替え回数の評価結果を表 7 に示す．評価には，アプリケーション切替え回数の評価と同様のモデルを使用した．ARINC653 および PRM は評価対象から除外した．表 7 から，評価対象とした全ての階層型スケジューリングアルゴリズムで，オーバヘッドを考慮したシミュレーションを行った結果タスク切替え回数が増加していることがわかる．タスク切替え回数の合計は，オーバヘッドを考慮するかどうかに関わらず，ARINC 拡張手法が最も少なく，BSS が最も多かった．

[*1] OH：オーバヘッド

表 7　タスク切替え回数

アルゴリズム	OH*1	タスク→タスク	タスク→割込み	割込み→タスク	割込み→割込み	合計
ARINC 拡張手法	なし	1808	2063	515	1316	5702
	あり	1859	2167	545	1375	7003
BSS	なし	1578	3848	1032	2271	8729
	あり	1689	3879	1105	2432	9105
TPA	なし	1488	3857	1029	2250	8624
	あり	1586	3881	1106	2368	8948

表 8　シミュレーション中にシステムがアイドル状態であった時間の合計 [us]

OH*1 なし	OH*1 あり
68484	66466

4.4　考察

4.4.1　スケジューリング可能性に関する考察

MG のうち MG − L を統合に用いた場合のシミュレーションで，BSS ではスケジューリング不可能であったが，TPA ではスケジューリング可能であったのは，TPA の起動遅延機能によるものであると考えられる．PRM は，タスクセットから必要な CPU 利用率とリソース供給周期を計算するが，評価では，EG と MG の CPU 利用率を固定して実験を行ったため，スケジューリング可能となるようなパラメータが見つからなかったと考えられる．今後の課題として，CPU 利用率や統合 ECU の性能を変動させるような評価方法を検討する必要がある．

MG-H を統合に用いた場合のシミュレーションでは，TPA がスケジューリング不可能であったが，ARINC 拡張手法ではスケジューリング可能であった．これは，統合によって新たに発生したオーバヘッドによって，アプリケーションが必要とする性能が増加し，割り当てた CPU 利用率だけではスケジューリングできなくなったからだと考えられる．ARINC 拡張手法は，TPA と比べてアプリケーション切替え回数，タスク切替え回数共に少なかったことから，オーバヘッドによる影響が小さかったと考えられる．

4.4.2　切替え回数変化に関する考察

ARINC 拡張手法では，オーバヘッドを考慮するかどうかに関わらずアプリケーション切替え回数は変化しなかった．これは，システム周期を変化させなかったからである．評価に使用した ARINC 拡張手法のシステム周期は，スケジューリング可能となるシステム周期のうち，最大の値である．そのため，タスクの切替えやアプリケーションの切替えによって発生するオーバヘッドによって，アプリケーションがアイドル状態である時間，すなわちアプリケーションの中に実行可能なタスクが存在しない時間は短くなったが，システム周期には影響しなかった（表 8）．

一方，BSS または TPA を用いたシミュレーションでは，アプリケーション切替え回数は増加した．増加の原因の 1 つとして，オーバヘッドによって各タスクの実行時間が増加した結果，アプリケーションが連続して動作する時間が長くなり，オーバヘッドがない状況ではアイドル状態で起動していた割込みが，他のアプリケーション実行中に起動するようになったことが挙げられる．例えば，図 3 のようなスケジューリングを考える．図 3 は，EG のタスク τ_1^{EG} と MG のタスク τ_1^{MG}，τ_2^{MG} を，オーバヘッドを考慮しない場合と，オーバヘッドを考慮した場合のスケジューリング結果である．各タスクのパラメータは，τ_1^{EG} が起動オフセット 0，実行時間 4，相対デッドライン 25，τ_1^{MG} が起動オフセット 4，実行時間 1，相対デッドライン 6，τ_2^{MG} が起動オフセット 11，実行時間 1，相対デッドライン 6 とし，PreTaskHook，PostTaskHook，PreAppHook の実行時間を全て 1 とする．オーバヘッドを考慮しなければ，τ_1^{EG} は時刻 4 で終了するが，オーバヘッドを考慮すると，他のアプリケーションに邪魔されながら，時刻 21 まで実行が続く．アプリケーション切替えに着目すると，オーバヘッドを考慮しないスケジューリングでは，時刻 4 で EG から MG に切り替わり，時刻 11 で τ_2^{MG} が起動するが，直前で動作していたアプリケーションが MG なので，アプリケーションの切替えが発生したとはみなさない．したがって，時刻 25 までに発生するアプリケーション切り替えの回数は 1 回である．オーバヘッドを考慮した場合，τ_1^{EG} の実行が時刻 4 までに完了せず，τ_2^{MG} が起動する時刻 11 になっても完了していないことから，アプリケーション切り替え回数は 4 回になる．MG には相対デッドラインが短く，かつ周期的に発生する割込みが多く存在するため，この例のようなケースが発生することが多い．

