適応型スケジューリング方式の
実アプリケーションによる評価と改良

北陸先端科学技術大学院大学
情報科学研究科

森本 恵一

平成 28 年 3 月

修 士 論 文

適応型スケジューリング方式の
実アプリケーションによる評価と改良

1410042　森本 恵一

主指導教員　田中 清史
審査委員主査　田中 清史
審査委員　井口 寧
　　金子 峰雄

北陸先端科学技術大学院大学

情報科学研究科

平成 28 年 2 月

概 要

近年の組込みシステムの多種多様化により，周期タスクのスケジューラビリティを保証しつつ，非周期タスクの応答時間を短縮するためのリアルタイムタスクスケジューリング方式が求められている．過去の研究において，Total Bandwitdh Server（TBS）[2] のデッドライン計算の際に，最悪実行時間（WCET）の代わりに予測した実行時間（PET）を用いることで非周期タスクの応答時間を短縮させるスケジューリング方式，Adaptive TBS（ATBS）が提案された [3]．しかし，その実行時間の予測方法が最適とはいえず，十分な応答時間の削減が実現されていない．また，シミュレーションによる評価が行われたが，評価に使用されたタスクは乱数によって生成されたものであり，実際のシステムにおける提案手法の有用性を示せていない．

本研究では，ATBS の性能を向上させるための 2 つの手法，Adaptive TBS Modified（ATBSM）と，ATBSM+dwcet を提案する．ATBSM は，ATBS において実行時間の予測を高精度化することによって，性能を向上させる手法である．対象アプリケーションごとに実行時間の予測式を生成し使用することで，高精度な実行時間予測を実現する．ATBSM+dwcet は，PET が実際の実行時間よりも短くなった (過小見積となった) 場合の影響を小さくすることによって，性能を向上させる手法である．ATBS および ATBSM では過小見積となった場合，WCET を使用したデッドラインに切り替えるため応答時間が短縮がされない．ATBSM+dwcet では，実行時間と比例関係にある要因の値に応じた段階的な最悪実行時間（discrete WCET : dwcet）を WCET の代わりに使用することで，過小見積時の影響を小さくする．

組込み向けベンチマーク集である MiBench[4] を対象タスクとし，提案手法による性能の向上をシミュレーションによって評価した．MiBench の各アプリケーションを非周期タスクおよび周期タスクとして扱い，非周期タスクの平均応答時間と周期タスクの絶対ジッタおよび相対ジッタを評価の対象とした．評価に使用した全アプリケーションの平均値において，非周期タスクの応答時間は，ATBS と比較した際に，ATBSM によって最大 30.2%，ATBSM+dwcet によって最大 31.3%短縮された．周期タスクの絶対ジッタは，TBS と比較した際，ATBSM によって最大 15.1%，ATBSM+dwcet によって最大 15.9%短縮された．また，実行時間が長いインスタンスに対してより高精度に実行時間予測を行うことでジッタの短縮を狙う予測式を提案し，ATBSM と ATBSM+dwcet に適用したところ，両方ともジッタ短縮用の予測式によって絶対ジッタの短縮率が増加した．相対ジッタにおいても，絶対ジッタと同様の傾向が得られた．

目 次

第 1 章　はじめに **1**
1.1 背景 1
1.2 目的 2
1.3 タスク実行に関する説明 3
1.4 論文の構成 3

第 2 章　既存の研究 **4**
2.1 Earliest Deadline First（EDF） 4
2.1.1 EDF によるスケジューリングの例 4
2.2 Total Bandwidth Server（TBS） 5
2.2.1 TBS によるスケジューリングの例 6
2.3 Adaptive Total Bandwidth Server（ATBS） 7
2.3.1 予測した実行時間 (Predicted Execution Time：PET) 7
2.3.2 ATBS の定義 7
2.3.3 実装の複雑性 8
2.3.4 ATBS によるスケジューリングの例 8

第 3 章　提案手法 1：ATBSM **11**
3.1 PET 導出に使用する実行時間の予測式の生成 11
3.1.1 実行時間予測に使用する要因（予測要因）の検出 11
3.1.2 予測式生成のための事前実行 12
3.1.3 予測式の生成 12
3.1.4 実際のアプリケーションにおける予測式の生成の例 13
3.2 ATBSM における PET の導出 15
3.3 ATBSM によるスケジューリングの例 15

第 4 章　提案手法 2：ATBSM+dwcet **17**
4.1 段階的な最悪実行時間（dwcet）の定義 17
4.2 ATBSM+dwcet によるスケジューリング例 18

第 5 章　MiBench **20**
5.1 Automotive and Industrial Control 20

5.2 Consumer Devices 22
5.3 Office Automation 25
5.4 Network 25
5.5 Security 26
5.6 Telecommunications 27

第 6 章 評価 **31**
6.1 評価 1：非周期タスクの平均応答時間 31
6.1.1 評価方法 31
6.1.2 結果 32
6.2 評価 2：周期タスクのジッタ 39
6.2.1 ジッタ短縮用の予測式について 39
6.2.2 評価方法 39
6.2.3 結果（絶対ジッタ） 40
6.2.4 結果（相対ジッタ） 47

第 7 章 まとめ **53**
7.1 結論 53
7.2 今後の課題 54

第1章　はじめに

1.1　背景

近年，組込みシステムの多種多様化によって1つのシステム内に周期タスクと非周期タスクが混在することが一般的になっている．例として自動車のシステムを1つのプロセッサで処理する場合が考えられる．エンジン制御といった一定の周期で処理が必要となるシステムのタスクや，カーオーディオといった動作するまでに多少の遅延があっても重大な問題が起こらないシステムのタスクが混在している．前者のタスクはハードデッドラインを持ち，デッドラインを満たせなかった（デッドラインミスとなった）場合に重大な損失を被るタスクである．そのため必ずデッドライン以内に処理を完了しなければならない．後者のタスクはデッドラインを持たず，リアルタイム処理が求められないタスクである．可能な限り応答時間 (タスクの起動要求から全ての処理が完了するまでの時間) を短縮することが望まれる．このような異なるタスクが混在するタスクセットを扱うため，周期タスクのデッドラインを守りつつ非周期タスクの応答時間を短縮するためのリアルタイムタスクスケジューリング方式が求められている．

周期タスクと非周期タスクが混在するタスクセットをスケジューリングするためのアルゴリズムの1つとして，Total Bandwidth Server(TBS)[2] がある．TBS は Earliest Deadline First(EDF)[1] をベースとするスケジューリング方式であり，デッドラインが近いタスクを優先して実行する．デッドラインを持たない非周期タスクには，非周期インスタンスの実行時間を使用してデッドラインが与えられる．TBS は CPU 使用率の合計が 100%を超えないようにデッドラインを決定することで全タスクのデッドラインを守る (スケジューラビリティを保証する)．そのために，非周期インスタンスの実行時間としてシステム実行前に見積もられた最悪実行時間 (Worst-Case Execution Time : WCET) を使用することが必須である．

現在のプロセッサとプログラムの複雑性から WCET の見積りは悲観的にならざるを得ず，実際のシステム実行時における実行時間は WCET より短くなる場合が一般的である．これは TBS のような非周期インスタンスの実行時間に基づいて優先順位を決定するスケジューリング方式において，最適なスケジュールを得ることへの障害となる．そこで，非周期タスクの応答時間を短縮するために，WCET の代わりに予測した実行時間 (Predicted Execution Time : PET) を使用してデッドラインを決定するスケジューリング方式，Adaptive Total Bandwidth Server（ATBS）[3] が提案された．ATBS は非周期インスタンスのデッドライン計算において，WCET の代わりに PET を使用することで非周期

インスタンスのデッドラインを短くする．結果として，非周期インスタンスの優先順位が高くなりやすくなり，応答時間の短縮に繋がる．加えて，PET が実際の実行時間よりも短くなった（過小見積となった）場合，WCET を使用してデッドラインの再計算を行うことでスケジューラビリティを保証している．この研究においてシミュレーションによる評価が行われ，TBS よりも非周期タスクの応答時間が短縮されることが実証されてきた．

過去の研究において，ATBS の PET は過去の非周期インスタンスの実行時間に基づいた値が使用されていた．このような実行時間の予測方法では入力によって実行時間が大幅に変化するタスクに対して予測が困難であり，十分な応答時間の削減が実現できない．幅広いアプリケーションに対してリアルタイム性能を向上させるためには，高精度に実行時間を予測する方法が必要である．また，シミュレーションで使用された各インスタンスの実行時間は乱数によって生成されたものであり，この実行時間が実際の組込みシステムで使用される各アプリケーションの実行時間と同様の性質を持つとはいえない．従って，実際のシステムにおけるスケジューリングの有用性を示すためには，実際のアプリケーションの実行時間を反映した上での評価が必要である．

1.2 目的

本研究は，実アプリケーションを対象とし，過去に提案されたスケジューリング方式 ATBS における PET の高精度化によるリアルタイム性能の向上を目的とする．

本研究では ATBS において実行時間の予測を高精度化した手法，Adaptive TBS Modified（ATBSM）を提案する．過去に提案されたスケジューリング方式で使用された実行時間の予測方法は実アプリケーションに対して最適とはいえず，十分なリアルタイム性能の向上が実現できていない．対象アプリケーションごとに実行時間の予測式を作成し使用することによって高精度で実行時間を予測する方法を提案する．しかし，必ずしも実行時間を正確に予測し過小見積の数を 0 にできるとは限らない．ATBS において過小見積となった場合，WCET を使用してデッドラインを再計算するためデッドラインが長くなる．そこで，過小見積時において WCET の代わりに，実行時間に比例する要因の値に応じた段階的な最悪実行時間（discrete WCET：dwcet）を使用することで過小見積となった際の影響を小さくする手法，ATBSM+dwcet を提案する．この 2 つの提案手法と既存のスケジューリング方式をシミュレーションによって比較し，非周期タスクの応答時間と周期タスクの絶対ジッタおよび相対ジッタによってリアルタイム性能を評価する．

シミュレーションにおいて，組込みシステムで使用されているアプリケーションの実行時間を用いて評価を行うことで，その実用上の性能を示す．評価に使用する実行時間を得るために，本研究では組込み向けベンチマーク集である MiBench[4] を使用する．これは，組込み向け分野で用いられるアプリケーションの多様性を反映するために，36 個の組込み向けアプリケーションを 6 つのカテゴリに分類して提供するものである．

