修 士 論 文

RTOS オーバヘッドの特性を考慮したスケジューリング方式の提案

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

長谷川　和輝

2015 年 9 月

修 士 論 文

RTOS オーバヘッドの特性を考慮したスケジューリング方式の提案

指導教員　田中 清史 准教授

審査委員主査　田中 清史 准教授
審査委員　金子 峰雄 教授
審査委員　井口 寧 教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

s1310056 長谷川　和輝

提出年月: 2015 年 8 月

概 要

本稿では、アプリケーションタスクに加えて、RTOS オーバヘッドとしてマルチタスクスケジューリングを行う際に引き起こされるタスクの切り替え処理を明確化し、スケジューリングの中に導入した新しいスケジューリング方式を提案する。提案するスケジーリング方式を用いることによって、タスク切り替え処理を含んだシステムに対して実行前にシステムのスケジュール可能性を確認できる。

また、本提案に用いる方法ではタスク切り替え処理を実際のシステムで稼働するより過大見積もりしている。この過大見積もりした量を正確にシステムで扱う方法とアプリケーションの応答時間向上への利用方法も提案する。

Slack として生じた過大見積もりを利用したシミューレションをすることで、アプリケーションの応答時間を計測し従来の何も行わない方法と比較する。提案手法である Slack の再利用をすることで、非周期応答時間を最大 31.56 %短縮させることができた。

目 次

第1章　はじめに

1.1　研究背景

様々なリアルタイム組込みシステム開発において、開発されるシステムは複雑性を増している。その複雑なシステム開発を補助する目的で Real-Time Operating System(RTOS) が用いられ、RTOS はシステム構築の効率化に用いられる。RTOS を使用することにより、開発者はリアルタイムシステムやマルチタスクシステムを作るためにタスク切り替え等の処理を個別に実装する必要が無くなる。しかしながら、この RTOS を使う利便性の代償として RTOS 処理は実行する際に、開発内容に直接関わっていない処理の時間であるオーバヘッドを生じる。

本研究は、RTOS によるオーバヘッドを考慮した上で、実環境でのスケジュール可能性を保証するスケジューリング方式の提案を第一の目的とする。今回の研究において特に注目する RTOS 処理は、タスクの切り替え処理である。このタスク切り替え処理には、2 つの時間的特性が見られる。

- アプリケーションタスクを起動する際に起こる周期的な処理
- アプリケーションタスクを終了する際に起こる非周期的な処理

この時間的特徴を活かし、アプリケーションタスクに加えて、タスク切り替え処理をスケジューリングに導入した新しいスケジューリング方式を提案する。この時間的特徴の差異を利用した方法を提案することで、理論と現実の差を埋めることで開発の負担が減ると考えた。

更に、システムを正しく動作させるために過剰に見積もった RTOS 処理帯域を非周期的タスクに割り当てることにより、非周期タスクの応答時間を更に短くする方法も同時に提案をする。

1.2　論文の構成

本論文は、5 章で構成される。本章では、この研究が扱う問題と方向性を述べた。続く第 2 章において、スケジューリングに関しての既存研究の成果を紹介し、既存研究の問題点に言及する。第 3 章では、本研究の提案手法に関して説明をし、第 4 章で評価の方法と結果をする。最後に第 5 章において、本論文のまとめと今後の展望を論じる。

第2章　既存の研究

この章では既存の研究による成果を説明する。参考文献として、G.C.Buttazz 著作の本 [1] を用いた。最初に周期的タスクをスケジューリングする手法を説明する。次に、非周期的要求を処理するための方法である様々なサーバ方式を紹介する。

2.1　既存のタスクスケジューリング手法

タスクを静的な優先度で扱う方法やタスクを動的な優先度で扱う方法等、様々な手法が既に多くの研究者によって提案されてきた。これらの方法の内、周期的なタスクを扱う方法と非周期的タスクを扱う方法に限定して解説をする。

2.1.1　Rate Monotonic　スケジューリング (RM)

この方法は周期的なタスクに対して、静的な優先度によって実行するタスクを決めシステムを管理する方法である。優先度は実行前に予め定められたタスクの周期によって決定され、周期がより短いタスクがより高い優先度を持ち優先的に実行される。

この方法を用いることにより、システムで用いるタスクセットが決まると事前にスケジューリングの様子を組み立てることができ、システムの挙動を予想できる。本論文では、この RM を基にして議論を進める。

2.2　非周期処理を扱う方法

ここでは、非周期要求を周期的に処理する既存の方法の紹介をする。既存手法の利点や欠点と共に、本研究の提案方法に用いることができるのかを述べる。

2.2.1　Background スケジューリング

到着した非周期的要求を、何も処理をする時がない空き時間の場合のみ実行する。この方式で割り当てられた非周期要求はシステムの上では最低優先度で実行され、非周期要求を処理する時間を設けることを保証しない。システムの負荷が慢性的に高い状況では、非周期的要求の処理が一切行われない場合も生まれてしまう。

2.2.2 Polling Server (PS)

非周期要求を周期的要求の様にみなし処理する方式の一つである。このサーバは、予め指定された周期毎に補充される容量を持つ。この方式の特徴として、サーバで処理しなければいけない次の要求が存在している間において、既に到着している非周期的要求をサーバの容量の許す限りサーバの優先度に応じて処理する。サーバで処理しなければいけない次の要求が存在しない場合は、どんなにサーバが容量を持っていてもその容量は空になる。この PS を用いた例を図 2.1 に示す。

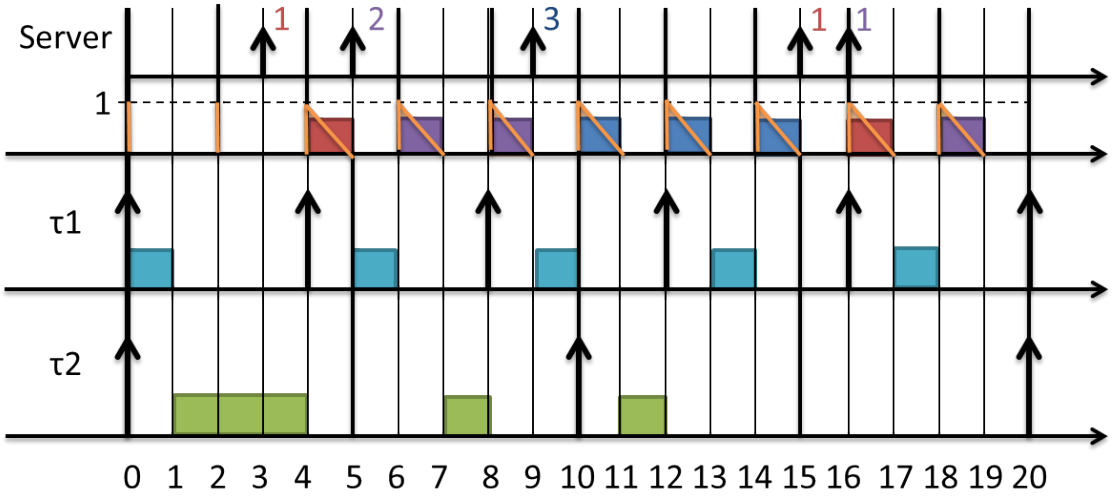

図 2.1: Polling Server の振る舞い

本研究では、周期的に発生するシステムティック上での周期的タスク要求の到着、及び開始処理をこのサーバで担当し管理する。

2.2.3 Deferrable Server (DS)

非周期要求を周期的要求の様にみなし処理する方式の一つである。このサーバは、予め指定された周期毎に補充される容量を持つ。PS と違いサーバに到着している処理が空でもサーバの容量は保つので、サーバの周期半ばに到着した処理でも容量の許す限りサーバの優先度に応じて処理することができる。しかし、この方法を用いることでスケジュール可能性を保証できる最大タスク使用率（LUB）が下がってしまう欠点が存在する。この DS の稼働する様子を図 2.2 に示す。

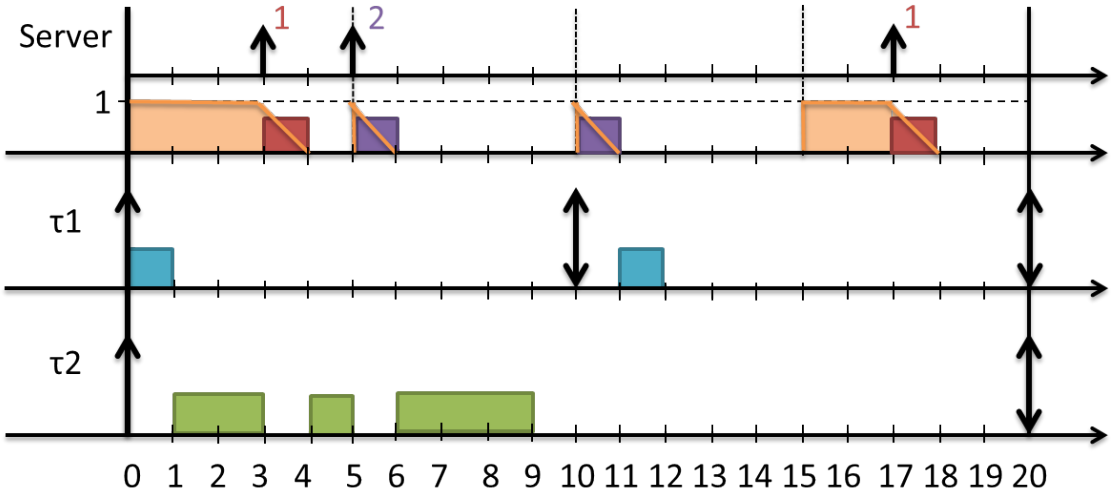

図 2.2: Deferrable Server の振る舞い

本研究では、非周期的に発生するタスク終了処理をこのサーバで管理する。

2.2.4 Priority Exchange (PE)

非周期要求を周期的要求の様にみなし処理する方式の一つであるが、DS と違い容量を先行して実行させたタスクの優先度まで下げて保存する。この方法では先に紹介した DS と違い、LUB が下がらないという利点がある。このサーバが非周期処理を処理する様子を図 2.3 にて示す。

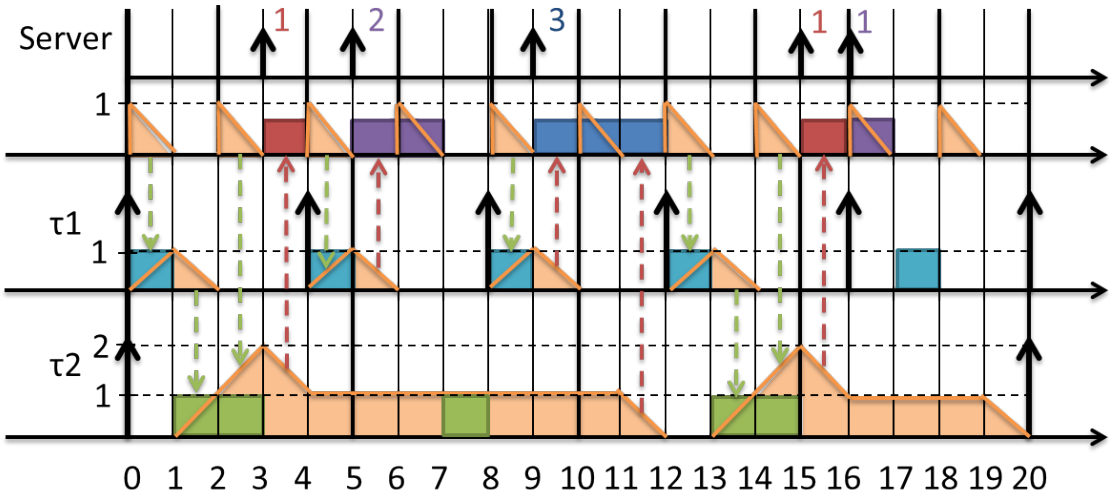

図 2.3: Priority Exchange の振る舞い

しかし、本研究において RTOS 処理は他のいかなる処理より優先して実行されなければならない処理を想定している。サーバの優先度が固定されないこの方法は、どのタスクより最優先で処理しなければならないタスク切り替え処理の用途に合わない。したがって、本研究の目的に適さなかったため用いることはしていない。

2.2.5 Sporadic Server (SS)

先に紹介した手法とは異なりサーバは周期を持つが、所定の時間に必ずサーバの容量を補充する仕組みを持たない。決まった時間の代わりに、サーバの容量を消費した時刻をシステムで把握し、その時刻から周期分の時間の先でサーバの容量を使用した分だけ補充する。この方式を用いると PE の様に優先度を変化させずに、DS の様な一定優先度を保ったサーバの容量が確保することができる。このサーバが非周期処理を処理する様子を図 2.4 にて図示した。

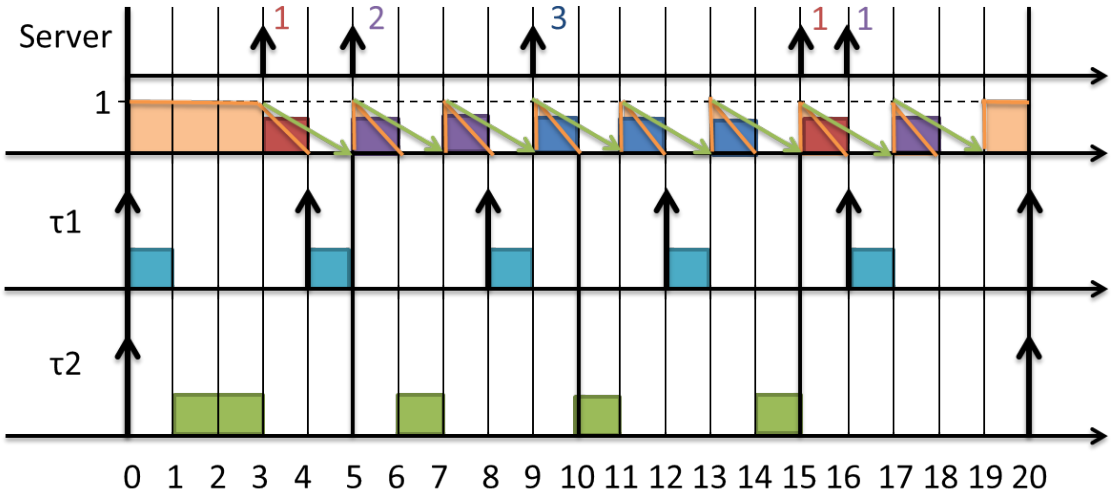

図 2.4: Sporadic Server の振る舞い

本研究において、この手法を用いることができるがこのサーバの実装には複雑性が伴う。そのため、本研究での非周期応答性の評価には、このサーバ方式は用いていない。

2.3　既存のスケジューリング手法の問題点

ここまでの既存の研究による方式はいずれも、これらのスケジューリングを行うための処理は仕組みに内包されていない。したがって、これらの研究の方式を用いてスケジュール可能性を保証しても、実際のシステムで処理を動かした場合に、稼働しているシステムがデッドラインミスを起こしスケジュール可能性が成り立たない場合がある。