タスク切替え回数がオーバヘッドを考慮することで増加した理由は，先述の図 3 の時刻 4 や時刻 11 のように，オーバヘッドを考慮していない状態では，アイドル状態で起動したためタスク切替えが発生しなかったが，オーバヘッドを考慮することで，直前に動作していたタスクの実行時間が長くなり，タスク切替えが発生したからであると考えられる．BSS よりも TPA の方がタスク切替え回数が少なかった理由は，起動遅延によって低優先度タスクが高優先度タスクによって中断される回数が減少したからであると考えられる．

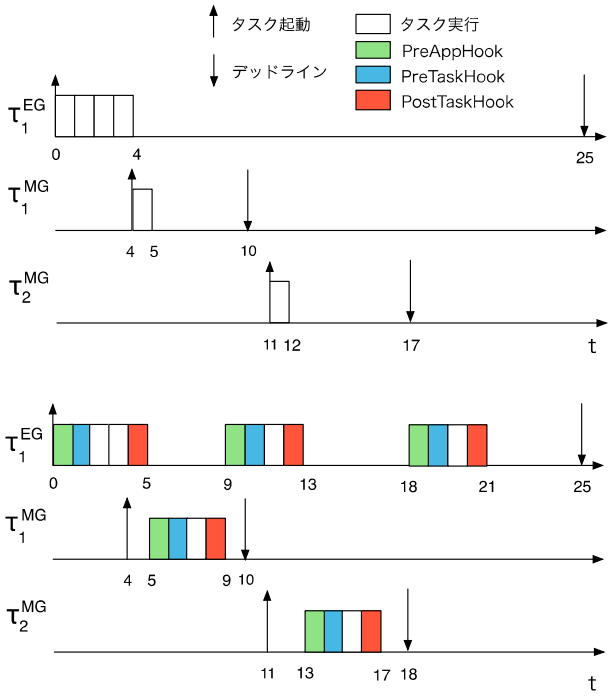

図 3 BSS（TPA）スケジューリングにおいてオーバヘッドを考慮しない場合（上）と考慮した場合（下）のアプリケーション切替え回数の変化

4.5 評価結果のまとめ

シミュレーションの結果を比較すると，ARINC 拡張手法が，統合対象となるアプリケーションの負荷が高くてもスケジューリング可能であり，アプリケーション切替え回数とタスク切替え回数の両方で，どの階層型スケジューリングアルゴリズムよりも回数が少なかった．TPA は，高い処理負荷を持つアプリケーションを統合対象とするとスケジューリング不可能となる部分はあったが，スケジューリング可能性，タスク切替え回数が BSS よりも優れていた．オーバヘッドによってタスクの実行時間が長くなった結果，タスクの切り替え回数は，全ての階層型スケジューリングアルゴリズムで統合を行った場合で増加した．

オーバヘッドが，アプリケーション統合後のスケジューリング可能性に大きな影響を与えることに加え，図 3 のように，タスクの実行時間が長くなることで，タスクの最悪応答時間が大幅に延長され，アプリケーション切替え回数やタスク切替え回数が増加することも明らかになった．

5. 関連研究

本研究では，シミュレータを用いて，階層型スケジューリングアルゴリズムの評価を行ったが，理論的な手法も提案されている，Linh T.X. らは，リアルタイムシステムに対して，オーバヘッドを考慮して分析するための手法を提案している [8]．この手法では，オーバヘッドを，タスクの実行時間に含められるものと，そうでないものに分類し，それぞれに対する計算方法と，オーバヘッドを考慮したインタフェースモデルおよびそのモデルにおけるオーバヘッドの計算方法を提案し，実機での結果と手法を用いた分析結果を比較している．比較の結果，スケジューリング可能性の解析は実際よりも悲観的な結果となったが，従来の理論的な解析よりも現実的な結果が得られた．文献 [8] では，評価にパラメータがランダムなタスクセットを使用しているが，本研究では実際の複雑なリアルタイムアプリケーションを対象とした．