1.3　タスク実行に関する説明

本論文で使用するタスク実行に関する図および用語の説明を行う．図 1.1 はタスクの 1 回のインスタンスの起動要求から実行終了までを示す．上向きの矢印は起動要求時刻を示し，インスタンスの起動要求が発生した時刻を示す．インスタンスは起動要求時刻の後，スケジューラのアルゴリズムに基づいて優先順位が決められ，その優先順位に基づいて実行が開始される．プロセッサがインスタンスを終了するために必要な時間を実行時間とする．起動要求時刻からインスタンスを終えるまでの時間を応答時間とする．下向きの矢印はデッドライン時刻を示す．周期タスクにおいて起動要求時刻からデッドライン時刻までの時間は周期と等しいとする．そのため，周期タスクにおいて起動要求時刻とデッドライン時刻が重なる．絶対ジッタは計測期間内にて実行された全インスタンスにおける最長応答時間と最短応答時間の差を示す．相対ジッタは計測期間内において，前後のインスタンスの応答時間の差の最大値を示す．

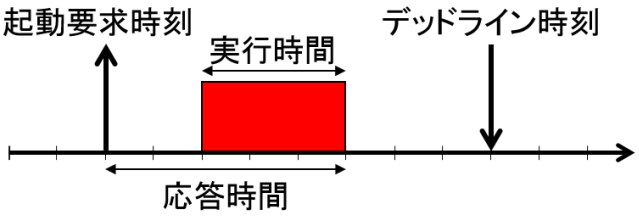

図 1.1: タスク実行についての例

1.4　論文の構成

本論文は 7 章から構成される．本章では本研究の背景と目的，および本論文において使用する図と用語の説明について述べた．2 章では，過去に提案されたの 3 つのスケジューリング方式，EDF，TBS，ATBS について述べる．それぞれのスケジューリングの方式におけるアルゴリズムと特徴，および具体的な例を示してスケジューリングの動作を説明する．3 章では，本研究で提案する手法，ATBSM について述べる．ATBSM は ATBS において PET を高精度化する手法であり，どのようにして PET の高精度化を行うのかを説明する．4 章では，本研究で提案する 2 つめの手法，ATBSM+dwcet について述べる．ATBSM+dwcet は過小見積となった際の影響を小さくするための手法であり，この手法のアルゴリズムについて説明する．5 章では，次章の評価にて使用する組込み向けベンチマーク集である MiBench アプリケーションについて述べる．各アプリケーションの概要および評価で使用する実行時間の予測式について説明する．6 章では，シミュレーションを用いた提案手法の評価の方法および結果について述べる．各スケジューリング方式における非周期タスクの平均応答時間および周期タスクの絶対・相対ジッタの結果を説明し，考察を述べる．最後に，7 章にて本研究から得られた結論と今後の課題について述べる．

第2章　既存の研究

本章では，過去に提案された 3 つのリアルタイムタスクスケジューリング方式である，EDF，TBS，ATBS について説明する．

2.1　Earliest Deadline First（EDF）

Earliest Deadline First（EDF）は動的スケジューリングアルゴリズムの 1 つである．すなわち，EDF はシステムの稼働中にタスクの優先順位を変化させる．タスクの優先順位はデッドラインによって決められ，デッドラインが近いタスクほど優先順位が高くなる．あるタスクの実行中によりデッドラインが近いタスクの起動要求が発生した場合は，前者のタスクの実行を中断し後者のタスクの実行を開始する．

EDF の特徴は，スケジューラビリティを保証しながら周期タスクセットの CPU 使用率を最大 100%にできることである．全周期タスクの CPU 使用率の合計を U_p とした場合，以下の条件式を満たすシステムでは全てのタスクのデッドラインが守られることが保証される．

$$U_p \leq 1 \tag{2.1}$$

n 個の周期タスクから構成されるタスクセットの U_p は以下の式によって求められる．

$$U_p = \sum_{i=1}^{n} \frac{C_i}{T_i}$$

ここで，C_i はタスク i の 1 回の実行時間，T_i はタスク i の周期である．

2.1.1　EDF によるスケジューリングの例

EDF によるスケジューリングの例を図 2.1 に示す．この図は表 2.1 に示す 2 つの周期タスクから構成されるタスクセットを EDF によってスケジューリングした結果である．このタスクセットにおける U_p は

$$U_p = \frac{2}{4} + \frac{3}{10} = \frac{16}{20} = 0.8$$

であり，式 (3.1) の条件を満たすためスケジューラビリティが保証される．

時刻 0 のとき，$\tau 1$ と $\tau 2$ の 1 番目の実行の起動要求が同時に発生する．このときデッドラインは $\tau 2$ の 1 番目の実行よりも $\tau 1$ の 1 番目の実行の方が近いため，$\tau 1$ の 1 番目の実行が開始する．時刻 2 のとき，$\tau 1$ の 1 番目の実行が終了し，待機していた $\tau 2$ の 1 番目の実行が開始．時刻 4 のとき，$\tau 1$ の 2 番目の実行の起動要求が発生する．このときデッドラインは $\tau 2$ の 1 番目の実行よりも $\tau 1$ の 2 番目の実行の方が近いため，$\tau 2$ の 1 番目の実行は中断され，$\tau 1$ の 2 番目の実行が開始する．時刻 6 のとき，$\tau 1$ の 2 番目の実行が終了し，中断されていた $\tau 2$ の 1 番目の実行が再開する．このように，EDF はデッドラインが近いタスクの優先順位を高くし，優先順位に従って実行するタスクを切り替える．

表 2.1: 周期タスクセット

タスク名	周期	実行時間
$\tau 1$	4	2
$\tau 2$	10	3

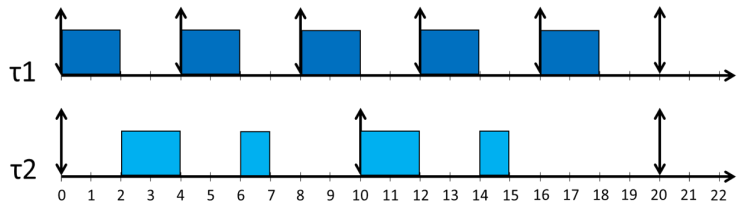

図 2.1: EDF によるスケジューリングの例

2.2 Total Bandwidth Server（TBS）

TBS はハードデッドラインを持つ周期タスクとデッドラインを持たない非周期タスクが混在するタスクセットをスケジューリングするために提案された方式の 1 つである．EDF アルゴリズムをベースとする動的優先度サーバであるため，スケジューラビリティを保証しながらタスクセットの CPU 使用率を最大 100%にできる．加えて，TBS は非周期タスクの応答時間性能が比較的高く，かつ複雑な実装を必要としないため実用性の高い方式である．

TBS は周期タスクと非周期タスクが混在するタスクセットを EDF によってスケジューリングするために，デッドラインを持たない非周期タスクに仮のデッドラインを与える．非周期タスクの k 番目のインスタンスのデッドライン時刻 d_k は以下の式によって与えられる．

$$d_k = max\left(r_k, d_{k-1}\right) + \frac{C_k^{WCET}}{U_s} \tag{2.2}$$

ここで，r_k は非周期タスクの k 番目のインスタンスの起動要求時刻，d_{k-1} は非周期タスクの $k-1$ 番目のインスタンスのデッドライン時刻，C_k^{WCET} は非周期タスクの k 番目のインスタンスの WCET，U_s は非周期タスクの実行を管理するサーバに割り当てられた CPU 使用率（バンド幅）である．d_0 は 0 とする．非周期タスクのインスタンスの起動要求が発生するたびに，そのインスタンスに U_s のバンド幅が割り当てられる．このバンド幅を基にデッドライン時刻が計算される．式 (2.2) の右辺の第 1 項は，連続する 2 つのインスタンスにおいて，それぞれのインスタンスに割り当てられたバンド幅が重ならないように調整する．式 (2.2) の右辺の第 2 項は，バンド幅に応じたデッドラインの長さを計算する．

TBS では式 (2.2) によって非周期タスクに仮のデッドライン時刻が与えられ，その後全ての周期タスクと非周期タスクが EDF アルゴリズムによってスケジュールされる．TBS は $U_s + U_p \leq 1$ の条件を満たすときスケジューラブルであることが証明されている．そのため，U_s は最大 $1 - U_p$ まで使用できる．

2.2.1 TBS によるスケジューリングの例

TBS によるスケジューリングの例を図 2.2 に示す．表 2.1 に示す 2 つの周期タスクが存在し，$U_p = 0.8$ となる．また，$U_s = 1 - U_p = 1 - 0.8 = 0.2$ である．時刻 2 のとき，非周期タスクの 1 番目のインスタンスの起動要求が発生する．このインスタンスの WCET と実際の実行時間はそれぞれ，$C_1^{WCET} = 4$，$C_1^{ET} = 2$ である．このインスタンスのデッドライン時刻 d_1 は，

$$d_1 = r_1 + \frac{C_1^{WCET}}{U_s} = 2 + \frac{4}{0.2} = 22 \tag{2.3}$$

となる．時刻 7 のとき，非周期インスタンス以外に待機しているインスタンスがないため非周期インスタンスを開始する．この非周期インスタンスは時刻 16 に終了し，その応答時間は $16 - 2 = 14$ となる．