この理想化した環境と実環境の差を埋める方法が求められる。本研究は、この 2 つの環境の差を埋める方法を提案するものである。

第3章　提案手法

ここでは、本研究で提案する手法に関する概念などを解説する。最初に、新たに用いた時間概念であるシステムティックと処理ティックの説明をする。次に、スケジューリングによるタスク切り替え処理をどの様に記述するかを述べる。最後に、タスク切り替え処理を可視化した上で非周期応答性を向上する手法を記述する。

3.1　新たに導入する時間の単位

本研究はRTOSによるタスク切り替え処理というオーバヘッドを管理するにあたり、時間軸に対して2つの概念を導入する。

- 周期的タスクの発生やデッドラインを定義するシステムティック
- 実際の処理の時間単位を定義する処理ティック

システムティックとはタスクの事前計画を行う最小時間単位になる。プログラマによって設定される実行する周期タスク達はこのティック上で処理の要求が到着し開始する。ただし、非周期的要求はシステムティックの上で処理が到着するとは限らない。

処理ティックはタスク切り替え等のRTOSオーバヘッドの量を定義する時間単位となる。また、プログラマが作った周期タスク達はこの処理ティックの間隔を最小として実行時間が揺れることを許す。システムティックは複数の処理ティックによって構成される。

3.2　タスク切り替え処理の特徴

タスク切り替え処理を分析した結果、次の2つの特徴のどちらかを持つことが分かった。

- 周期的に発生するタスクの起動処理
- 非周期的に発生するタスクの終了処理

導入した時間の単位の制約からタスクの起動処理は、必ずシステムティック上のタイミングでのみ要求が発生する。それ以外では発生することが無く完全に周期的な振る舞いをする。この周期的なタスクの起動処理を扱うサーバを、周期管理サーバと呼ぶことにする。このサーバは、周期的に発生が見込まれる要求の処理に特化した仕組みで処理を行う。この周期管理サーバは、任意のタスクより優先して実行されなければならない。

完全に非周期的に発生する終了処理であるが、既存のサーバ方式を用いることにより周期的な振る舞いをするサーバに閉じ込める。これらを用いることで RM で扱うことができる。この終了処理を担うサーバを終了管理サーバと名付けた。このサーバは、非周期的に発生する終了要求にいかなる時もタスク処理より優先で応えることが必要とされる。

3.3 周期管理サーバ

タスク切り替えを管理する上で、起動の管理を担当するのがこの周期管理サーバである。タスクの起動処理は周期的にしか起こることが無いため、本研究では周期処理に特化できる Polling Server を用いた。このサーバの周期は、タスクが起動する可能性がありうるシステムティックの長さと等しい。タスクの起動によって RM を崩さない様にするために、タスクの起動がいかなるシステムティック上で発生しても即座にそのタスクの起動処理を行う必要がある。サーバの容量は、スケジュールに組み込まれているタスクの数に依存して決まる。3 つのタスクがシステムの中に存在している場合、3 つのタスクを同時に起動できるだけの容量を必要とする。

起動するタスクがシステムに存在しているタスクより少ない場合において、容量は Polling Server の振る舞い上使われずに消えてしまう。毎システムティックで常にタスク起動数が最高になることはスケジュール可能性の観点から非現実的なため、殆どの場合この容量が使い尽くされることは無い。使い尽くされるタイミングは、システム内のタスクの周期の最小公倍数分のシステムティックのみでとなる。

図 3.1 は、周期管理サーバの振る舞いを図示したものである。このシステムには、2 つの周期的タスク、τ_1 と τ_2 が存在する。τ_1 は周期が 4 システムティックを持ち、実行時間が 1 システムティックである。τ_2 が持つ周期は 6 システムティックで、実行時間は τ_1 と同じ 1 システムティックである。システムがタスクを 1 つ起動させるのに 1 処理ティック必要だとした場合、タスクの数が 2 つなので 2 処理ティックを必要とする。

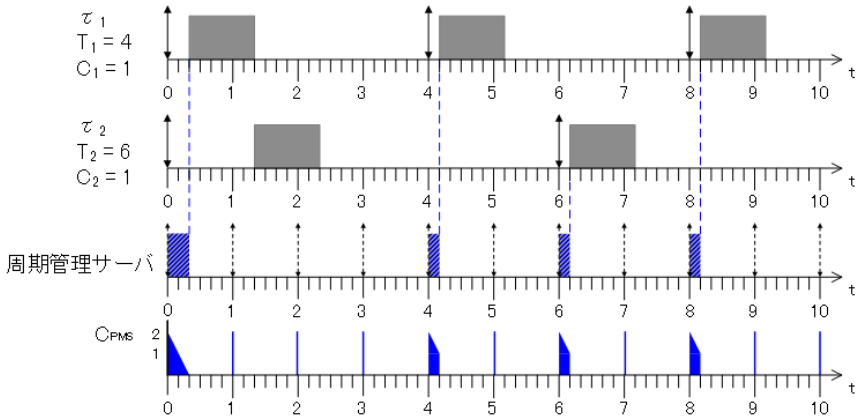

図 3.1: 周期管理サーバの振る舞いの様子

次に、この周期管理サーバを含めたシステムのスケジュール可能性について述べる。Polling Server は Hard Real-Time Computing Systems の本 [1] によると、RM として議論することできるとされている。このシステムの周期的タスクの数を n、周期的タスクの実行時間を C_i、周期的タスクの周期を T_i、周期管理サーバの容量を C_{PMS}、周期を T_{PMS} とすると十分条件は以下の 3.1 式となる。

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_{PMS}}{T_{PMS}} \leq (n+1)[2^{1/n+1} - 1] \tag{3.1}$$

3.4 終了管理サーバ

タスク切り替えを管理する上で、終了の管理を担当するのがこの終了管理サーバである。タスクの終了処理は非周期的に発生するため、本研究では非周期処理を RM に帰着させられる Deferrable Server を用いた。このサーバの周期は起動管理サーバと同じで、タスクが起動する可能性がありうるシステムティックの長さと等しい。タスクの終了によって RM を崩さない様にするために、タスクの終了がいかなる処理ティック上で発生しても即座にそのタスクの終了処理を行う必要がある。サーバの容量も、スケジュールに組み込まれているタスクの数と早期終了の有無に依存して決まる。3 つのタスクがシステムの中に存在している場合を考える。このシステムのタスクに対して早期終了を許さない時は、1 システムティックの間に 2 つ以上のタスクが終了することはない。したがって、容量は 1 つのタスクを終了する分だけ存在していればいい。しかし、早期終了を許す場合 3 つのタスクを同一ティックに終了できるだけの容量を必要とする。

本研究においては、早期終了を許す方向で話を進める。しかし、スケジュール可能性の保証の問題を扱う場合は最悪のケースを想定しなければならない。したがって、スケジュール可能性を扱う場合のみ、早期終了を許さないこととして終了管理サーバを考える。

図 3.2 は、終了管理サーバの振る舞いを図示したものである。このシステムには、2 つの周期的タスク、τ_1 と τ_2 が存在する。τ_1 は周期が 4 システムティックを持ち、実行時間が 1 システムティックである。τ_2 が持つ周期は 6 システムティックで、実行時間は τ_1 と同じ 1 システムティックである。このシステムはタスクの実行時間が最悪な場合のみを考え、システムがタスクを 1 つ終了させることができれば十分だと仮定する。一回のタスク終了処理に 1 処理ティック必要な時、サーバの容量は 1 処理ティックとなる。

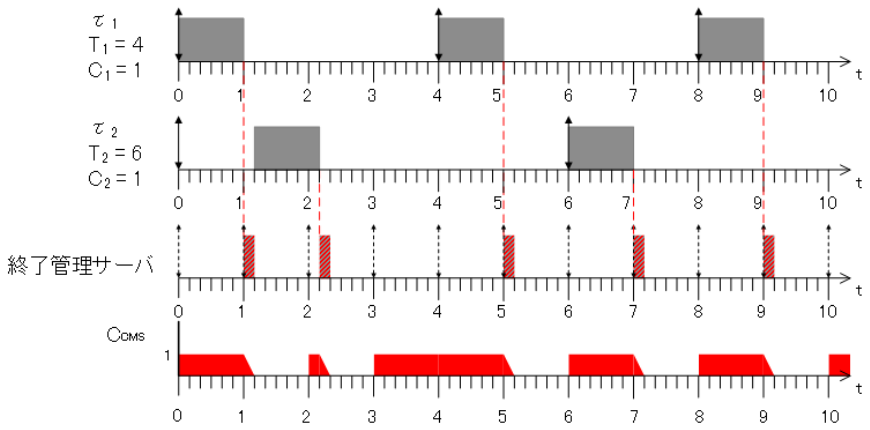

図 3.2: 終了管理サーバの振る舞いの様子

次に、この終了管理サーバを含めたシステムのスケジュール可能性について述べる。Deferrable Server も、RM として議論することできるとされている。このシステムの周期的タスクの数を n とし、周期的タスクの実行時間を C_i、周期的タスクの周期を T_i、周期管理サーバの容量を C_{AMS}、周期を T_{AMS} とすると十分条件は以下の 3.2 式となる。

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n[\left(\frac{C_{AMS}/T_{AMS}+2}{2C_{AMS}/T_{AMS}+1}\right)^{1/n+1} - 1] \tag{3.2}$$

3.5　2つの管理サーバの統合

周期管理サーバと終了管理サーバの2つを用いたシステムの様子を図3.3に示す。周期的タスクが起動された時に、周期サーバの容量を使い再優先にタスクの起動処理を行う。その後、タスクが終了した時刻にて終了管理サーバの容量を使い終了の処理及びのタスクの切り替えを行う。

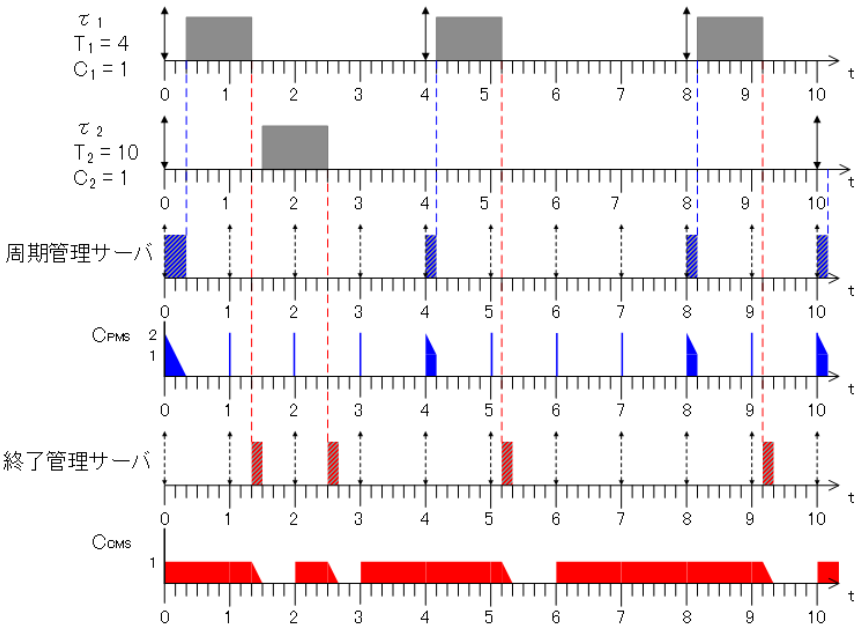

図 3.3: 周期管理サーバと終了管理サーバの両方の振る舞いの様子

このシステムのスケジュール可能性を保証する十分条件は以下の3.3式で与えられる。

$$\sum_{i=1}^{n} \frac{C_i}{T_i} + \frac{C_{PMS}}{T_{PMS}} \leq (n+1)\left[\left(\frac{C_{AMS}/T_{AMS}+2}{2C_{AMS}/T_{AMS}+1}\right)^{1/n+1} - 1\right] \tag{3.3}$$

3.6　管理サーバを用いることでの問題点

ここまでにおいてスケジュール可能性を保証する問題は解決することができた。しかしながら、システムが必要とする周期的タスクの数が多くなればなる程に、管理サーバの容量が全て使われない場合が増えてしまう。この無駄は、周期管理サーバではシステムで全てのタスクが起動する以外の場合に生じる。終了管理サーバにおいては、同一システムティック内において終了するタスクが複数無い場合である。このタスク切り替え処理の負荷を過大見積もりをしてしまう現象は、減らすことができなかった。

3.7 SLACKの再利用

両方の管理サーバに共通していることは、システム内の全てのタスクを処理しない場合において容量が浪費されるか残されてしまうことである。しかし、この全てのタスクが処理されるタイミングは早期終了を許した現実的なシステムにおいて事前の予測は不可能である。本論文では、事前の予測ができない代わりに実行中に使われないと判断できるサーバの容量を計測する方法を紹介する。これがSlackの再利用である。

多くの汎用的なRTOSシステムにおいて、周期的な処理だけでなく非周期的な要求は存在している。システムにおいて最高優先度で使えるこれらSlackを非周期的要求を実行するために用いることで、アプリケーションの応答速度の向上を見込む。

周期管理サーバにおいてのSlackは、システムティック上のタスクが起動された直後に判別できる。起動されなかったタスクの数を起動に必要なオーバヘッドで掛けた値がSlackの値と等しくなる。別の言い方ならば、Polling Serverの仕組み上容量を保持できない値そのものがその時に利用できるSlackである。ただし、このSlackは使用せず保存して他の処理が終えた後で用いることはできない。利用できる場合は、Slackの有無が分かるより前に処理の要求が存在している時である。

終了管理サーバにおいてのSlackは、毎システムティック上で計測することが可能となる。タスクは、終了した後から次の起動するシステムティックまでには終了処理が生じることはない。タスクが終了した直後では、実行した終了処理によって既にサーバの容量が消費されてしまっている。よって、タスクが終了したシステムティック内ではSlackが生じることはない。言い換えれば、この終了管理サーバのSlackはシステムティック上でのみ生じうる。サーバの容量が補充される次のシステムティックの時刻から次の起動までがその終了したタスクが生む終了管理サーバのSlackとなる。このSlackの元はDeferrable Serverの容量なので、少なくとも次のシステムティックまでは任意の処理ティックのタイミングに用いることができる。