アプリケーション統合を行う手法には，階層型スケジューラを用いる以外の手法も存在する．Timo Kerstan らは，仮想化によって，リアルタイムアプリケーションと，そのアプリケーションが動作している RTOS をそのまま仮想化し，統合するような仮想化システムの設計方法を提案している [9]．提案された設計方法では，個別の ECU で動作している各アプリケーションを 1 つのバーチャルマシン（VM）上に実装し，複数の VM を高性能なプロセッサ上で実行する．さらに，VM をパーティションとして，Single Time Slot Pericdic Pertitions（STSPPs）[10] と呼ばれるパーティション方式を利用し，パーティションの起動周期と統合後のシステムが動作するプロセッサの性能を適切に定めることで，デッドラインを満たすことが可能であることを確認している．

6. おわりに

本論文では，リアルタイムシステムを実機で動作させた際に発生するオーバヘッドの一部をシミュレーションできるスケジューリングシミュレータを用いて，階層型スケジューリングアルゴリズムの比較評価を行った．統合対象となるアプリケーションには，実際に利用されている車載制御アプリケーションをモデル化したものを使用した．評価では，タスクのコンテキストスイッチ，割込み出入口処理，アプリケーション切替え時に必要なメモリ保護領域設定の書き替えのオーバヘッドを考慮した．評価の結果，ARINC653 割込み受付拡張手法を採用して統合を行った場合に，オーバヘッドを考慮した状態で高い負荷のアプリケーションモデルを統合してもスケジューリング可能であり，アプリケーション切替え回数が最も少なかった．

今後は，本論文でアプリケーション切替え回数およびタスク切替え回数の評価対象から除外した ARINC653 および PRM に対する評価，今回対象としていないオーバヘッドのシミュレーション方法の検討，新たな評価指標や評価方法を加えた，アルゴリズムの比較評価を行う予定である．

参考文献

[1] Airlines Electoronic Engneering Commitee: AVIONICS APPLICATION SOFTWARE STANDARD INTERFACE, (2008).
[2] Lipari, G. and Baruah, S. ”Efficient Scheduling of Real-Time Multi-Task Applications in Dynamic Systems.

IEEE Real-Time Technology and Applications Symposium. pp.166-175 (2000).
[3] 松原豊，本田晋也，高田広章，”時間保護のためのタスク起動遅延付き階層型スケジューリングアルゴリズム．”，情報処理学会論文誌，vol.52，No.8，pp.2387-2401 (2011).
[4] Insik Shin，Insup Lee．”Periodic Resource Model for Compositional Real-Time Guarantees”．Real-Time Systems Symposium, 2003. RTSS 2003. 24th IEEE．pp.2-13 (2003).
[5] Lipari, G., Carpenter, J. and Baruah, S. ” A framework for acheving interapplication isolation in multiprogrammed, hard real-time environments”, Proc. IEEE Real-time System Symposium, pp.217226 (2000).
[6] 佐野泰正，松原豊，本田晋也，高田広章，”リアルタイムアプリケーション向けタスク処理定義可能なスケジューリングシミュレータ”，組込みシステムシンポジウム 2011 (ESS2011)．(2011).
[7] 後藤隼弐，本田晋也，長尾卓哉，高田広章：トレースログ可視化ツール TraceLogVisualizer(TLV)，コンピュータソフトウェア， Vol.27, No.4, pp.823 (2010).
[8] Linh T.X. Phan，Meng Xu，Jaewoo Lee，Insup Lee，Oleg Sokolsky．”Overhead-Aware Compositional Analysis of Real-Time Systems”．19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)．pp.237-246 (2013).
[9] Timo Kerstan, Daniel Baldin, Stefan Groesbrink．“Full virtualization of real-time systems by temporal partitioning”．OSPERT2010, pp.24-32 (2010).
[10] Mok, A.K. Xiang Feng, Deji Chen．”Resource partition for real-time systems”．Real-Time Technology and Applications Symposium, 2001, pp.75-84 (2001).