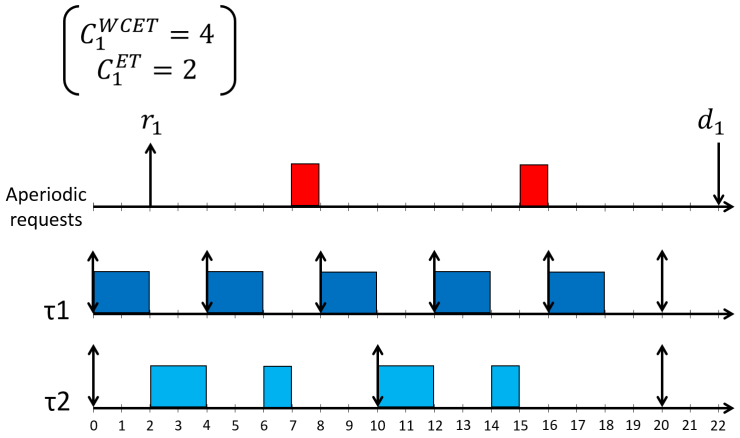

図 2.2: TBS によるスケジューリングの例

2.3 Adaptive Total Bandwidth Server（ATBS）

TBS において全タスクのスケジューラビリティを保証するためには，非周期タスクのデッドライン計算において非周期タスクの WCET を使用することが必須である．現在のプロセッサとプログラムの複雑性から WCET の見積りは悲観的にならざるを得ず，実際のシステム実行時における実行時間は WCET より短くなる場合が一般的である．TBS は式 (2.2) によって非周期タスクのデッドラインを計算するため，非周期タスクの WCET が過大に見積もられている場合，過大見積の分だけデッドラインが長くなる．そのため，EDF アルゴリズムによって非周期タスクの優先順位が低くなり，応答時間が長くなる場合が増加する．

ATBS は非周期タスクの応答時間を短縮するために，TBS のデッドライン計算において，非周期タスクの WCET の代わりに予測した実行時間 (Predicted Execution Time：PET) を使用する．予測した実行時間が経過しても非周期タスクの実行が終了しなかった (過小見積となった) 場合は，WCET を使用してデッドラインの再計算を行い，残りの実行のスケジュールを行う．この方法によって，ハードデッドラインを持つ周期タスクのスケジューラビリティを保証しつつ，かつ非周期タスクの実行が予測した実行時間以内に終了した場合はより早いデッドライン時刻の効果から応答時間が短縮される．

2.3.1 予測した実行時間 (Predicted Execution Time：PET)

ATBS において，非周期タスクのデッドライン計算のために実行時間の予測が必要となる．過去の研究 [3] では，以下の方法によって実行時間の予測を行った．

$$C_{i_k}^{PET} = \alpha \times C_{i_{k-1}}^{PET} + (1-\alpha) \times C_{i_{k-1}}^{ET} \tag{2.4}$$

ここで，$C_{i_k}^{PET}$ は非周期タスク J_i の k 番目のインスタンスの PET，$C_{i_{k-1}}^{ET}$ は同じタスクの $k-1$ 番目のインスタンスの実際の実行時間である．$C_{i_0}^{PET}$ は非周期タスク J_i の WCET とする．この式は，α を重み係数とし，$k-1$ 番目の PET と $k-1$ 番目の実際の実行時間との重み付き平均から k 番目の非周期インスタンスの PET が導出されることを示す．対象タスクの実行時間の変化に従う PET を導出するために，そのタスクの過去の実行時間を使用している．

2.3.2 ATBS の定義

ATBS において，非周期タスクのインスタンスを 2 つのインスタンスに分割する．それらを異なるインスタンスとみなすことで，TBS と同じ手法でスケジュールを行える．

以下の説明では，非周期タスクは区別されず（式 (2.4) の i を無視して），起動要求の順番による通し番号 k を持つとする．k 番目の非周期タスクのインスタンス J_k を，J_k^{PET} と J_k^{REST} に分割する．J_k^{PET} は J_k の開始から予測された終了時刻までの実行に対応する．

J_k^{REST} は予測された終了時刻から後の実行に対応する．すなわち，J_k が予測された終了時刻までに終了する場合，J_k^{REST} は存在しない．ここで，J_k の最悪実行時間を C_k^{WCET}，J_k の PET を C_k^{PET} とし，J_k^{REST} の実行時間 C_k^{REST} を以下のように計算する．

$$C_k^{REST} = C_k^{WCET} - C_k^{PET} \tag{2.5}$$

時刻 r_k に k 番目の非周期タスクの起動要求が発生したとき，2 つのインスタンスのデッドライン時刻は以下のように計算される．

$$d_k^{PET} = max\left(r_k, d_{k-1}\right) + \frac{C_k^{PET}}{U_s} \tag{2.6}$$

$$d_k^{REST} = d_k^{PET} + \frac{C_k^{REST}}{U_s} \tag{2.7}$$

TBS の場合におけるデッドライン時刻 d_k は式 (2.2) によって計算される．式 (2.5)，(2.6)，(2.7)，(2.2) より，$d_k^{REST} = d_k$ となる．従って，2 つのインスタンスのデッドラインは，非周期タスクの起動要求が発生した際に式 (2.6) と式 (2.2) によって計算できる．

また，2 つのインスタンスによる $max\left(r_k, d_{k-1}\right)$ と d_k の間での使用率が U_s と等しくなることから，ATBS と TBS が同じスケジューラビリティを持つことが証明されている．

2.3.3 実装の複雑性

TBS と比較したとき，ATBS は非周期タスクのデッドライン計算において PET を使用する点，および PET を動的に計算する点が異なる．加えて，非周期タスクのインスタンスを 2 つのインスタンスに分割する点が異なる．しかし，オペレーティングシステムはタスクを 1 つの情報セット（タスク制御ブロック）で管理するべきである．これは，PET が経過してもタスクの実行が終了していない場合に，デッドラインを再設定し，レディキューにタスクを再挿入することによって実現できる．実行時間が PET を経過したかどうかを検出するためには，全ティックのタイミングでスケジューラを実行する必要がある．タイマー/ティック割り込みの発生時にスケジューラを呼び出すことによって実現できるが，これはオペレーティングシステムが通常行う自然な手続きである．

2.3.4 ATBS によるスケジューリングの例

ATBS によるスケジューリングの例を図 2.3 と図 2.4 に示す．図 2.3 は非周期タスクの 1 番目の実行が PET 以内に終了した場合，図 2.4 は PET 以内に終了せず過小見積となった場合の結果を示している．2 つの図において，表 2.1 に示す 2 つの周期タスクが存在し，$U_p = 0.8$ となる．また，$U_s = 1 - U_p = 1 - 0.8 = 0.2$ である．時刻 2 のときに非周期タスクの 1 番目のインスタンスの起動要求が発生し，$C_1^{WCET} = 4$，$C_1^{ET} = 2$ とする．

（1）PET 以内に実行が終了した場合

1 番目の非周期インスタンスの PET を $C_1^{PET} = 3$ とする．このインスタンスの d_1^{PET} は，

$$d_1^{PET} = r_1 + \frac{C_1^{PET}}{U_s} = 2 + \frac{3}{0.2} = 17$$

となる．時刻 7 のとき，非周期インスタンスを開始する．時刻 14 のとき，$\tau 2$ の 2 番目のインスタンスよりも非周期インスタンスのデッドラインの方が近いため非周期インスタンスを再開する．この非周期インスタンスは時刻 15 に終了し，その応答時間は $15 - 2 = 13$ となる．これは図 2.2 に示した TBS の場合よりも短く，WCET より短い実行時間を使用したデッドラインを与えることによって応答時間が短縮される場合があることを示す．

（2）過小見積となった場合

1 番目の非周期インスタンスの PET を $C_1^{PET} = 1$ とする．このインスタンスの d_1^{PET} は，

$$d_1^{PET} = r_1 + \frac{C_1^{PET}}{U_s} = 2 + \frac{1}{0.2} = 7$$

となる．d_1^{REST} は式 (2.2) によって計算できるため，以下のようになる．

$$d_1^{REST} = r_1 + \frac{C_1^{WCET}}{U_s} = 2 + \frac{4}{0.2} = 22$$

時刻 3 のとき，非周期インスタンスの実行時間として PET が経過したがインスタンスは終了していない．そのため，デッドライン時刻を d_1^{PET} から d_1^{REST} に切り替え，残りのインスタンスのスケジュールを行う．この非周期インスタンスは時刻 16 に終了し，その応答時間は $16 - 2 = 14$ となる．これは図 2.2 に示した TBS の場合と等しく，ATBS において過小見積が発生した場合は TBS と等しい応答時間になることが分かる．

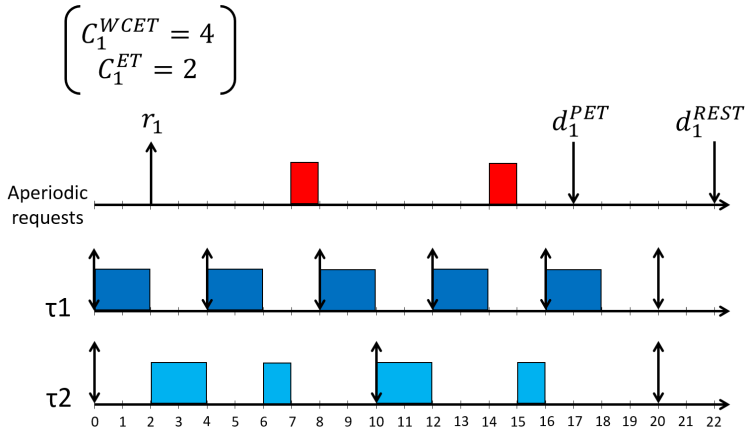

図 2.3: ATBS において PET 以内に実行が終了した場合の例

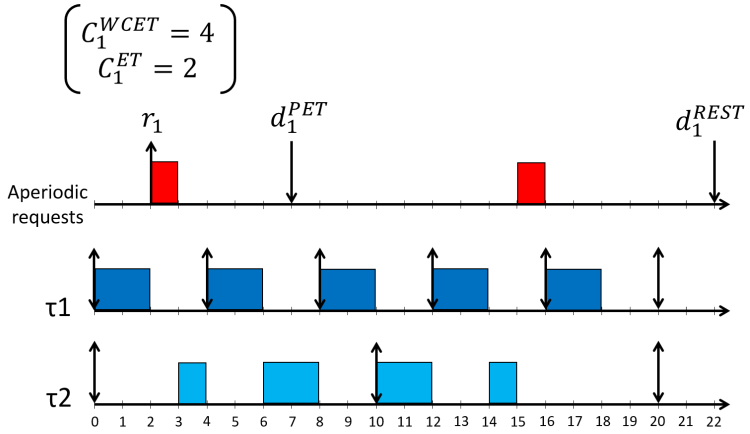

図 2.4: ATBS において過小見積となった場合の例

第3章　提案手法1：ATBSM

ATBS は，TBS における非周期タスクのデッドラインの計算において WCET の代わりに PET を使用する．非周期タスクにより早いデッドラインを与えることで優先順位が高くなりやすくなり，非周期タスクの応答時間の短縮を実現する．過去の研究では，PET として過去の非周期タスクの実行時間に基づく値が使用されていた．この実行時間の予測方法は，実行時間のばらつきが小さいアプリケーションに対して有効であるが，入力によって実行時間が大幅に変化するアプリケーションに対して予測が困難である．PET が実際の実行時間よりも過大に見積もられた場合は，十分にデッドラインを短くできず，十分な応答時間の短縮は見込めない．また過小見積となった場合は,（仮の）デッドラインミスが発生し，WCET を使用したデッドラインに切り替えて残りの実行をスケジュールするため，応答時間の短縮は見込めない．幅広いアプリケーションに対してリアルタイム性能を向上させるために，高精度に実行時間を予測する方法が必要である．