本研究では、Slackが生じた際にアプリケーションタスクを処理するサーバにSlackの容量を性質を維持したまま譲渡することで管理している。

第4章　提案手法の評価

4.1　評価環境

ここでは、評価をするために用いたシミュレータの環境とシミュレーションに用いたタスクセットの生成方法に関して記述する。最初に、シミュレーションに用いた周期的タスクセットとアプリケーションタスクセットの生成方法を述べる。次に、RM で走らせたシミュレータの環境を記す。

4.1.1　周期タスクの生成方法

周期的タスクを生成する際に必要な情報として下記の情報が挙げられる。

- 指数分布で発生する疑似乱数生成に必要な種 (Seed)
- 生成したいタスクセットのプロセッサ利用率
- 生成したいタスクセットの許容誤差値
- タスクの周期、最悪実行時間とシミュレータ上で利用する平均実行時間の平均値

周期的タスクを生成するプログラムに入力した情報として、疑似乱数生成に必要な Seed 値は Z シェル $RAMDOM 値を使用した。生成したいプロセッサ利用率は 50 %から 70 %までの値を 5 %刻みで入力し、それぞれのタスクセットの許容誤差は指定したプロセッサ利用率を上限とし下限を-1 %までとした。タスク情報のそれぞれの平均値は周期の情報を 100 システムティックとし、最悪実行時間の情報を 27 システムティックとした。シミュレータ上で走らせる際に用いる平均実行時間は、20 システムティックとした。周期的タスクそのものを生成する方法の概略は図 4.1 で示す。

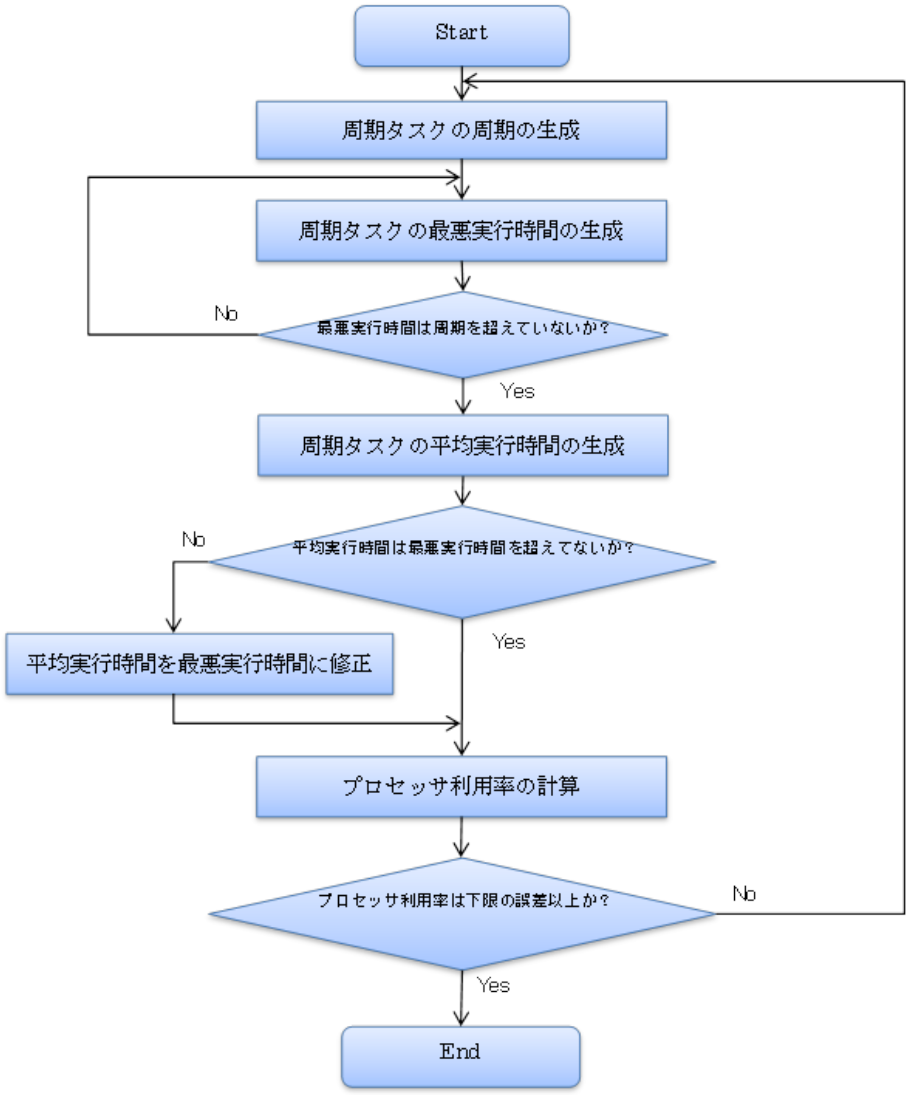

図 4.1: 周期タスク生成の流れ

これらの情報を参照して、指数分布となる乱数生成装置を用いることによりタスクセットの情報を生成した。指数分布となる乱数生成装置の一部に一様分布の乱数発生装置が必要となる。そのい実装は、メルセルヌ・トゥイスタのアルゴリズムを用いた。

単純に生成すると、タスク情報次第では指定したプロセッサ利用率を達成するために必要な残りの利用率が数%未満となる場合がある。この場合、パラメータ指定に依存するが、数%のタスクを生成できない為に無限ループに近い症状が現れる。この対策として、上限試行回数を超えたタスク生成を場合に限り、既存のタスク情報を全て捨てて最初から生成しなおす方法をとる。この既存情報を捨て最初から生成する場合、失敗した回数として情報を記録する。この失敗の回数が閾値以上になった場合、タスクの生成を諦めタスク生成に失敗したことを伝える。また、タスクの生成がプロセッサ利用率の面で満たしてい

たとしても LUB 以上のタスクセットが生成された場合、再生成の閾値を超えた扱いと同様に失敗扱いとしてその旨を通知する。これらの流れは図 4.2 にて示す。

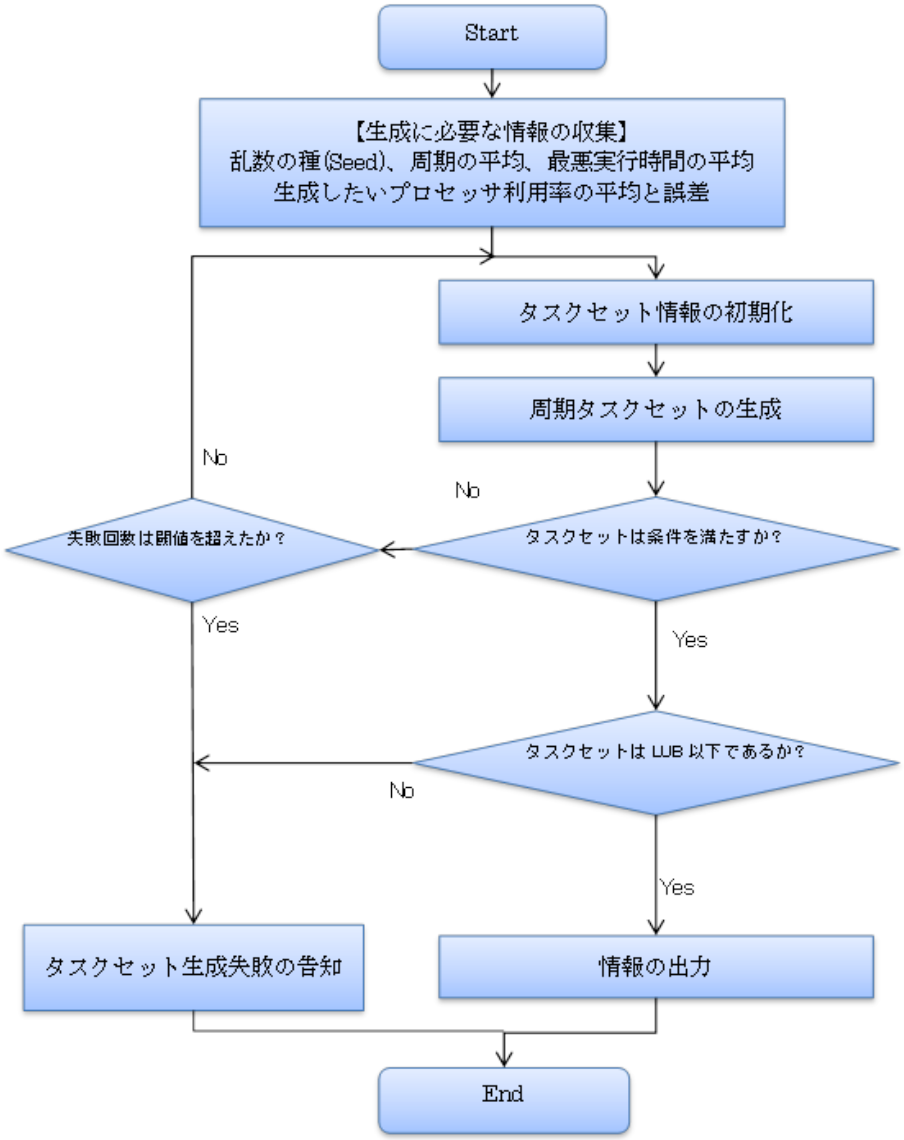

図 4.2: 周期タスクセット生成の流れ

4.1.2 アプリケーションタスクの生成方法

アプリケーションタスクを生成する際に必要な情報として下記の情報が挙げられる。

- 指数分布で発生する疑似乱数生成に必要な種 (Seed)
- アプリケーションタスクの実行に必要な時間の平均値
- アプリケーションタスクが発生する間隔の平均値
- シミュレータが終了する内部時間

アプリケーションタスクを生成するプログラムには、Z シェルの $RAMDOM 値を使用した疑似乱乱数生成に必要な Seed 値を用いる。ほかにも生成したいタスクセットのアプリケーションタスクが 1 回発生した際、実行に必要な時間の平均値とアプリケーションタスクが発生する間隔の平均値が必要となる。本研究では、アプリケーションタスクが実行に必要な時間の平均値を目的のプロセッサ率システムティックとした。また、アプリケーションタスクが発生する間隔の平均値を 100 システムティックとした。アプリケーションタスクが発生する可能性がある時間の指定である、後のシミュレータが終了するまでの内部時刻の指定を行うことによりシミュレータが終了する時刻直前までに発生するアプリケーションリクエストを予め生成する。この値は 10 万システムティックとした。これらの情報を用いて、非周期的タスクを生成する流れを図 4.3 に示す。

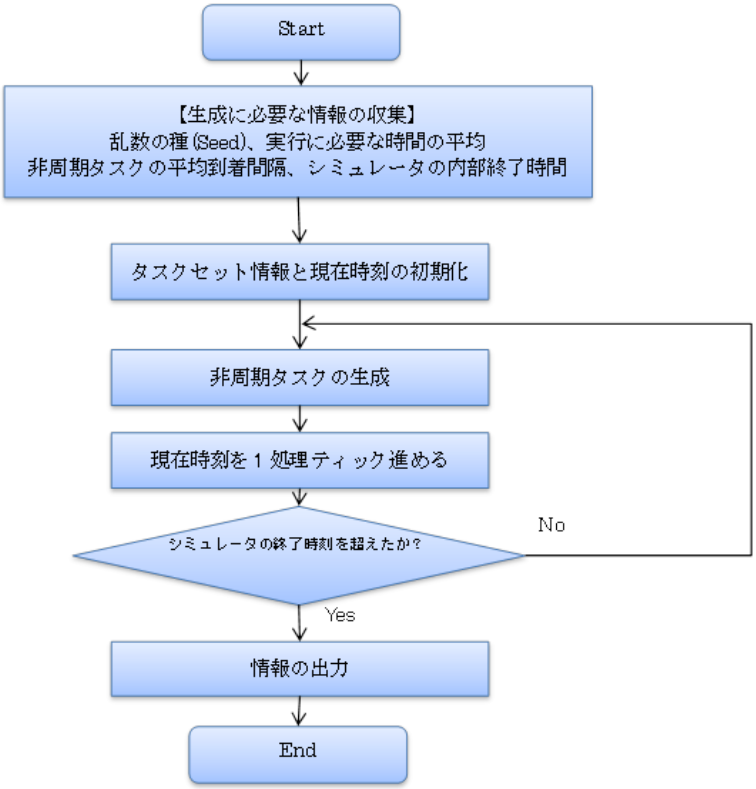

図 4.3: アプリケーションタスク生成の流れ

これらの情報を参照して、指数分布となる乱数生成装置を用いるによりタスクセットの情報を生成した。指数分布となる乱数生成装置の一部に一様分布の乱数発生装置が必要となる。その部分の実装は周期タスクセットを生成した時と同様に、メルセルヌ・トゥイスタのアルゴリズムを用いた。

4.1.3 RM シミュレータ

先の節で説明した方法によって生成されたタスクを、RM によって動作するシミュレータに入力してシステムの挙動を計測した。本研究においてアプリケーションタスクを実行する優先度は最低で、Background スケジューリングと等しい。なお、シミュレータが実行する前にオーバヘッドの量を処理ティックで指定する必要がある。このシミュレータは動き始めると 10 万システムティックまで観測を行い、観測結果を出力する。出力する情報として次の情報が挙げられる。

- 周期的タスクは観測時間内でデッドラインを守れたか？

- シミュレーション上のシステム負荷
- アプリケーションタスクの平均応答時間

である。これらの情報を用いてアプリケーションの応答性の評価を行った。

4.2 評価結果

ここでは、アプリケーションの応答時間と各サーバの容量を交換することによる応答時間への寄与の量を結果で示す。最初に提示するデータはオーバヘッドの値と周期タスクの負荷を一定にした時のアプリケーションの負荷に対する応答時間を示す。次にアプリケーションの負荷を一定にし、オーバヘッドの値と周期的タスクの負荷によってどの様な形で応答速度に違いが出るのかを図示する。

4.2.1 アプリケーションの応答時間

(単位：システムティック)

表 4.1: オーバヘッドが 1 処理ティックにした時の周期タスクの負荷率 50 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.24	3.55	4.89	6.34	7.77	9.22	10.94	12.44	13.98	15.95
周期管理サーバから譲渡	2.12	3.47	4.83	6.29	7.73	9.19	10.91	12.41	13.95	15.92
終了管理サーバから譲渡	2.19	3.52	4.87	6.32	7.76	9.21	10.93	12.43	13.97	15.94
周期&終了管理サーバから譲渡	2.08	3.45	4.81	6.28	7.72	9.18	10.89	12.40	13.94	15.91