ATBS において実行時間の予測を高精度化した手法，Adaptive TBS Modified（ATBSM）を提案する．まず，システム開発者が対象アプリケーションに対して，様々な入力による実行時間の変化を調べることで実行時間の予測式を生成する．ATBSM はこの予測式を使用することで高精度な実行時間予測を実現する．実行時間の予測式はタスクの入力を変数とするため，入力によって実行時間が大幅に変化するアプリケーションに対して有効である．PET をより実際の実行時間に近づけることで，非周期タスクのデッドライン時刻が早まり，応答時間が短縮される．

3.1　PET 導出に使用する実行時間の予測式の生成

ATBSM において，システム開発者が対象アプリケーションごとの実行時間の予測式をシステム稼働前に用意する必要がある．本節は，実行時間の予測式を生成するための手法とこの手法によって予測式を生成した例について述べる．

3.1.1　実行時間予測に使用する要因（予測要因）の検出

始めに，予測式の変数とする要因を検出する．すなわち，対象アプリケーションにおいてどのような要因から実行時間を予測できるか調査する．この実行時間の予測に使用する要因を予測要因とする．

予測要因の検出を行うために，対象アプリケーションに対して様々な入力を使用して実行時間の計測を行う．計測した実行時間より，実行時間と比例関係にある要因を見つけ出し，その要因を予測要因とする．予測要因として，入力データサイズなどが挙げられる．

3.1.2 予測式生成のための事前実行

次に，予測要因に対して対象アプリケーションがシステム稼働時に取りうるおおよその範囲を対象として実行時間の計測を行う．システム開発者にとって，システム稼働時に対象アプリケーションが取りうる予測要因のおおよその範囲は既知であるとする．

システム実行時に取りうる予測要因のおおよその範囲を均等間隔で 20 区間に分け，1 区間につき 10 個の入力による実行時間の計測を行う．合計 200 個の入力による実行時間の計測を行う．

3.1.3 予測式の生成

ATBS において，PET をより実際の実行時間に近づけるほど応答時間の短縮が見込める．過小見積となった場合，WCET を使用したデッドライン時刻となるため応答時間の短縮が見込めない．それゆえ応答時間を短縮するためには，PET と実際の実行時間の差を小さくし，かつ過小見積の数を少なくする予測式が有効となる．

上記のことを踏まえて，3.1.2 で計測した実行時間を使用して実行時間の予測式を生成する．予測式はスケジューラの計算量を可能な限り小さくするため，1 次関数とする．予測式は以下のアルゴリズムによって生成される．以下の説明において，事前実行にて得られた 200 個の計測結果を $(x_0, y_0), (x_1, y_1), ..., (x_k, y_k), ..., (x_{199}, y_{199})$ と示す．y_k は計測した実際の実行時間，x_k はその計測に使用した予測要因の値である．

I, 始めに，最小二乗法を使用して，事前実行にて得られた 200 個の計測結果から近似直線 $f(x) = a_0x + a_1$ を求める．すなわち，以下の式に示すように，誤差の二乗和である $\sum_{k=0}^{200}(y_k - (a_0x_k + a_1))^2$ が最小となる a_0 と a_1 を求める．

$$\sum_{k=0}^{200}(y_k - (a_0x_k + a_1))^2 \rightarrow min$$

Ⅱ, 次に，200 個全ての計測データにおいて，実際の実行時間 y_k と先ほど求めた近似直線 $f(x)$ から得られる予測実行時間 $f(x_k) = a_0x_k + a_1$ を比較し，過小見積となる数を数える．$y_k > f(x_k)$ の条件を満たすとき，過小見積とする．過小見積となる数が閾値以下になる場合，近似直線 $f(x)$ を予測式として決定し，予測式の生成を終了する．閾値は 65 とする．

Ⅲ, もし過小見積の数が閾値を超える場合は，閾値以下になる近似直線を生成する．まず，1 つ前の計算で生成された近似直線を $f^{pre}(x)$ とする．以下の式によって，新たに近似直線 $f(x) = a_0x + a_1$ を求める．

$$\sum_{k=0}^{200}(W_k \times (y_k - (a_0x_k + a_1))^2) \rightarrow min$$

$$W_k = \begin{cases} W_k + 0.1 & (y_k > f^{pre}(x_k)) \\ W_k & (otherwise) \end{cases}$$

ここで，W_k は誤差に付ける重みであり，W_k の初期値を 1 とする．$y_k > f^{pre}(x_k)$ の条件を満たすとき，y_k と $f(x_k)$ の誤差である $y_k - (a_0x_k + a_1)$ に付ける重みを増やす．過小見積となる誤差に 1 よりも大きな重みを付けることで，誤差の二乗和の最小値を計算する際に過小見積となる誤差の影響が大きくなる．それゆえ上記の式は，過小見積となる誤差をより小さくするための近似直線を生成し，その新たに生成された近似直線における過小見積の数が 1 つ前の計算で生成された近似直線の場合よりも減少することを狙っている．

新たな近似直線が生成された後，その近似直線を使用して過小見積の数を数える．もし閾値を超えるならば，新たに生成された近似直線を $f^{pre}(x)$ として，再計算を行う．この繰り返しにより少しずつ過小見積となる誤差の重みを増やすことで，最終的に過小見積の数が閾値以下となる予測式が生成される．

予測式の生成は，基本的に最小二乗法を使用して行われる．上述したように，ATBS において応答時間を短縮させるためには，PET と実際の実行時間の差を小さくし，かつ過小見積の数を少なくする予測式にするべきである．そのため，最小二乗法によって予測式を生成した後，過小見積の数が閾値よりも多い場合は再生成を行う．これは過小見積となる誤差の重みの増加および再生成された式における過小見積の数のカウントを繰り返すことによって行われる．また，この予測式生成のアルゴリズムは全て自動で行われる．

この予測式生成のアルゴリズムにおいて，事前実行にて得られた 200 個の計測結果から予測式が生成される．しかし，必ずしも 200 個全てを使用する必要はなく，目的に応じた予測式の生成を行うことができる．例えば，予測要因の値における上位 100 個の計測結果のみを使用することによって，予測要因の値が大きいときにより高精度に予測を行える式が生成される．

3.1.4 実際のアプリケーションにおける予測式の生成の例

MiBench ベンチマーク集における Automotive カテゴリの susan-corners アプリケーションを対象として，予測式の生成の例を示す．MiBench についての詳細は後述する．susan-corners アプリケーションは画像内のオブジェクトのコーナーを検出するプログラムであ

り，画像ファイルを入力とする．実行時間の計測に使用した環境は，Intel Xeon E3-1280 3.6GHz，Linux 2.6.32 である．実際の組込みシステムに適用する場合を想定したとき，使用される組込みプロセッサとの性能差を考慮する必要がある．上記の環境と組込みプロセッサの性能を比較した際，同一のタスクの実行において上記の環境による実行時間は組込みプロセッサの場合よりも 10 倍短くなると仮定する．このため，組込みプロセッサにおける 1 ティックを 1ms とし，上記のプロセッサを使用した際の実行時間の計測では 0.1ms を 1 ティックとして扱った．

始めに，予測要因の検出の例を示す．対象アプリケーションにおいて予測要因の検出を行うために，45 個の入力を使用して実行時間の計測を行った．この 45 個の入力において，使用した入力データサイズが 3 種類あり，それぞれ 65KB，257KB，1025KB である．それぞれのサイズごとに 15 個ずつ入力を用意し，計 45 個となった．計測した実行時間を図 3.1 に示す．横軸が入力データサイズ，縦軸が実行時間を示す．図 3.1 から，対象アプリケーションにおいて入力データサイズと実行時間が (粗い) 比例関係にあることが分かるので，予測要因を入力データサイズとした．

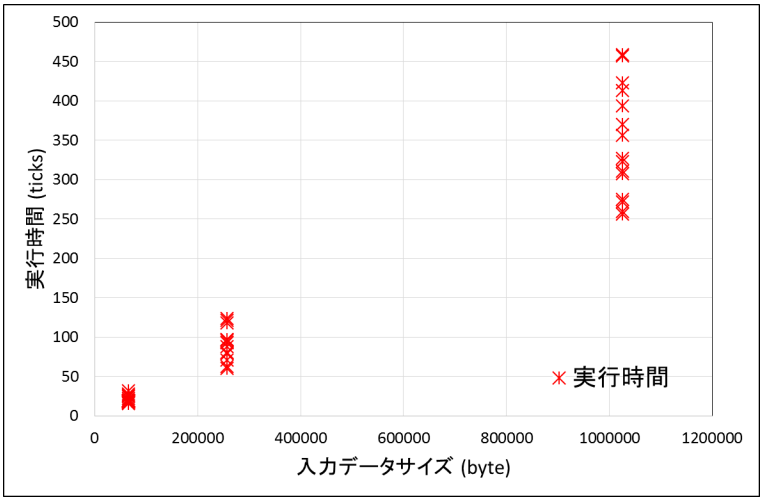

図 3.1: 予測要因の検出における実行時間

次に，事前実行と予測式の生成の例を示す．事前実行で得られた実行時間とその実行時間から生成された予測式を図 3.2 に示す．横軸が入力データサイズ，縦軸が実行時間を示す．対象アプリケーションにおいて，システム実行時に取りうる入力データサイズのおおよその範囲が 4.25KB から 85KB までと仮定した．この範囲を均等間隔で 20 区間に分け，1 区間につき 10 個の入力を用意し，合計 200 個の入力を使用して実行時間の計測を行った．200 個の計測結果から最小二乗法を用いて予測式を生成した際，過小見積の数は 64 個となったため，再生成は行われなかった．

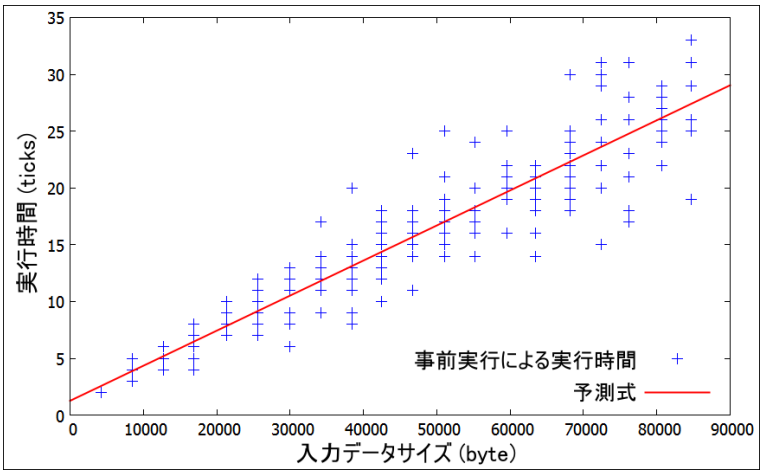

図 3.2: 事前実行による実行時間と得られた予測式

3.2 ATBSM における PET の導出

ATBSM では，非周期タスクの起動要求は常に予測要因の値と使用する予測式を指定するための番号を含むものと仮定する．k 番目の非周期インスタンスにおける入力の予測要因の値が $input_k$，使用する予測式の番号が $type_k$ であるとき，この非周期インスタンスの PET は以下の式によって求められる．

$$C_k^{PET} = a_0^{type_k} \times input_k + a_1^{type_k} \tag{3.1}$$

ここで，C_k^{PET} は k 番目の非周期インスタンスの PET，$a_0^{type_k}$ と $a_1^{type_k}$ は $type_k$ に対応する予測式の定数である．

ATBSM は非周期タスクの起動要求が発生した際，まず $type_k$ を参照して使用する予測式を選択する．その予測式と $input_k$ より，PET を導出する．

3.3 ATBSM によるスケジューリングの例

ATBSM によるスケジューリングの例を図 3.3 に示す．表 2.1 に示す 2 つの周期タスクが存在し，$U_p = 0.8$，$U_s = 1 - U_p = 1 - 0.8 = 0.2$ となる．表 3.1 は，スケジュール対象のタスクセットに含まれる全非周期タスク用の予測式を示す．時刻 2 のとき，$C_1^{WCET} = 4$，$C_1^{ET} = 2$ である 1 番目の非周期タスクのインスタンスの起動要求が発生する．また，この非周期インスタンスにおける予測要因の値が $input_1 = 1500$，使用する予測式の番号が $type_1 = 0$ である．表 3.1 より，予測式の 2 つの定数が $a_0^0 = 0.00155$，$a_1^0 = -0.39526$ となるため，PET は以下のようになる．ただし，整数にするために小数点以下を切り上げるとする．

$$C_1^{PET} = ceil(a_0^0 \times input_1 + a_1^0)$$

$$
\begin{aligned}
&= ceil(0.00155 \times 1500 + (-0.39526)) \\
&= ceil(1.92974) \\
&= 2
\end{aligned}
$$

この値は実際の実行時間と等しい．このインスタンスの d_1^{PET} は，

$$d_1^{PET} = r_1 + \frac{C_1^{PET}}{U_s} = 2 + \frac{2}{0.2} = 12$$

となる．周期タスクと非周期タスクのデッドライン時刻が等しい場合においては，周期タスクの優先順位が非周期タスクよりも高いとする．この非周期インスタンスは時刻 11 に終了し，その応答時間は $11 - 2 = 9$ となる．これは図 2.3 に示した ATBS の場合よりも短い．

ATBSM において，必ずしも図 3.3 に示す結果になるとは限らない．実行時間の予測の結果によっては，ATBS の例として示した，図 2.3 や図 2.4 と同じ結果になる．これは ATBS においても同様であり，正確に実行時間の予測を行うことができれば図 3.3 と同じ結果になる．ATBSM は，ATBS よりも実行時間予測を高精度に行うことで，図 3.3 に示す結果となる可能性を上げている．

表 3.1: 予測式の表

予測式の番号	a_0	a_1
0	0.00155	-0.39526
1	0.00031	1.26504
2	0.00003	0.93158

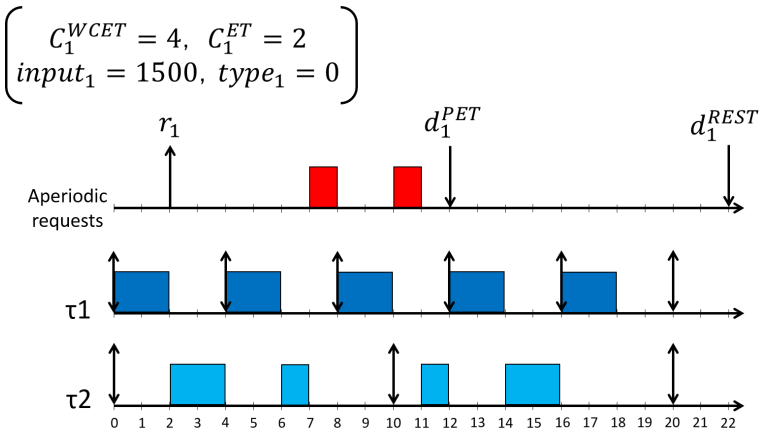

図 3.3: ATBSM によるスケジューリングの例

第4章　提案手法2：ATBSM+dwcet

ATBS において過小見積となった場合，WCET を使用したデッドラインに切り替えて残りの実行をスケジュールする．このとき，TBS の場合とデッドライン時刻が等しくなるため，応答時間の短縮が見込めない．ATBSM では，実行時間の予測式の生成の際に過小見積の数を減らす工夫を加えたが，あらゆるアプリケーションにおいて過小見積の数を 0 にできるとは限らない．1 回でも過小見積が発生したならば，リアルタイム性能の低下を引き起こす．

そこで，過小見積時の影響を小さくする手法を提案する．これは，過小見積時のデッドライン計算において，WCET の代わりに予測要因の値に応じた段階的な最悪実行時間（discrete WCET : dwcet）を使用することによって実現される．実行時間と比例関係にある要因を持つアプリケーションにおいて，予測要因の値に応じて最悪実行時間が変動する．このことから，予測要因の値に応じた最悪実行時間を一定の間隔ごとに用意しておき，過小見積時に予測要因の値に基づいた dwcet を選択しデッドライン計算に使用する．WCET を使用した場合よりもデッドラインが早くなり，過小見積時の影響が小さくなる．ATBSM において，過小見積時のデッドライン計算に dwcet を使用する手法を，ATBSM+dwcet とする．

4.1　段階的な最悪実行時間（dwcet）の定義

dwcet は，システム稼働時に対象アプリケーションが取りうる予測要因の範囲において，一部の範囲のみを対象としたときの最悪実行時間である．対象アプリケーションにおいて，予測要因の値が x_k 以下となるときの最悪実行時間を，$dwcet_k$ とする．システム稼働時において対象アプリケーションが取りうる予測要因の最大値を x_{MAX}，用意する dwcet の数を K 個とするとき，$x_k = k \times (x_{MAX}/K)$ とする．このとき，$dwcet_K = WCET$ となる．dwcet は非周期タスクの予測要因の値に従って選択され，過小見積時のデッドライン計算に使用される．このとき選択された dwcet が WCET より短い限り，過小見積時のデッドラインが早くなる．なお，最悪実行時間の見積りはシステム開発者によるものであり，本論文では扱わない．

入力データサイズが予測要因となるアプリケーションにおいて，5 段階の dwcet が用意されたときの例を図 4.1 に示す．横軸が入力データサイズ，縦軸が dwcet を示す．入力データサイズの一定の間隔ごとの最悪実行時間を見積り，それぞれを $dwcet_1$ から $dwcet_5$ とし

ている．図 4.1 において，非周期タスクの起動要求における入力データサイズが x_1 より大きく，かつ x_2 以下となるとき，$dwcet_2$ が過小見積時のデッドラインの計算に使用される．

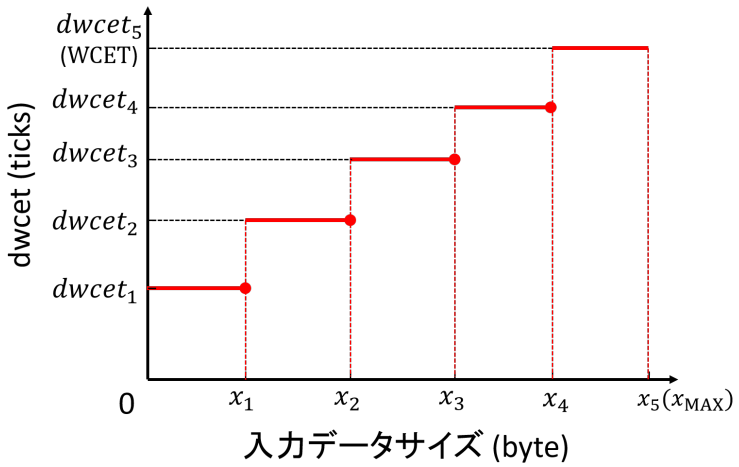

図 4.1: 入力データサイズが予測要因となるアプリケーションにおいて，5 段階の dwcet が用意された例

4.2 ATBSM+dwcet によるスケジューリング例

ATBSM+dwcet において過小見積が起きた際のスケジューリングの例を図 4.2 に示す．表 2.1 に示す 2 つの周期タスクが存在し，$U_p = 0.8$，$U_s = 1 - U_p = 1 - 0.8 = 0.2$ となる．各非周期タスクは表 3.1 を参照して，使用する予測式を決定する．時刻 2 のとき，$C_1^{WCET} = 4$，$C_1^{ET} = 2$ である 1 番目の非周期タスクのインスタンスの起動要求が発生する．また，この非周期インスタンスにおいて使用する予測式の番号が $type_1 = 0$，予測要因の値が $input_1 = 900$ である．表 3.1 より，予測式の 2 つの定数が $a_0^0 = 0.00155$，$a_1^0 = -0.39526$ となるため，PET は以下のようになる．ただし，整数にするために小数点以下を切り上げるとする．

$$\begin{aligned} C_1^{PET} &= ceil(a_0^0 \times input_1 + a_1^0) \\ &= ceil(0.00155 \times 900 + (-0.39526)) \\ &= ceil(0.99974) \\ &= 1 \end{aligned}$$