表 4.2: オーバヘッドが 1 処理ティックにした時の周期タスクの負荷率 55 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.69	3.97	5.28	6.67	8.09	9.52	11.17	12.64	14.14	16.07
周期管理サーバから譲渡	2.45	3.82	5.17	6.58	8.01	9.45	11.10	12.58	14.09	16.01
終了管理サーバから譲渡	2.57	3.89	5.23	6.63	8.05	9.49	11.14	12.61	14.12	16.04
周期&終了管理サーバから譲渡	2.36	3.76	5.12	6.54	7.98	9.42	11.08	12.55	14.06	15.99

表 4.3: オーバヘッドが 1 処理ティックにした時の周期タスクの負荷率 60 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.71	4.06	5.47	6.99	8.48	10.06	11.87	13.48	15.14	17.24
周期管理サーバから譲渡	2.59	3.99	5.42	6.95	8.45	10.03	11.83	13.45	15.12	17.22
終了管理サーバから譲渡	2.71	4.06	5.47	6.99	8.48	10.06	11.87	13.48	15.14	17.24
周期&終了管理サーバから譲渡	2.59	3.99	5.42	6.95	8.45	10.03	11.83	13.45	15.12	17.22

表 4.4: オーバヘッドが 1 処理ティックにした時の周期タスクの負荷率 65 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.70	3.97	5.27	6.69	8.08	9.55	11.24	12.74	14.28	16.24
周期管理サーバから譲渡	2.58	3.90	5.21	6.64	8.05	9.52	11.21	12.71	14.25	16.21
終了管理サーバから譲渡	2.70	3.97	5.27	6.69	8.08	9.55	11.24	12.74	14.28	16.24
周期&終了管理サーバから譲渡	2.58	3.90	5.21	6.64	8.05	9.52	11.21	12.71	14.25	16.21

表 4.5: オーバヘッドが 1 処理ティックにした時の周期タスクの負荷率 70 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	3.53	4.92	6.38	7.96	9.56	11.20	13.16	14.85	16.66	18.87
周期管理サーバから譲渡	3.33	4.80	6.29	7.89	9.50	11.15	13.10	14.80	16.61	18.83
終了管理サーバから譲渡	3.53	4.92	6.38	7.96	9.56	11.20	13.16	14.85	16.66	18.87
周期&終了管理サーバから譲渡	3.33	4.80	6.29	7.89	9.50	11.15	13.10	14.80	16.61	18.83

(単位：システムティック)

表 4.6: オーバヘッドが 2 処理ティックにした時の周期タスクの負荷率 50 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.25	3.56	4.91	6.37	7.81	9.27	11.00	12.51	14.05	16.04
周期管理サーバから譲渡	2.02	3.42	4.80	6.28	7.73	9.20	10.93	12.44	13.99	15.98
終了管理サーバから譲渡	2.16	3.51	4.87	6.34	7.78	9.25	10.97	12.48	14.03	16.02
周期&終了管理サーバから譲渡	1.96	3.37	4.77	6.25	7.70	9.18	10.91	12.42	13.97	15.96

表 4.7: オーバヘッドが 2 処理ティックにした時の周期タスクの負荷率 55 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.69	3.98	5.30	6.69	8.11	9.54	11.20	12.67	14.18	16.11
周期管理サーバから譲渡	2.27	3.70	5.08	6.51	7.95	9.40	11.07	12.55	14.07	16.00
終了管理サーバから譲渡	2.47	3.84	5.19	6.61	8.04	9.48	11.14	12.62	14.13	16.06
周期&終了管理サーバから譲渡	2.14	3.59	4.99	6.44	7.89	9.34	11.01	12.50	14.01	15.95

表 4.8: オーバヘッドが 2 処理ティックにした時の周期タスクの負荷率 60 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.72	4.07	5.48	7.01	8.51	10.09	11.90	13.52	15.19	17.30
周期管理サーバから譲渡	2.49	3.93	5.38	6.93	8.43	10.02	11.84	13.46	15.14	17.24
終了管理サーバから譲渡	2.72	4.07	5.48	7.01	8.51	10.09	11.90	13.52	15.19	17.30
周期&終了管理サーバから譲渡	2.49	3.93	5.38	6.93	8.43	10.02	11.84	13.46	15.14	17.24

表 4.9: オーバヘッドが 2 処理ティックにした時の 65 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.71	3.98	5.27	6.69	8.09	9.56	11.25	12.76	14.29	16.26
周期管理サーバから譲渡	2.47	3.84	5.17	6.61	8.02	9.49	11.19	12.69	14.24	16.20
終了管理サーバから譲渡	2.71	3.98	5.27	6.69	8.09	9.56	11.25	12.76	14.29	16.26
周期&終了管理サーバから譲渡	2.47	3.84	5.17	6.61	8.02	9.49	11.19	12.69	14.24	16.20

表 4.10: オーバヘッドが 2 処理ティックにした時の周期タスクの負荷率 70 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	3.53	4.93	6.39	7.97	9.58	11.21	13.18	14.87	16.69	18.90
周期管理サーバから譲渡	3.16	4.70	6.22	7.83	9.45	11.11	13.07	14.77	16.60	18.81
終了管理サーバから譲渡	3.53	4.93	6.39	7.97	9.58	11.21	13.18	14.87	16.69	18.90
周期&終了管理サーバから譲渡	3.16	4.70	6.22	7.83	9.45	11.11	13.07	14.77	16.60	18.81

(単位：システムティック)

表 4.11: オーバヘッドが 4 処理ティックにした時の周期タスクの負荷率 50 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.27	3.60	4.96	6.44	7.89	9.37	11.12	12.65	14.21	16.23
周期管理サーバから譲渡	1.88	3.33	4.75	6.26	7.73	9.23	10.98	12.52	14.09	16.11
終了管理サーバから譲渡	2.11	3.49	4.88	6.37	7.83	9.32	11.07	12.60	14.17	16.19
周期&終了管理サーバから譲渡	1.81	3.26	4.69	6.20	7.68	9.18	10.93	12.47	14.05	16.07

表 4.12: オーバヘッドが 4 処理ティックにした時の周期タスクの負荷率 55 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.71	4.00	5.32	6.72	8.15	9.59	11.26	12.74	14.26	16.20
周期管理サーバから譲渡	2.02	3.50	4.92	6.38	7.84	9.31	11.00	12.49	14.02	15.97
終了管理サーバから譲渡	2.32	3.74	5.12	6.56	8.01	9.47	11.14	12.63	14.15	16.10
周期&終了管理サーバから譲渡	1.85	3.34	4.77	6.25	7.72	9.19	10.88	12.38	13.92	15.87

表 4.13: オーバヘッドが 4 処理ティックにした時の周期タスクの負荷率 60 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.73	4.09	5.52	7.05	8.56	10.15	11.97	13.60	15.29	17.41
周期管理サーバから譲渡	2.32	3.83	5.32	6.88	8.41	10.02	11.85	13.48	15.17	17.30
終了管理サーバから譲渡	2.73	4.09	5.52	7.05	8.56	10.15	11.97	13.60	15.29	17.41
周期&終了管理サーバから譲渡	2.32	3.83	5.32	6.88	8.41	10.02	11.85	13.48	15.17	17.30

表 4.14: オーバヘッドが 4 処理ティックにした時の周期タスクの負荷率 65 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	2.72	3.99	5.29	6.71	8.11	9.58	11.28	12.79	14.33	16.29
周期管理サーバから譲渡	2.30	3.72	5.09	6.54	7.96	9.45	11.15	12.66	14.21	16.18
終了管理サーバから譲渡	2.72	3.99	5.29	6.71	8.11	9.58	11.28	12.79	14.33	16.29
周期&終了管理サーバから譲渡	2.30	3.72	5.09	6.54	7.96	9.45	11.15	12.66	14.21	16.18

表 4.15: オーバヘッドが 4 処理ティックにした時の周期タスクの負荷率 70 %のアプリケーション応答時間

非周期負荷率	1	2	3	4	5	6	7	8	9	10
管理サーバ無し	3.55	4.94	6.41	8.00	9.61	11.25	13.22	14.92	16.75	18.97
周期管理サーバから譲渡	2.88	4.51	6.08	7.72	9.36	11.04	13.01	14.72	16.56	18.78
終了管理サーバから譲渡	3.55	4.94	6.41	8.00	9.61	11.25	13.22	14.92	16.75	18.97
周期&終了管理サーバから譲渡	2.88	4.51	6.08	7.72	9.36	11.04	13.01	14.72	16.56	18.78

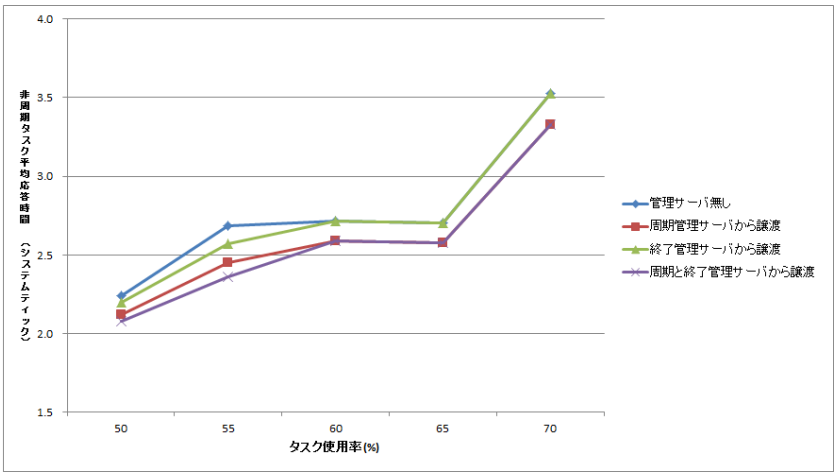

図 4.4: オーバヘッドが 1 処理ティックにした時の 1 %のアプリケーションの応答時間

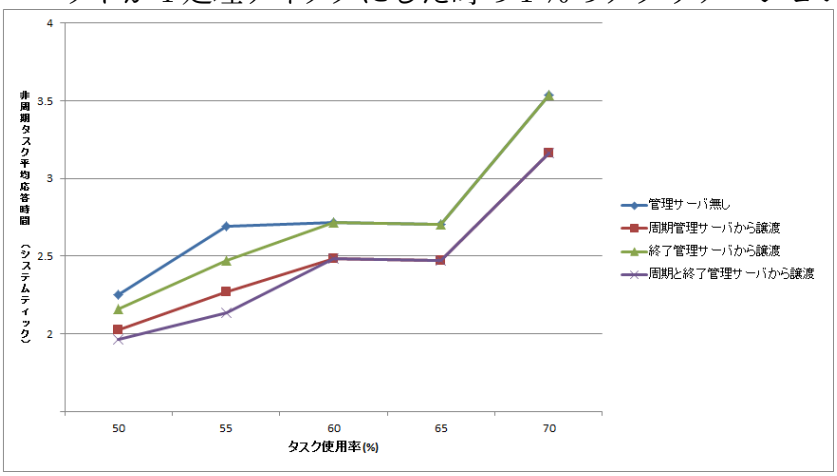

図 4.5: オーバヘッドが 2 処理ティックにした時の 1 %のアプリケーションの応答時間

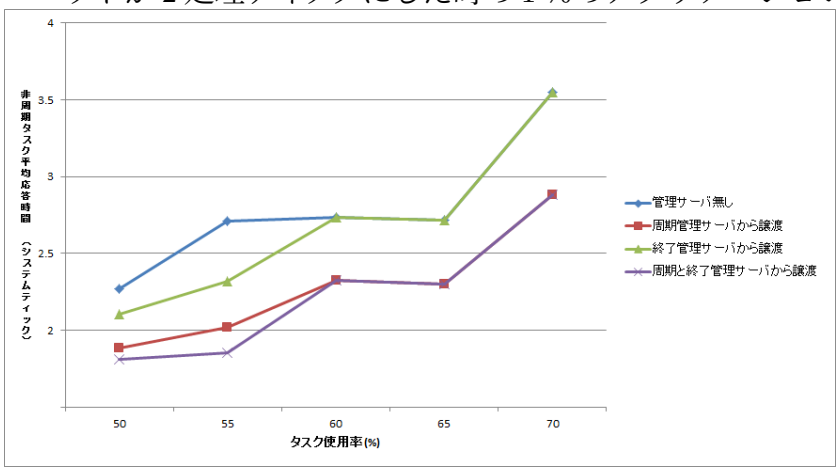

図 4.6: オーバヘッドが 4 処理ティックにした時の 1 %のアプリケーションの応答時間

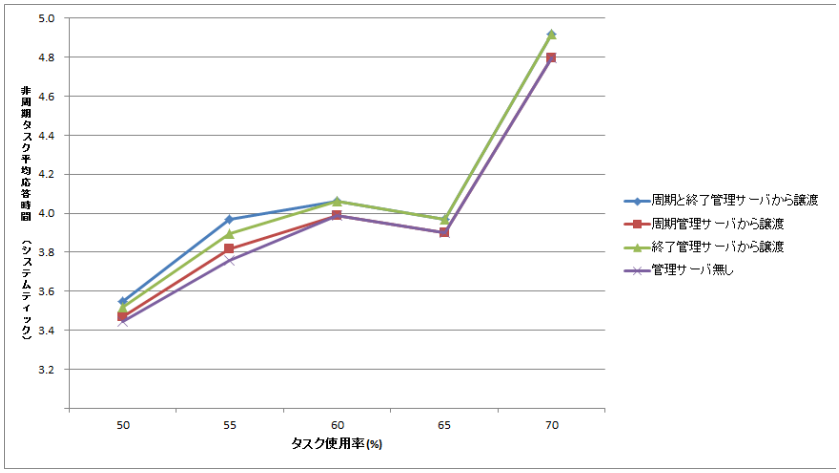

図 4.7: オーバヘッドが 1 処理ティックにした時の 2 %のアプリケーションの応答時間

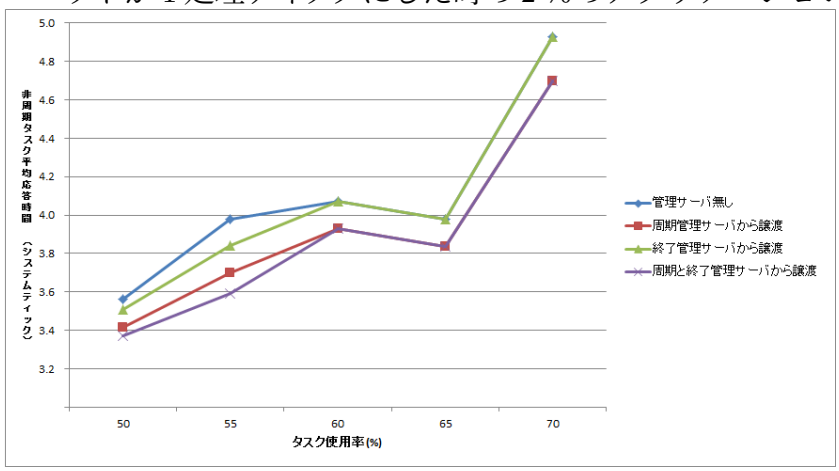

図 4.8: オーバヘッドが 2 処理ティックにした時の 2 %のアプリケーションの応答時間

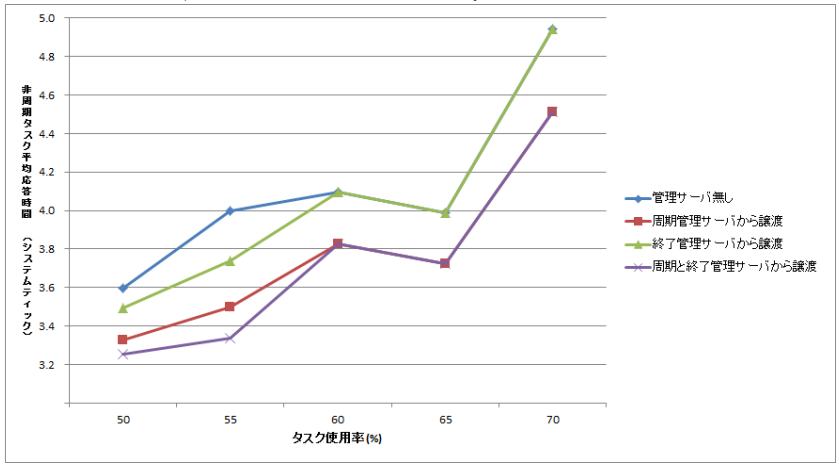

図 4.9: オーバヘッドが 4 処理ティックにした時の 2 %のアプリケーションの応答時間

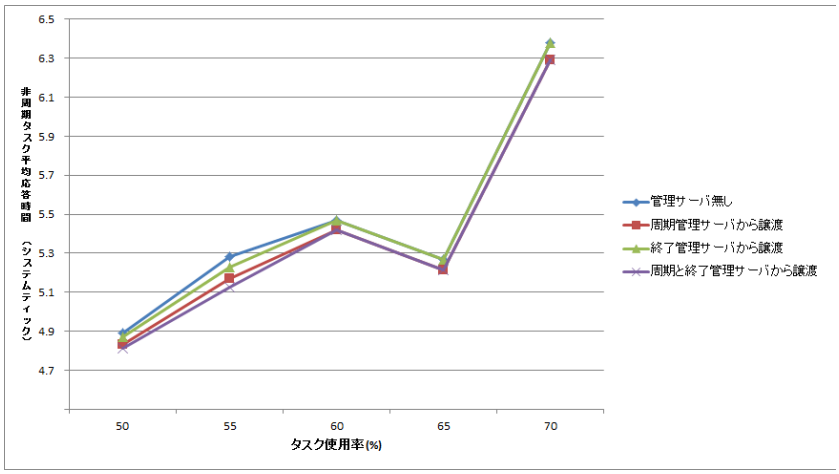

図 4.10: オーバヘッドが 1 処理ティックにした時の 3 %のアプリケーションの応答時間

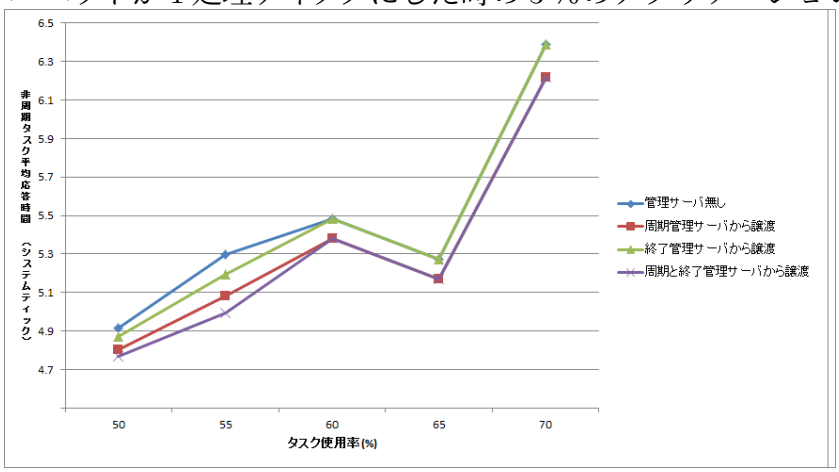

図 4.11: オーバヘッドが 2 処理ティックにした時の 3 %のアプリケーションの応答時間

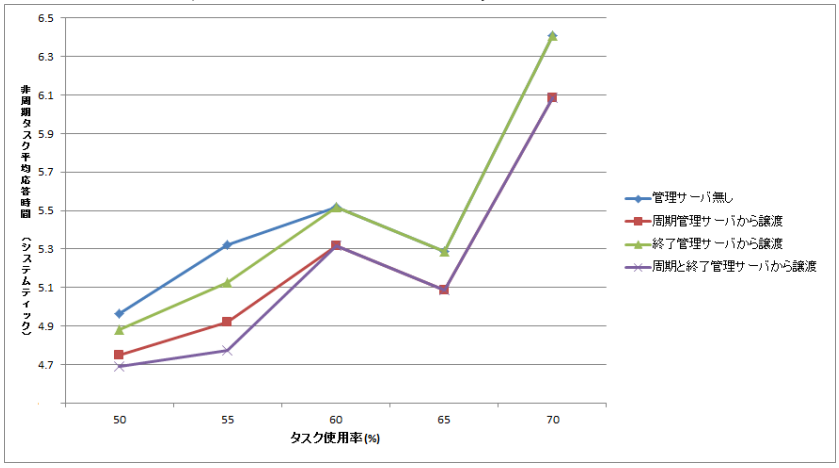

図 4.12: オーバヘッドが 4 処理ティックにした時の 3 %のアプリケーションの応答時間

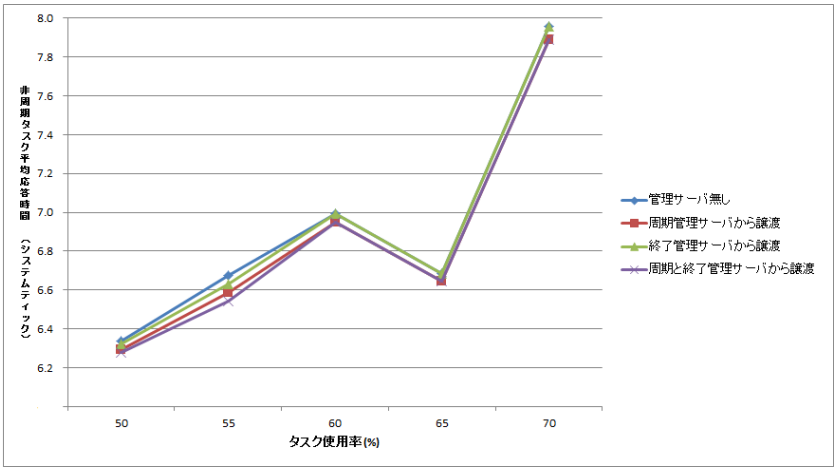

図 4.13: オーバヘッドが 1 処理ティックにした時の 4 %のアプリケーションの応答時間

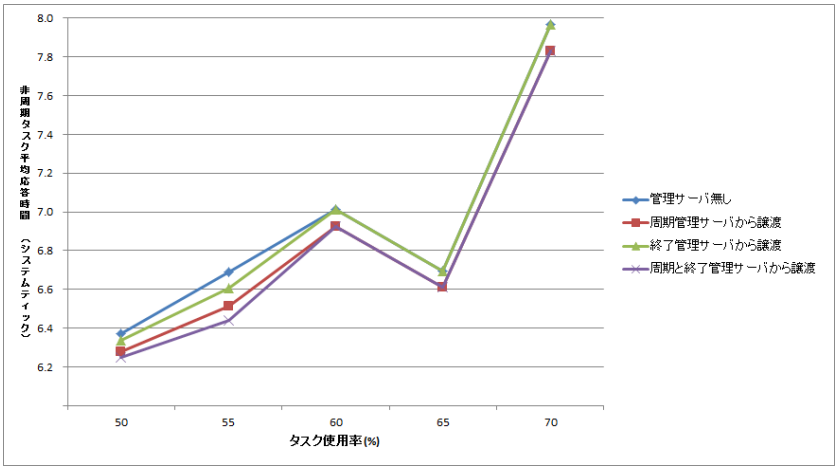

図 4.14: オーバヘッドが 2 処理ティックにした時の 4 %のアプリケーションの応答時間

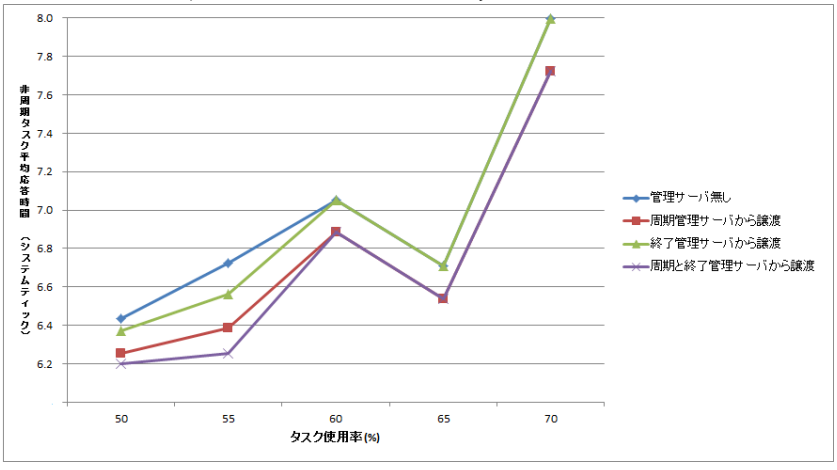

図 4.15: オーバヘッドが 4 処理ティックにした時の 4 %のアプリケーションの応答時間

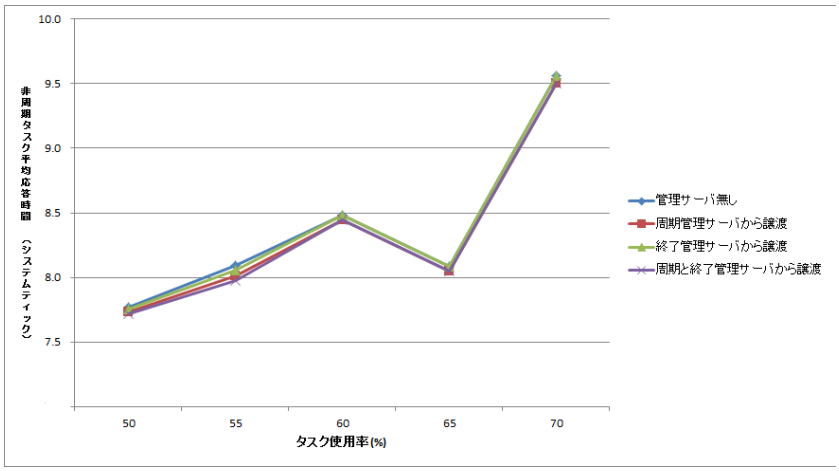

図 4.16: オーバヘッドが 1 処理ティックにした時の 5 %のアプリケーションの応答時間

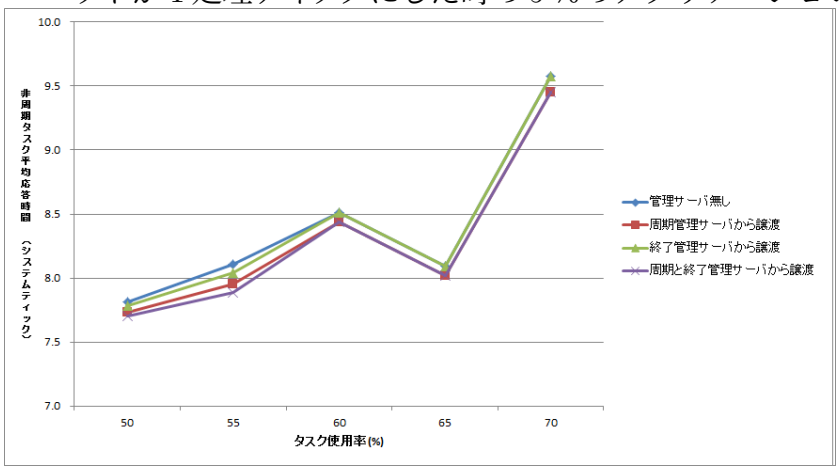

図 4.17: オーバヘッドが 2 処理ティックにした時の 5 %のアプリケーションの応答時間

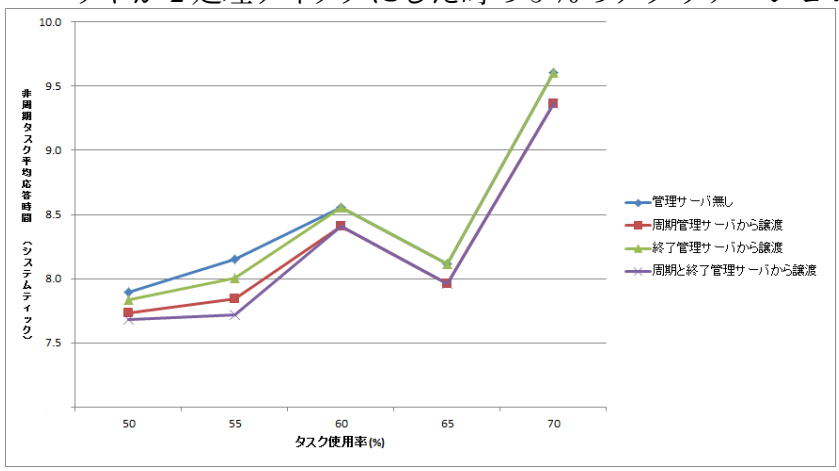

図 4.18: オーバヘッドが 4 処理ティックにした時の 5 %のアプリケーションの応答時間

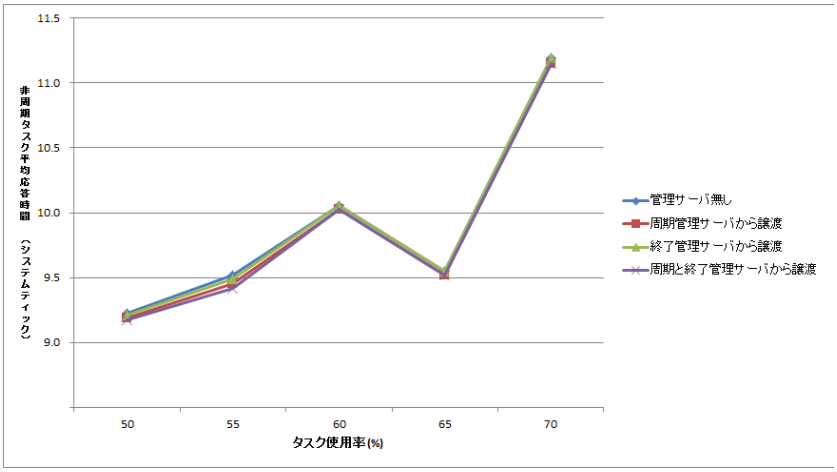

図 4.19: オーバヘッドが 1 処理ティックにした時の 6 %のアプリケーションの応答時間

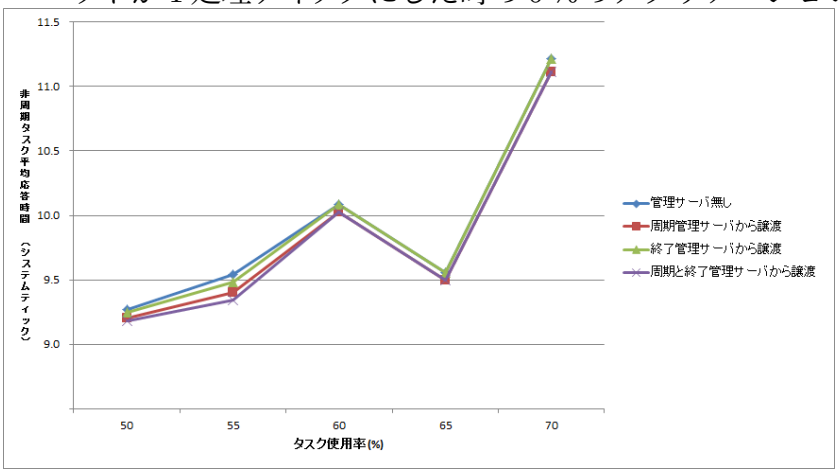

図 4.20: オーバヘッドが 2 処理ティックにした時の 6 %のアプリケーションの応答時間

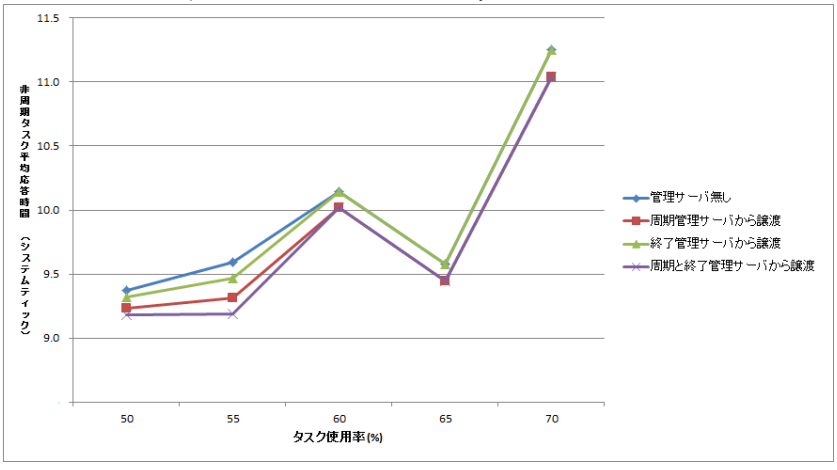