予測要因の値より，この非周期インスタンスの dwcet が $C_1^{dwcet} = 3$ になるとする．この値を過小見積時に使用するデッドライン d_1^{REST} の計算に使用する．このインスタンスの

d_1^{PET} と d_1^{REST} は，

$$d_1^{PET} = r_1 + \frac{C_1^{PET}}{U_s} = 2 + \frac{1}{0.2} = 7$$
$$d_1^{REST} = r_1 + \frac{C_1^{dwcet}}{U_s} = 2 + \frac{3}{0.2} = 17$$

となる．この非周期インスタンスは時刻 15 に終了し，その応答時間は $15 - 2 = 13$ となる．これは図 2.4 に示した，ATBS において過小見積となった場合よりも短い．

過小見積が発生しない場合，ATBSM と ATBSM+dwcet は同じ結果となる．

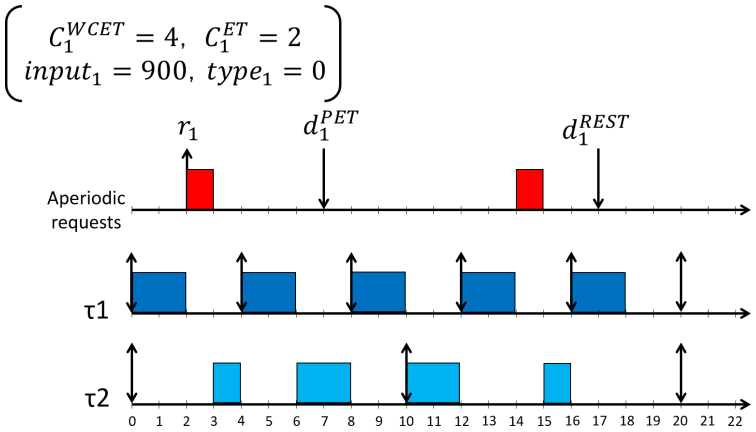

図 4.2: ATBSM+dwcet のスケジュールにおいて過小見積が起きた場合の例

第5章　MiBench

MiBench は無料の組込み向けベンチマーク集である．組込み分野で用いられるアプリケーションの多様性を反映するために，36 個[1] の組込みアプリケーションを 6 つのカテゴリに分類して提供している．6 つのカテゴリは，Automotive and Industrial Control，Consumer Devices，Office Automation，Network，Security，Telecommunications である．MiBench の全アプリケーションを表 5.1 に示す．全てのアプリケーションは標準の C 言語で記述されており，コンパイラのサポートを持つ任意のプラットフォームで使用できる．

本研究において，提案手法の評価のためにスケジューリングのシミュレーションを行う．そのシミュレーションに使用するタスクの実行時間として，MiBench の各アプリケーションの実際の実行時間を使用する．36 個のアプリケーションのうち，実行時間の計測に使用した環境 (Intel Xeon E3-1280 3.6GHz，Linux 2.6.32) で実行可能であり，かつ予測要因が検出できた 27 アプリケーションをシミュレーションに使用する．

本章の以下の節では，カテゴリごとに，シミュレーションで使用する 27 アプリケーションの概要および提案手法のために生成した予測式について述べる．また，本章の以下の節で示す全ての図は，各アプリケーションにおける (予測式生成のための) 事前実行で得られた実行時間とその実行時間から生成された予測式を示す．全ての図において，横軸が予測要因，縦軸が実行時間である．予測要因は各アプリケーションごとに異なる．

5.1　Automotive and Industrial Control

このカテゴリは，自動車や産業機器の制御システムにおいて使用されるアプリケーションを対象とする．それらのシステムにおいて使用される典型的なアプリケーションとして，エアーバッグの制御，エンジン性能の監視，物体の形状認識等が挙げられる．それらのアプリケーションの特徴を表現するために，このカテゴリでは基本的な演算，ビット操作，ソート，形状認識を行うプログラムを提供している．本研究の評価において，qsort，susan-edges，susan-corners，susan-smoothing アプリケーションを使用する．

qsort：クイックソートアルゴリズムを使用して，昇順にソートを行うアプリケーションである．ソートの対象は，データの位置を表現する 3 次元の座標データである．予測要

[1] 文献 [4] では 35 個としている．これは，文献 [4] における Consumer/jpeg アプリケーションを，本論文では Consumer/jpeg-enc. および Consumer/jpeg-dec. という 2 つのアプリケーションとして扱っているためである．

表 5.1: MiBench を構成する全アプリーション

Auto./Industrial	Consumer	Office	Network	Security	Telecomm.
basicmath	jpeg enc.	ghostscript	dijkstra	blowfish enc.	CRC32
bitcount	jpeg dec.	ispell	patricia	blowfish dec.	FFT
qsort	lame	rsynth		pgp sign	IFFT
susan-edges	mad	sphinx		pgp verify	ADPCM enc.
susan-corners	tiff2bw	stringsearch		rijndael enc.	ADPCM dec.
susan-smoothing	tiff2rgba			rijndael dec.	GSM enc.
	tiffdither			sha	GSM dec.
	tiffmedian				
	typeset				

因は入力データサイズ (ソート数) とした．事前実行による実行時間と生成された予測式を図 5.1 に示す．

susan-edges : 画像内のオブジェクトのエッジを検出するアプリケーションである．予測要因は入力データサイズ (画像ファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.2 に示す．

susan-corners : 画像内のオブジェクトのコーナーを検出するアプリケーションである．予測要因は入力データサイズ (画像ファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.3 に示す．

susan-smoothing : スムージング処理によって画像のノイズを除去するアプリケーションである．予測要因は入力データサイズ (画像ファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.4 に示す．

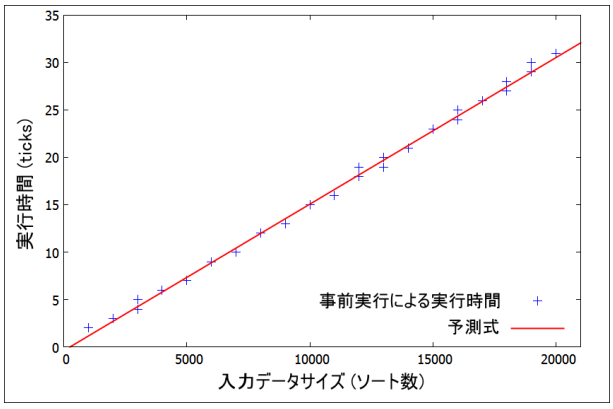

図 5.1: Aut./qsort

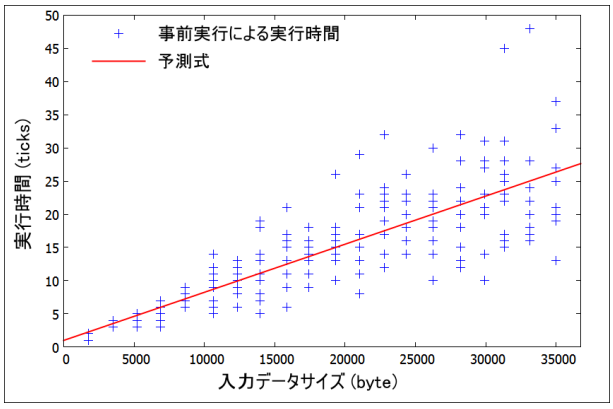

図 5.2: Aut./susan-edges

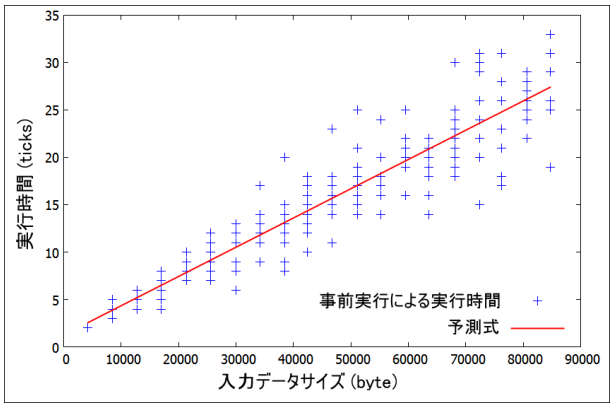

図 5.3: Aut./susan-corners

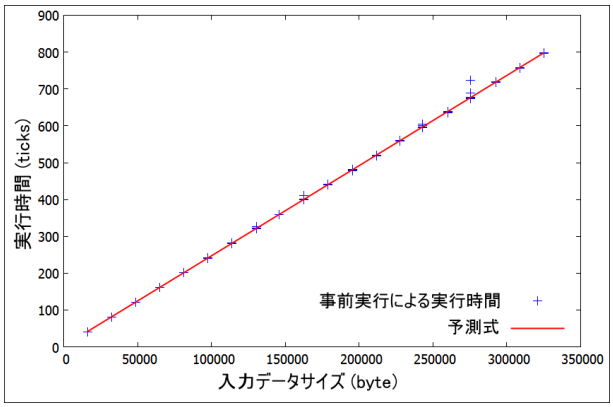

図 5.4: Aut./susan-smoothing

5.2 Consumer Devices

このカテゴリは，スキャナー，デジタルカメラ，携帯情報端末（PDA）等のコンシューマ向けの機器において使用されるアプリケーションを対象とする．典型的なアプリケーションはマルチメディアデータの処理を行うアプリケーションであり，jpeg 圧縮/復元，画像のカラーフォーマットの変換，ディザリング処理，カラーパレットの削減，MP3 圧縮/復号等を行うプログラムを提供している．本研究の評価において，jpeg-encode/decode, lame，mad，tiff2bw，tiff2rgba，tiff2dither，tiff2median アプリケーションを使用する．

jpeg-encode : 画像ファイルの圧縮を行うアプリケーションである．画像ファイルの形式を JPEG 形式に変換する．予測要因として，入力データサイズ (画像ファイルサイズ) と入力画像形式を使用する．入力データサイズのみでも実行時間の予測を行えるが，入力画像形式を考慮することでより予測の精度が上がるため，2 つの予測要因を用いることにした．使用する入力画像形式を BMP 形式と PPM 形式の 2 種類とし，それぞれの入力画像形式ごとに入力データサイズを変数とする予測式を生成した．事前実行による実行時間と生成された予測式を図 5.5 に示す．入力画像形式が BMP 形式の場合は予測式_1 を，PPM 形式の場合は予測式_2 を使用して実行時間の予測を行う．

jpeg-decode : 圧縮された画像ファイルの復元を行うアプリケーションである．JPEG 形式の画像ファイルを他の形式の画像ファイルに変換する．予測要因は，入力データサイズ (圧縮された画像ファイルサイズ) と出力画像形式とした．使用する出力画像形式を BMP 形式と PPM 形式の 2 種類とし，それぞれの出力画像形式ごとに入力データサイズを変数とする予測式を生成した．事前実行による実行時間と生成された予測式を図 5.6 に示す．出力画像形式が BMP 形式の場合は予測式_1 を，PPM 形式の場合は予測式_2 を使用して実行時間の予測を行う．

lame : 音声ファイルの圧縮を行うアプリケーションである．WAVE 形式の音声ファイルを MP3 形式に変換する．予測要因は，入力データサイズ (音声ファイルサイズ) と入力音声ファイルのサンプリングレートとした．使用する入力音声ファイルのサンプリングレートを 22.05kHz と 44.1kHz の 2 種類とし，それぞれのサンプリングレートごとに入力デー

タサイズを変数とする予測式を生成した．事前実行による実行時間と生成された予測式を図 5.7 に示す．入力音声ファイルのサンプリングレートが 22.05kHz の場合は予測式_1 を，44.1kHz の場合は予測式_2 を使用して実行時間の予測を行う．

mad : 圧縮された音声ファイルの復号を行うアプリケーションである．MP3 形式の音声ファイルを WAVE 形式に変換する．予測要因は，入力データサイズ (圧縮された音声ファイルサイズ) と入力音声ファイルのサンプリングレートとした．使用する入力音声ファイルのサンプリングレートを 22.05kHz と 44.1kHz の 2 種類とし，それぞれのサンプリングレートごとに入力データサイズを変数とする予測式を生成した．事前実行による実行時間と生成された予測式を図 5.8 に示す．入力音声ファイルのサンプリングレートが 22.05kHz の場合は予測式_1 を，44.1kHz の場合は予測式_2 を使用して実行時間の予測を行う．

tiff2bw : tiff 形式のカラー画像をグレースケール画像に変換するアプリケーションである．予測要因は入力データサイズ (画像ファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.9 に示す．

tiff2rgba : tiff 形式のカラー画像の色表現を，RGB カラーに変換するアプリケーションである．予測要因は入力データサイズ (画像ファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.10 に示す．

tiff2dither : tiff 形式のグレースケール画像にディザリング処理を行うアプリケーションである．画像の解像度とサイズを減らすために使用される．予測要因は入力データサイズ (画像ファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.11 に示す．

tiff2median : tiff 形式のカラー画像にメディアンフィルタ処理を行うアプリケーションである．画像のノイズ除去を行うために使用される．予測要因は入力データサイズ (画像ファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.12 に示す．

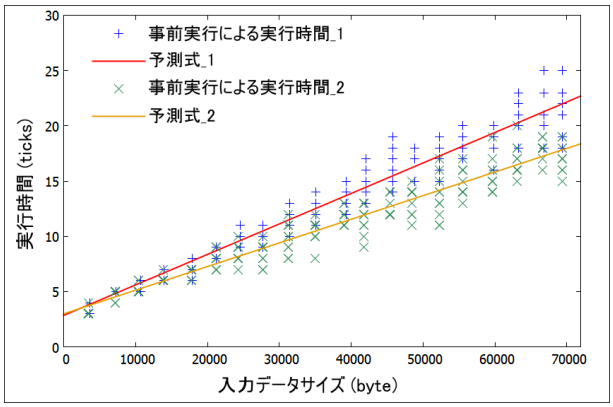

図 5.5: Con./jpeg-encode

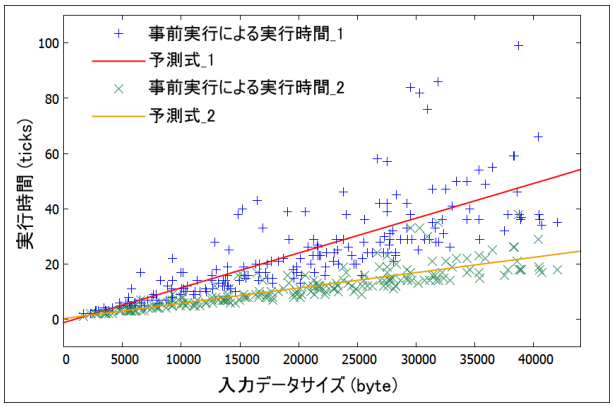

図 5.6: Con./jpeg-decode

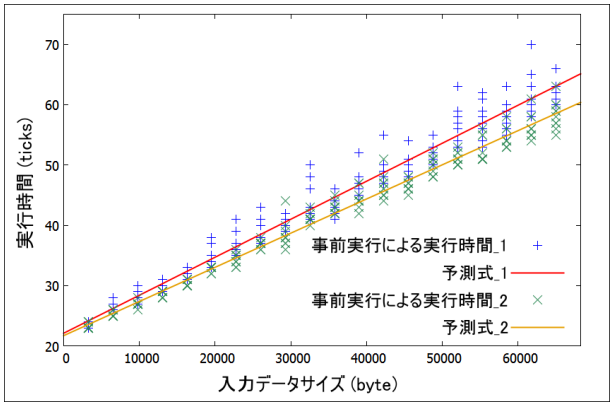

図 5.7: Con./lame

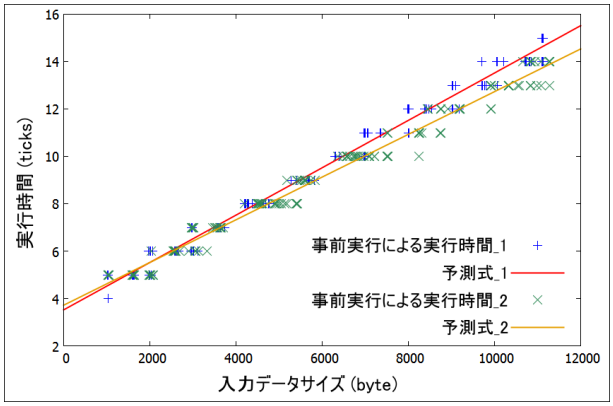

図 5.8: Con./mad

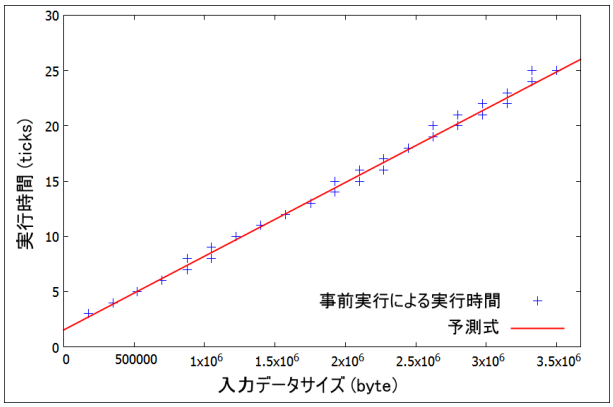

図 5.9: Con./tiff2bw

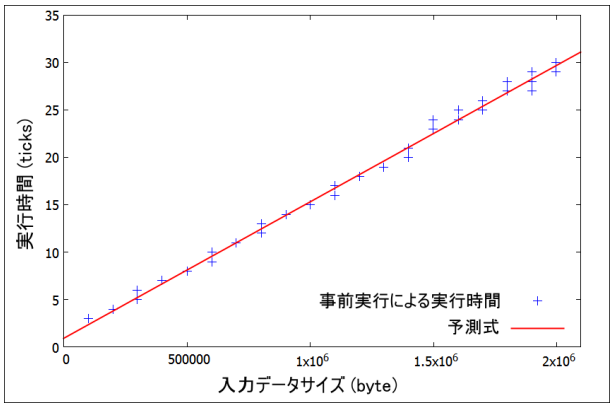

図 5.10: Con./tiff2rgba

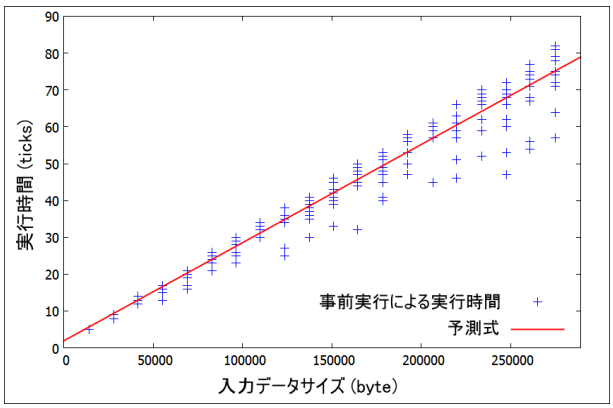

図 5.11: Con./tiff2dither

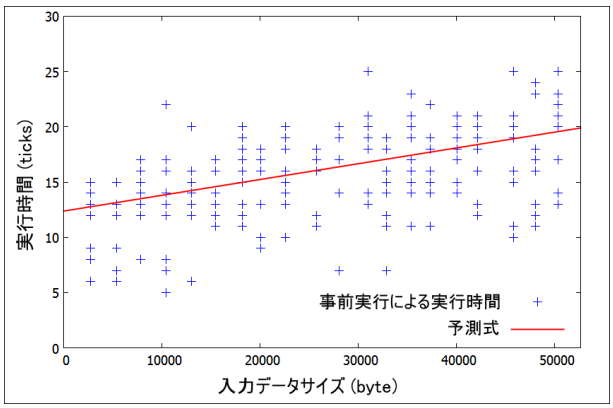

図 5.12: Con./tiff2median

5.3 Office Automation

このカテゴリは，プリンター，ファックス，ワードプロセッサ等の事務用機器において使用されるアプリケーションを対象とする．典型的なアプリケーションはテキスト操作に関するアプリケーションであり，Postscript ファイルの処理，文字列探索，スペルチェック，テキスト読み上げ等のプログラムを提供している．本研究の評価において，ghostscript，rsynth アプリケーションを使用する．

ghostscript：Postscript ファイルを画像ファイルに変換するアプリケーションである．予測要因は出力画像ファイルのページ数とした．出力ページ数は PostScript ファイルの先頭部分に記述されているため，先頭部分を読み取ることで取得できる．事前実行による実行時間と生成された予測式を図 5.13 に示す．

rsynth：テキスト音声合成を行うアプリケーションである．テキストファイルに記述されている文章を読み上げる．予測要因は入力データサイズ (テキストファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.14 に示す．