図 4.21: オーバヘッドが 4 処理ティックにした時の 6 %のアプリケーションの応答時間

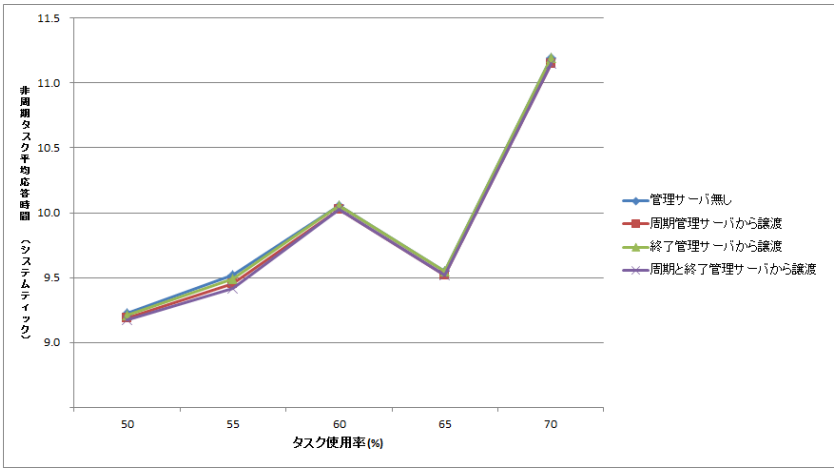

図 4.22: オーバヘッドが 1 処理ティックにした時の 7 %のアプリケーションの応答時間

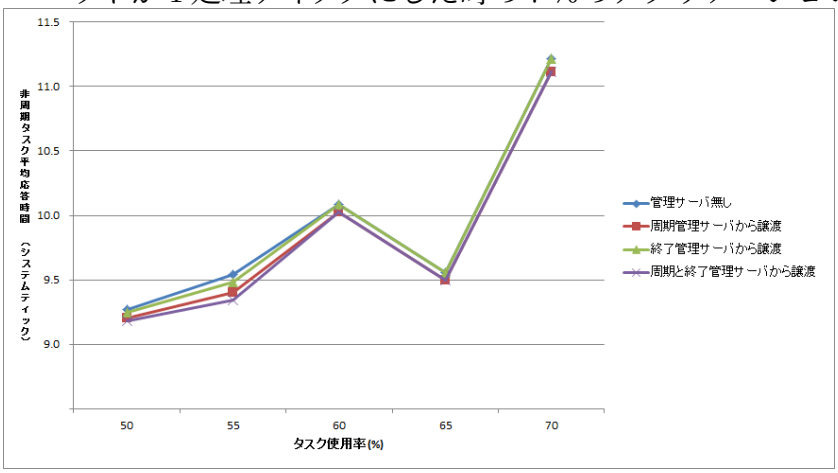

図 4.23: オーバヘッドが 2 処理ティックにした時の 7 %のアプリケーションの応答時間

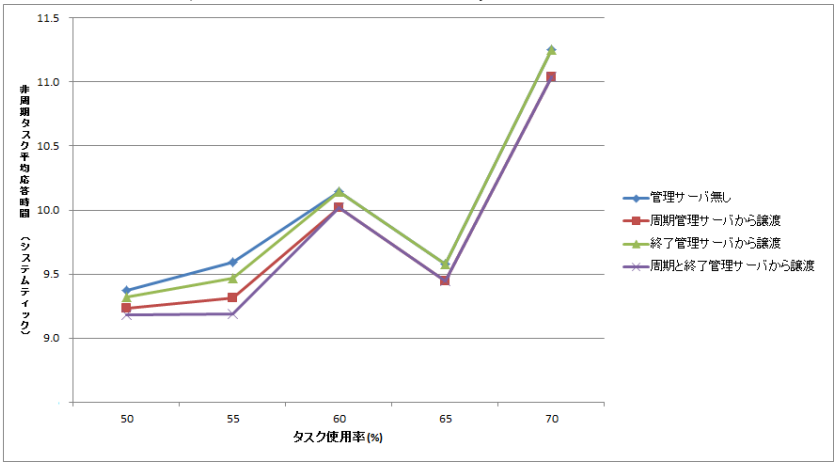

図 4.24: オーバヘッドが 4 処理ティックにした時の 7 %のアプリケーションの応答時間

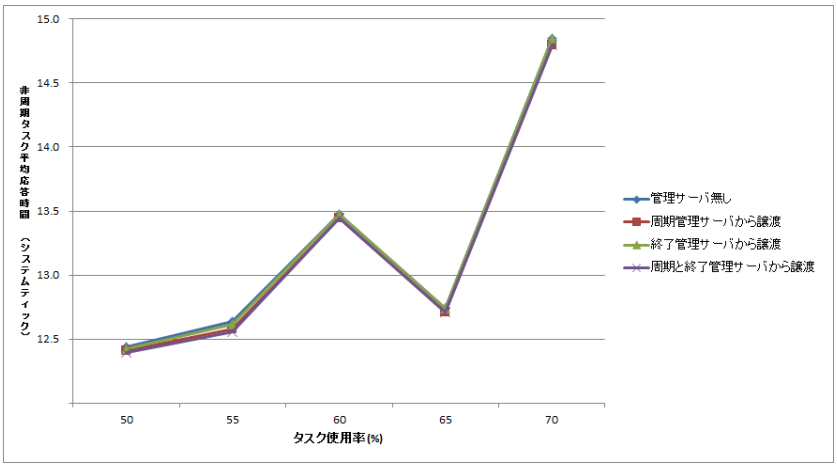

図 4.25: オーバヘッドが 1 処理ティックにした時の 8 %のアプリケーションの応答時間

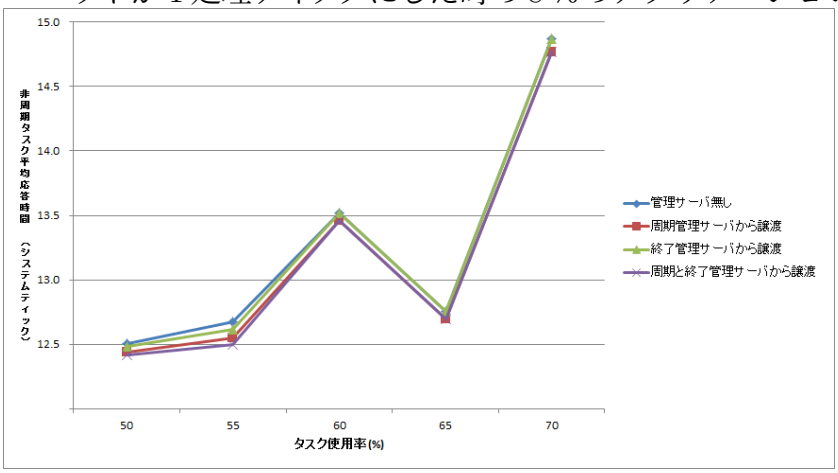

図 4.26: オーバヘッドが 2 処理ティックにした時の 8 %のアプリケーションの応答時間

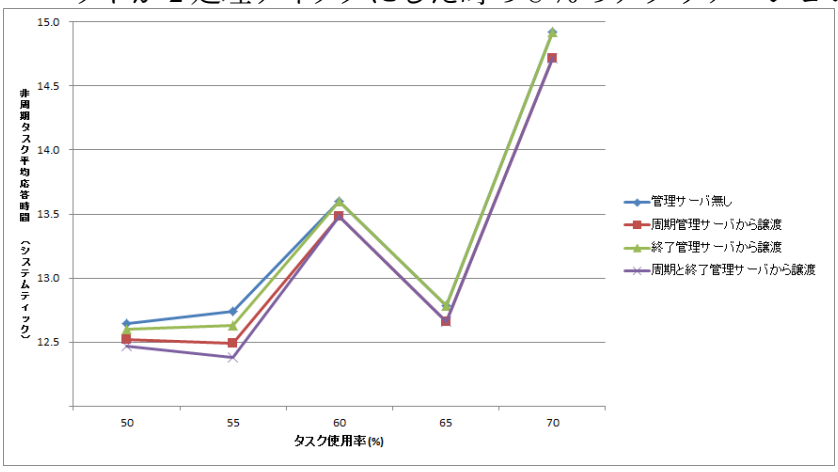

図 4.27: オーバヘッドが 4 処理ティックにした時の 8 %のアプリケーションの応答時間

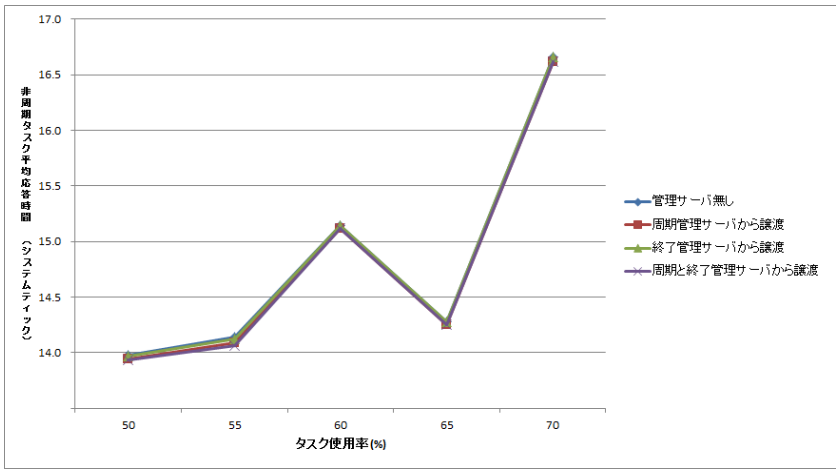

図 4.28: オーバヘッドが 1 処理ティックにした時の 9 %のアプリケーションの応答時間

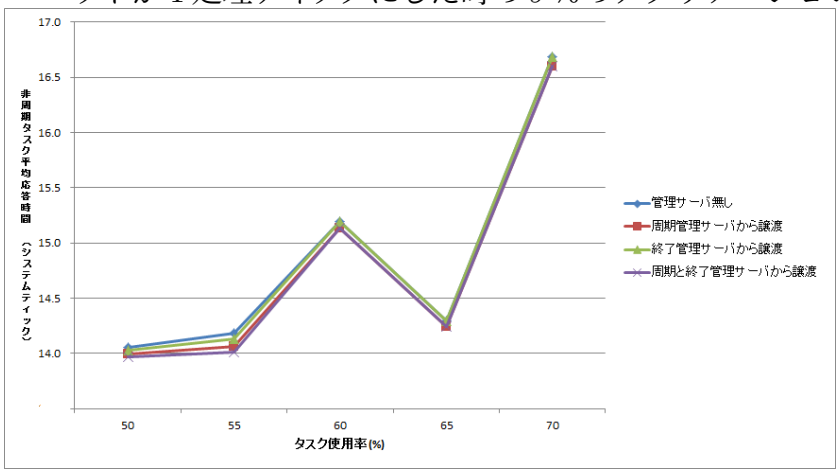

図 4.29: オーバヘッドが 2 処理ティックにした時の 9 %のアプリケーションの応答時間

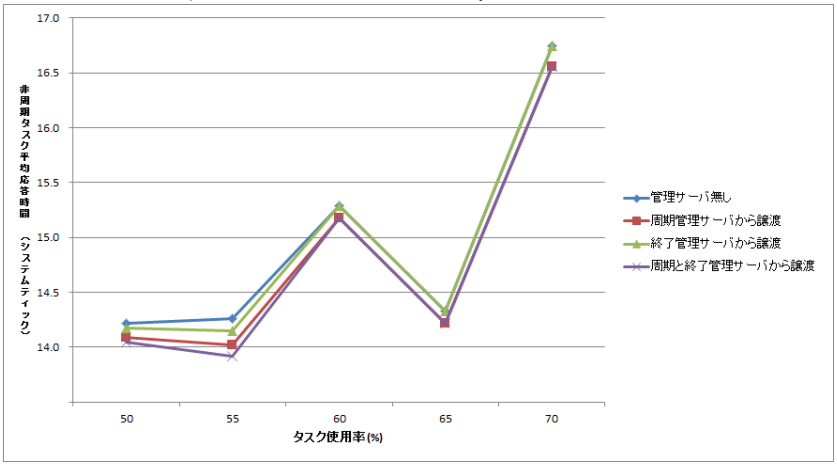

図 4.30: オーバヘッドが 4 処理ティックにした時の 9 %のアプリケーションの応答時間

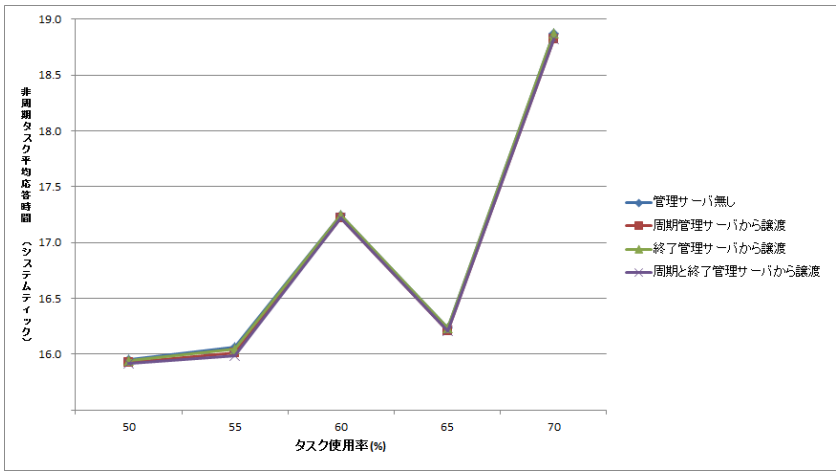

図 4.31: オーバヘッドが 1 処理ティックにした時の 10 %のアプリケーションの応答時間

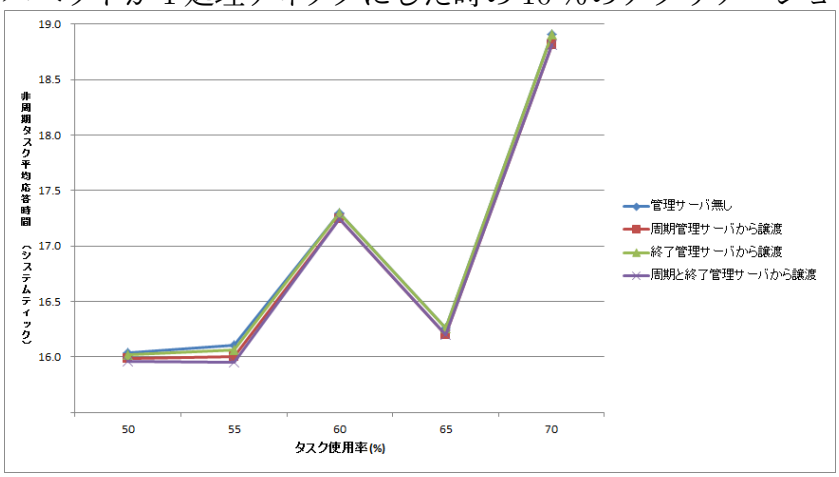

図 4.32: オーバヘッドが 2 処理ティックにした時の 10 %のアプリケーションの応答時間

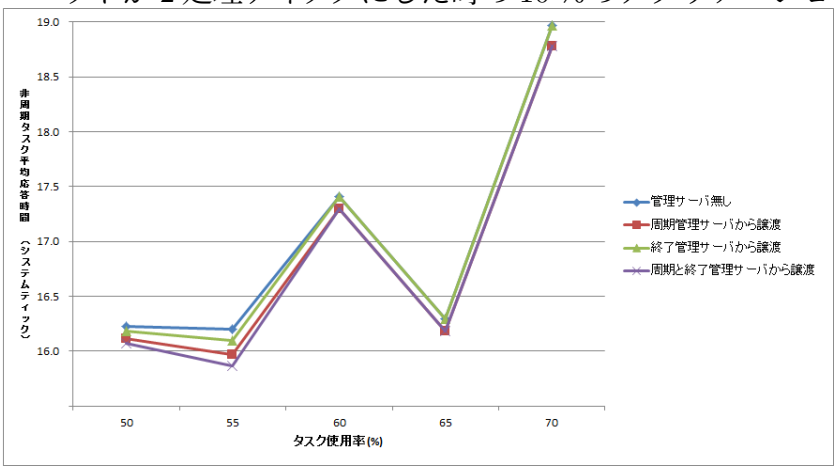

図 4.33: オーバヘッドが 4 処理ティックにした時の 10 %のアプリケーションの応答時間

4.2.2　アプリケーションの応答時間向上率

タスク切り替えのオーバヘッドが 1 処理ティックの場合のグラフを示す。

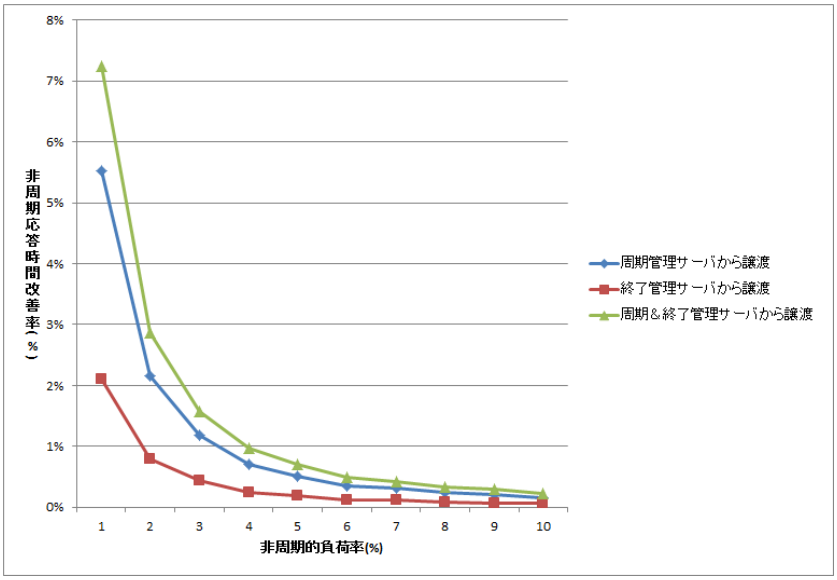

図 4.34: 周期的タスクの負荷が 50 %の時のアプリケーションの応答時間向上率の比較

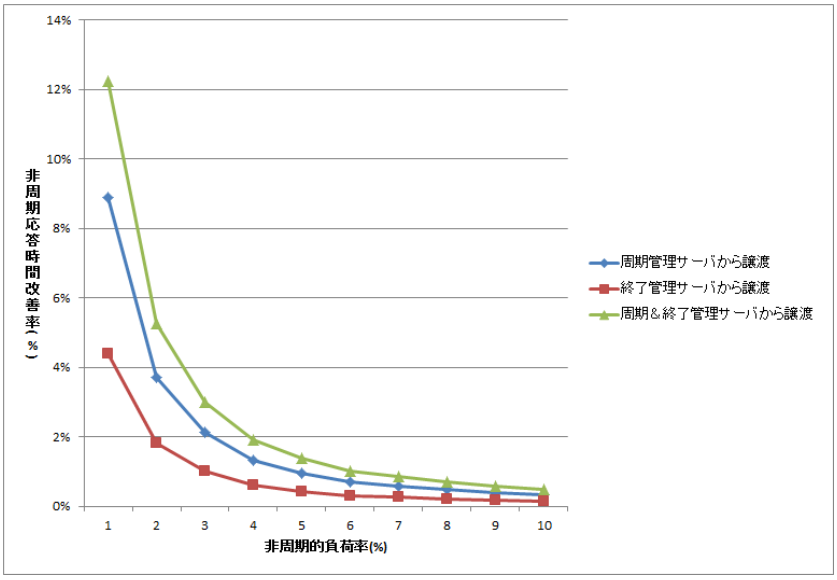

図 4.35: 周期的タスクの負荷が 55 %の時のアプリケーションの応答時間向上率の比較

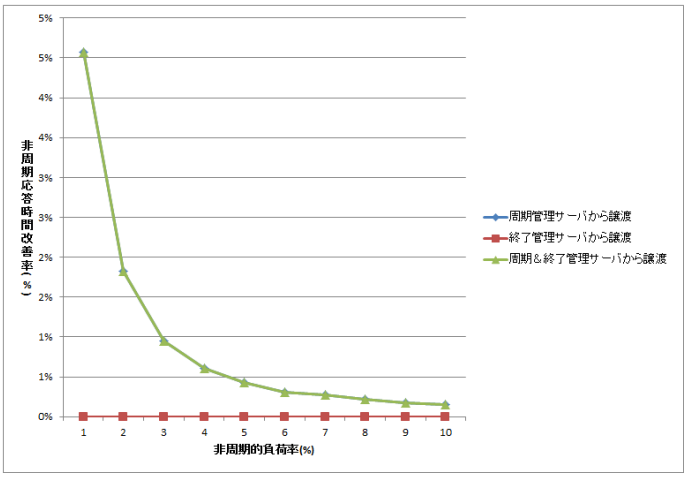

図 4.36: 周期的タスクの負荷が 60 %の時のアプリケーションの応答時間向上率の比較

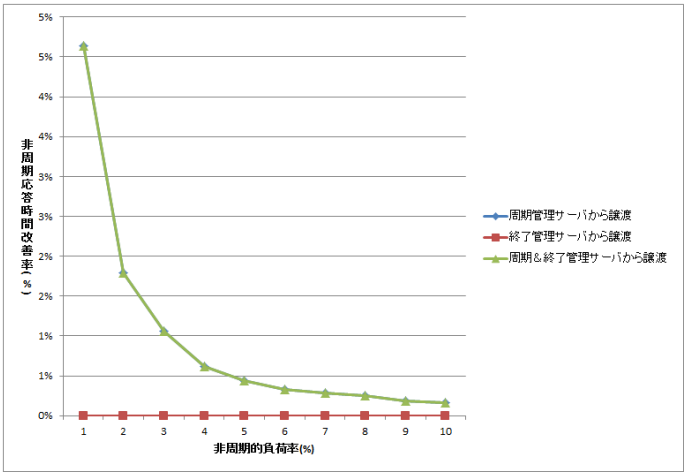

図 4.37: 周期的タスクの負荷が 65 %の時のアプリケーションの応答時間向上率の比較

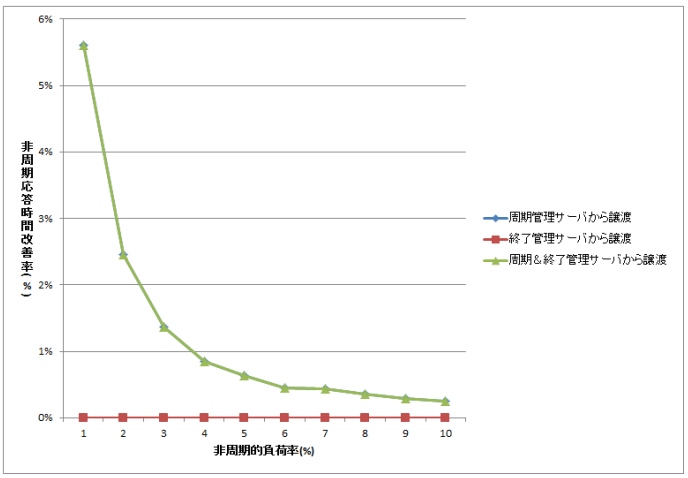

図 4.38: 周期的タスクの負荷が 70 %の時のアプリケーションの応答時間向上率の比較

タスク切り替えのオーバヘッドが 2 処理ティックの場合のグラフを示す。

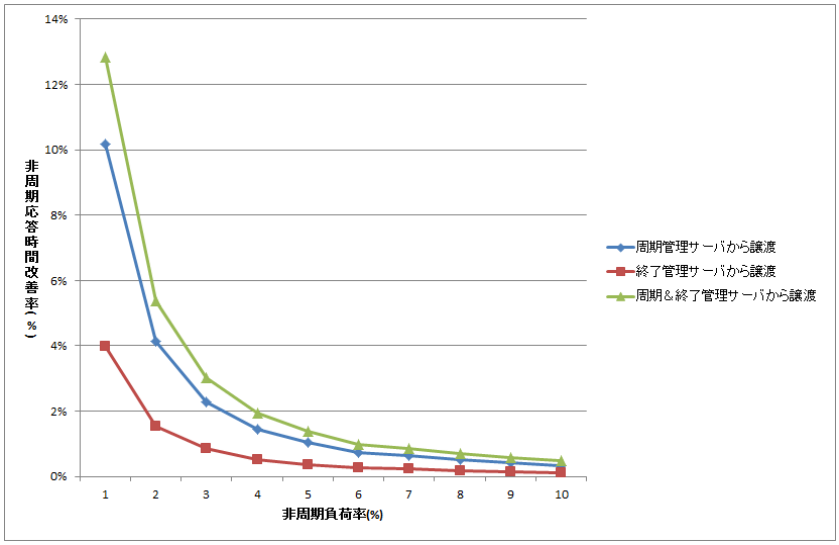

図 4.39: 周期的タスクの負荷が 50 %の時のアプリケーションの応答時間向上率の比較

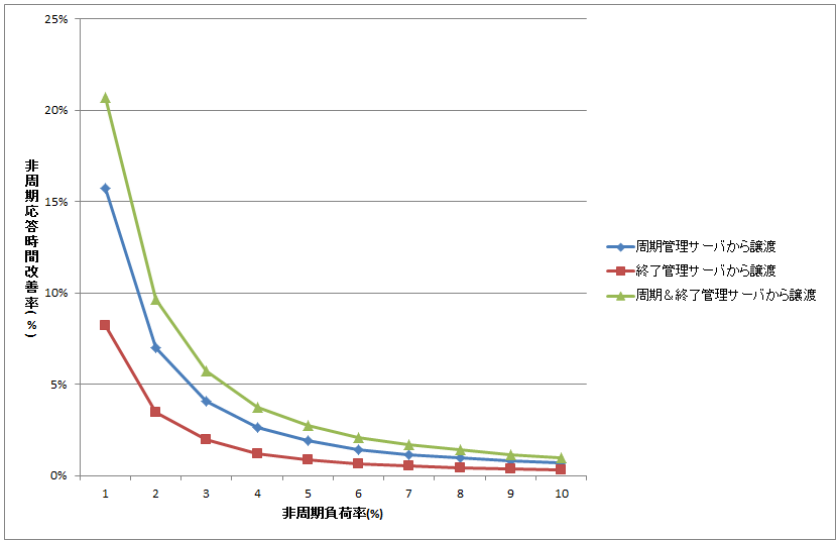

図 4.40: 周期的タスクの負荷が 55 %の時のアプリケーションの応答時間向上率の比較

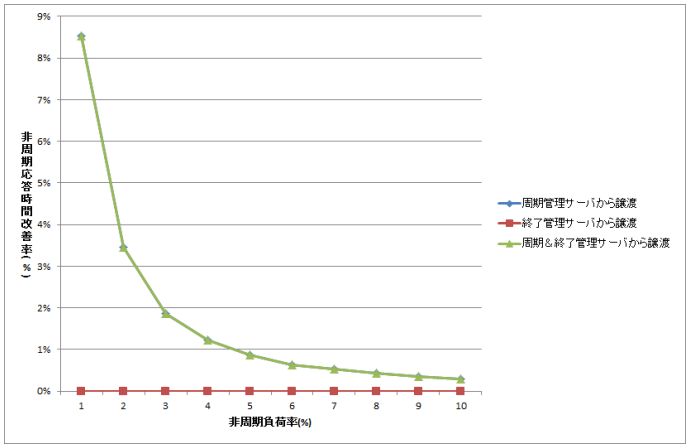

図 4.41: 周期的タスクの負荷が 60 %の時のアプリケーションの応答時間向上率の比較

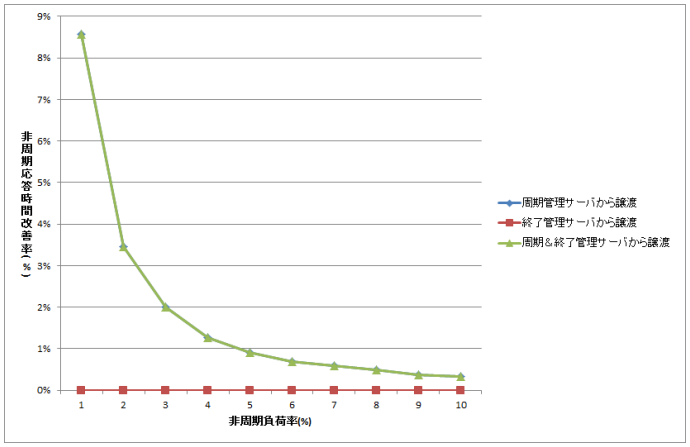

図 4.42: 周期的タスクの負荷が 65 %の時のアプリケーションの応答時間向上率の比較

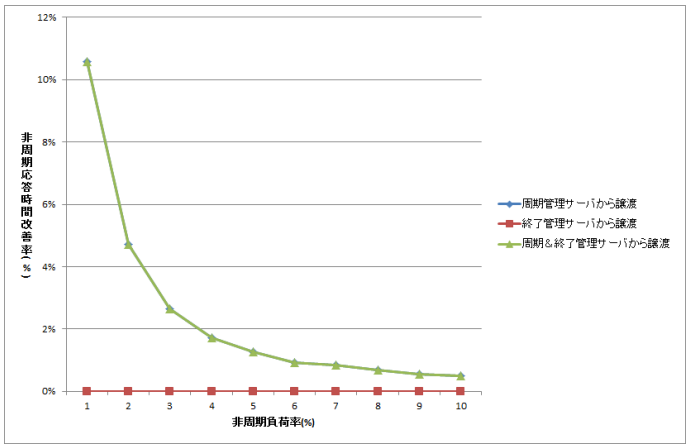

図 4.43: 周期的タスクの負荷が 70 %の時のアプリケーションの応答時間向上率の比較

タスク切り替えのオーバヘッドが 4 処理ティックの場合のグラフを示す。

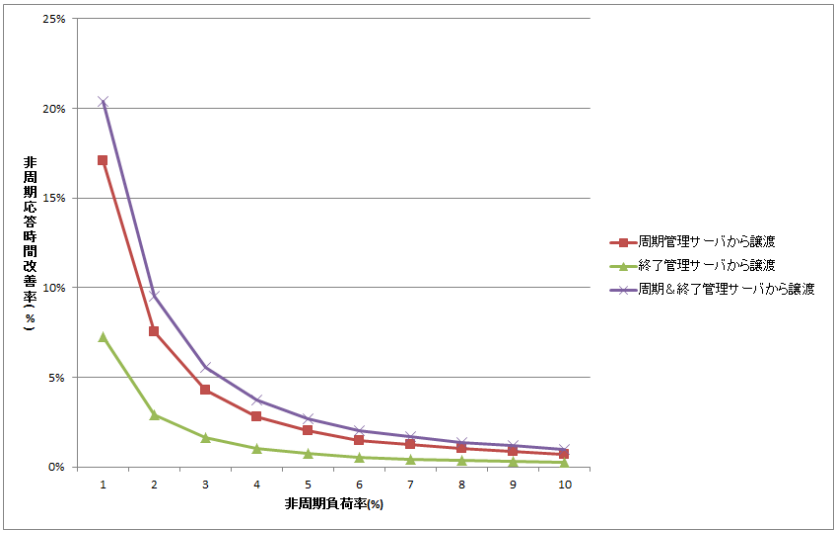

図 4.44: 周期的タスクの負荷が 50 %の時のアプリケーションの応答時間向上率の比較

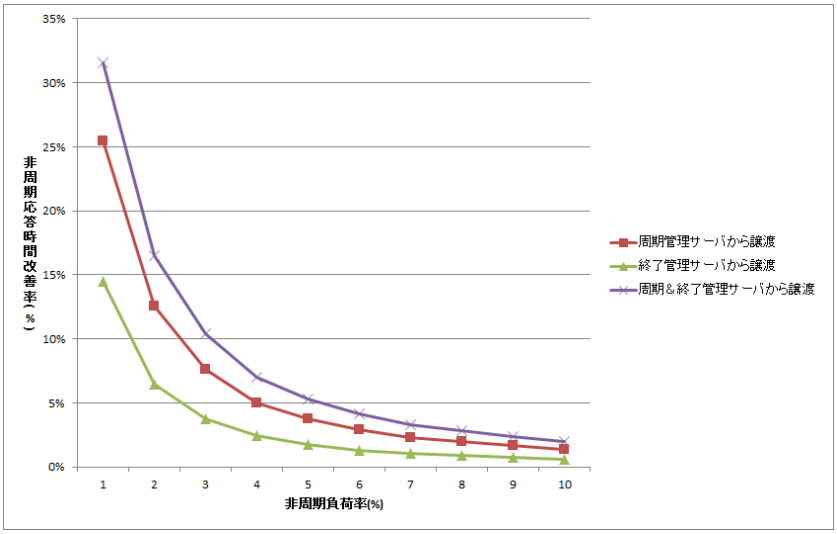

図 4.45: 周期的タスクの負荷が 55 %の時のアプリケーションの応答時間向上率の比較

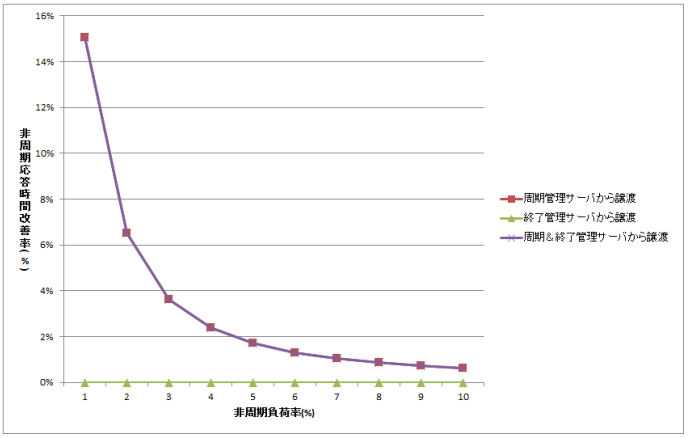

図 4.46: 周期的タスクの負荷が 60 %の時のアプリケーションの応答時間向上率の比較

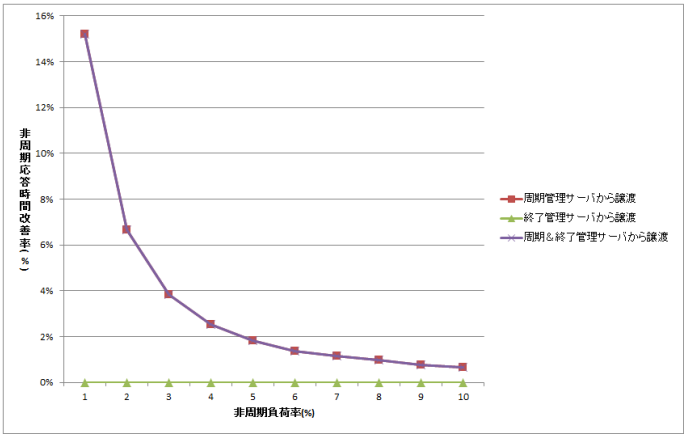

図 4.47: 周期的タスクの負荷が 65 %の時のアプリケーションの応答時間向上率の比較

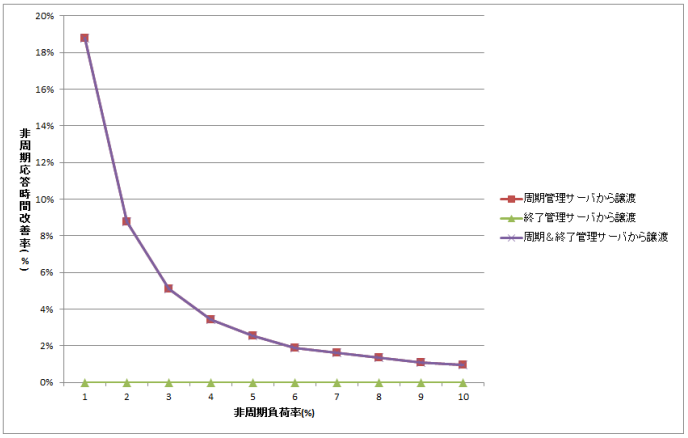

図 4.48: 周期的タスクの負荷が 70 %の時のアプリケーションの応答時間向上率の比較

4.3 結果考察

本研究の結果として、オーバヘッドの量に依存せずアプリケーション要求が低い程にSlackを再利用する効果が現れた。これは、今回のアプリケーションタスクの生成方法が、到着するアプリケーション要求の平均実行時間を長くすることで生成したためだと関係がある可能性があると考える。要求される総実行時間が長ければ長いほど、Slackを再利用することにより先に実行される時間は相対的に短くなるためだ。

周期管理サーバと終了管理サーバの性能向上への貢献は、周期管理サーバの方が終了管理サーバに比べて高い結果となった。特に終了管理サーバは周期タスクが高負荷の時に寄与の結果を確認するに至らなかった。これは周期的タスクセットを生成する際の都合も考えられる。今回扱ったタスクセットは全てRMで予め保証されているシステム負荷以上のセットは用いていない。3.3式で述べた様に、扱えるタスクセットはタスク数に相関が強くある。高いプロセッサー利用率を目指すほど、RMで保証されるタスクセットに含まれるタスク数が減りやすくなる。終了処理サーバから産まれるSlackはサーバの容量にも依存するので、タスクセットのタスク数が減ればSlackも容量も同時に減ってしまう。RMに保証される範囲の都合により、終了処理サーバの寄与が著しく下がってしまったと考えられる。

オーバヘッドの量に注目してそれぞれの結果を比較すると、オーバヘッドの値が大きくなるに比例して効果が大きくなることが分かる。オーバヘッド別による全ての周期タスクを用いた応答時間向上率の平均値は、1処理ティックの場合は1.49 %、2処理ティックの場合は2.79 %だった。しかし、オーバヘッドを4処理ティックにすると4.98 %が応答時間向上率の平均値になる。最大の効果が現れた条件は、周期的タスクが55 %でアプリケーションの負荷が1 %の時でアプリケーション応答時間の改善率は31.56 %に達した。これは短い周期でシステムを管理する高精度の割り込みを用いる組み込み機器になればなるほど効果が大きく見込めることを示す。

第5章　まとめ

本研究では、RTOS の機能であるタスク切り替えの処理をオーバヘッドとして扱いスケジューリングに組み込む方法を提案した。また、スケジューリングに組み込む際に過大に見積もった量を有効利用するための方法も提案した。これらを用いることで、何も管理しない方式に比べ、アプリケーションの応答時間を最大 31.56 %短縮させることができた。タスク切り替えの回数がシステムの中でも頻繁におこると想定される高精度の組み込み機器である程、効果が期待できることが分かった。

今後の課題として、本研究は単純化のためにスケジュール可能性を議論する題材に周期タスクのみを用いる方法を主に検討してきた。より現実のシステムに近づけるため、スケジュール可能性の保証の範囲が扱う対象として非周期的要求も許容することが求められる。

他にも、今回の提案方式によりスケジューリングに組み込んだ RTOS オーバヘッドは、比較的周期的要素の強いタスク切り替え処理のみであった。RTOS にはタスク切り替え以外にも多くの機能があるため、本研究で RTOS のオーバヘッドをスケジューリングに組み込めたとは言えない。特に、割り込みハンドラ等の非周期的要素が非常に強い要求に対してもスケジューリングに組み込む手法を提案できた場合、システム開発が更に容易になると考えられる。

第6章　謝辞

本論文を作成するにあたり、最後まで適切な助言を賜り熱心に指導してくださりました田中清史先生に感謝いたします。

同田中研究室の先輩の齋藤好宗さんと後輩の森本恵一さんには、実験の際等に適切な助言を賜りました。ここに感謝の意を表します。田中研究室、金子研究室の多くの方々に支えられて本論文ができました。本当に、ありがとうございます。

参考文献

[1] G.C.Buttazz, "Hard Real-Time Computing Systems: Predicatable Scheduling Algorithms and Applications", 3rd edition, Springer, 2011.

[2] C.L.Liu, J.W.Layland, " Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment ", Journal of the Association for Computing Machinery, Vol. 20, No. 1, pp. 46–61, January 1973.

[3] J.P.Lehoczky, L.Sha, J.K.Strosnider, " Enhanced Aperiodic Responsiveness in Hard Real-Time Environments ", Proc. of IEEE Real-Time Systems Symposium, pp. 261–270, December, 1987.