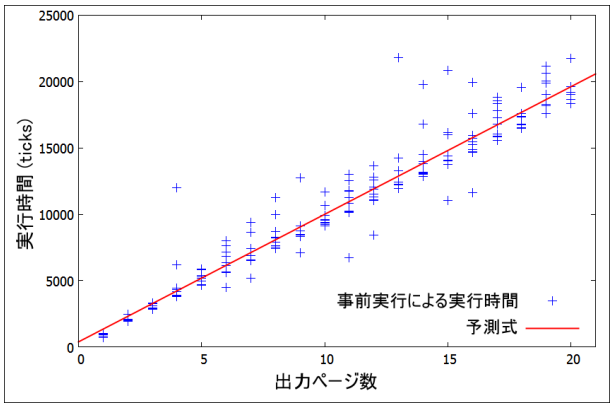

図 5.13: Off./ghostscript

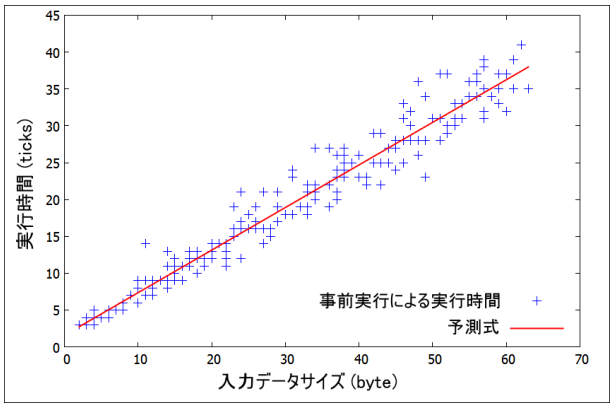

図 5.14: Off./rsynth

5.4 Network

このカテゴリは，スイッチやルータ等のネットワーク機器において使用されるアプリケーションを対象とする．グラフの最短経路探索，ルーティングテーブルの構築を行うプログラムを提供している．本研究の評価において，patricia アプリケーションを使用する．

patricia：パトリシアトライデータ構造の構築を行うアプリケーションである．パトリシアトライはネットワークアプリケーションにおけるルーティングテーブルとして使用されるデータ構造である．IP アドレスが記述されたテキストファイルを入力とする．予測要因は入力データサイズ (IP アドレス数) とした．事前実行による実行時間と生成された予測式を図 5.15 に示す．

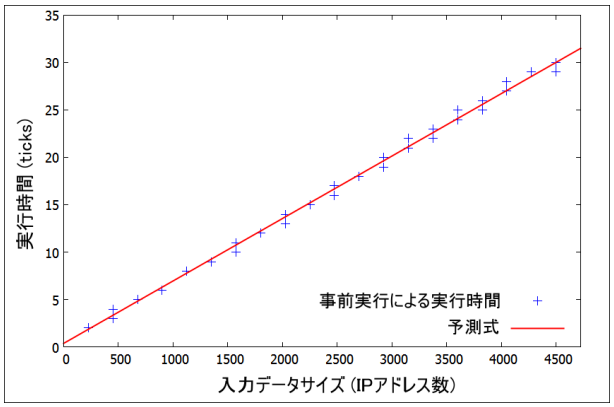

図 5.15: Net./patricia

5.5 Security

このカテゴリは，情報セキュリティのために使用されるアプリケーションを対象とする．データの暗号化/復号化，ハッシュ値生成を行うプログラムを提供している．本研究の評価において，blowfish-encrypt/decrypt，rijndael-encrypt/decrypt，sha アプリケーションを使用する．

blowfish-encrypt：Blowfish アルゴリズムを使用してテキストファイルの暗号化を行うアプリケーションである．Blowfish は，32bit から 448bit までの可変長の鍵を使用する対象ブロック暗号である．予測要因は入力データサイズ (テキストファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.16 に示す．

blowfish-decrypt：Blowfish アルゴリズムによって暗号化されたテキストファイルの復号化を行うアプリケーションである．予測要因は入力データサイズ (暗号化されたファイルのサイズ) とした．事前実行による実行時間と生成された予測式を図 5.17 に示す．

rijndael-encrypt：Rijndael アルゴリズムを使用してテキストファイルの暗号化を行うアプリケーションである．Rijndael は，鍵長とブロック長を 128，192，256bit の中から選択できるブロック暗号である．予測要因は，入力データサイズ (テキストファイルサイズ) と鍵長とした．使用できる鍵長は 3 種類であるため，それぞれの鍵長ごとに入力データサイズを変数とする予測式を生成した．事前実行による実行時間と生成された予測式を図 5.18 に示す．鍵長が 128bit の場合は予測式_1 を，192bit の場合は予測式_2 を，256bit の場合は予測式_3 を使用して実行時間の予測を行う．

rijndael-decrypt：Rijndael アルゴリズムによって暗号化されたテキストファイルの復号化を行うアプリケーションである．予測要因は，入力データサイズ (暗号化されたファイルのサイズ)) と鍵長とした．使用できる鍵長は 3 種類であるため，それぞれの鍵長ごとに入力データサイズを変数とする予測式を生成した．事前実行による実行時間と生成された予測式を図 5.19 に示す．鍵長が 128bit の場合は予測式_1 を，192bit の場合は予測式_2 を，256bit の場合は予測式_3 を使用して実行時間の予測を行う．

sha：ハッシュ値の計算を行うアプリケーションである．テキストファイルを入力とし，160bit のハッシュ値を求める。予測要因は入力データサイズ (テキストファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.20 に示す．

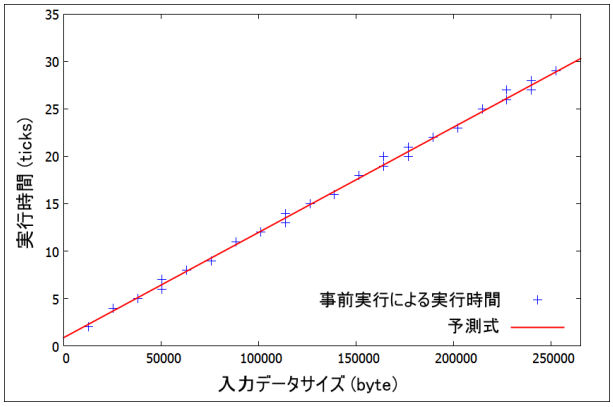

図 5.16: Sec./blowfish-encrypt

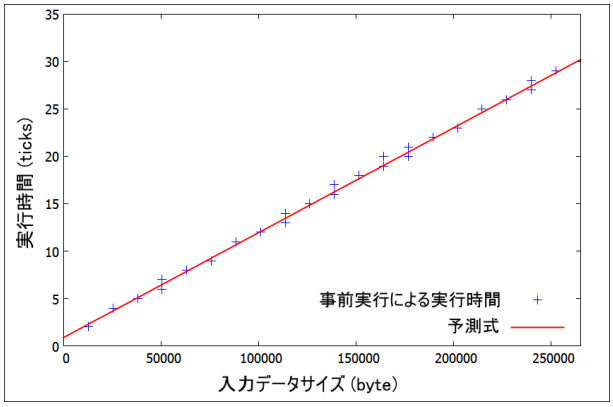

図 5.17: Sec./blowfish-decrypt

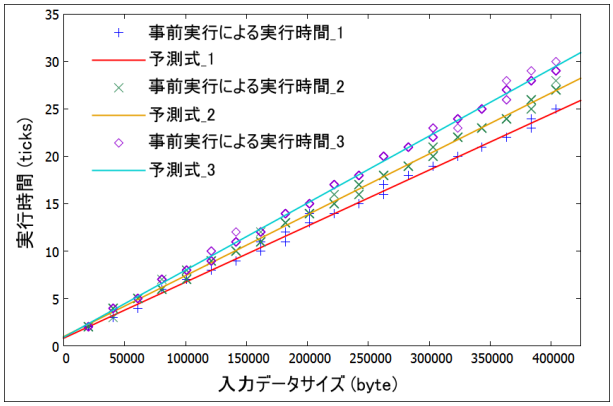

図 5.18: Sec./rijndael-encrypt

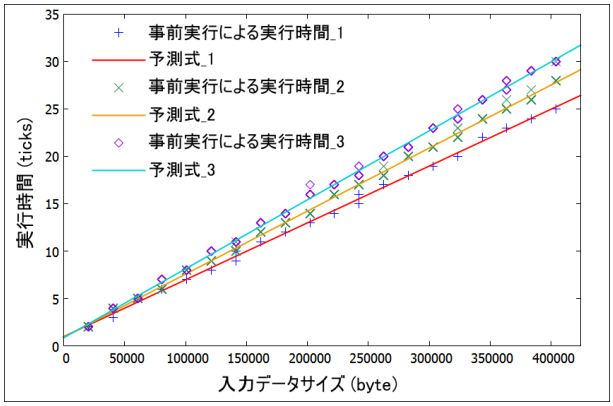

図 5.19: Sec./rijndael-decrypt

5.6 Telecommunications

このカテゴリは，無線通信機器において使用されるアプリケーションを対象とする．音声の符号化/復号化，周波数解析，誤り検出を行うプログラムを提供している．本研究の評価において，FFT，IFFT，ADPCM-encode/decode，GSM-encode/decode，CRC32 アプリケーションを使用する．

FFT/IFFT：高速フーリエ変換と逆フーリエ高速変換を行うアプリケーションである．フーリエ変換および逆フーリエ変換を行う波形の長さが入力値によって決定される．この

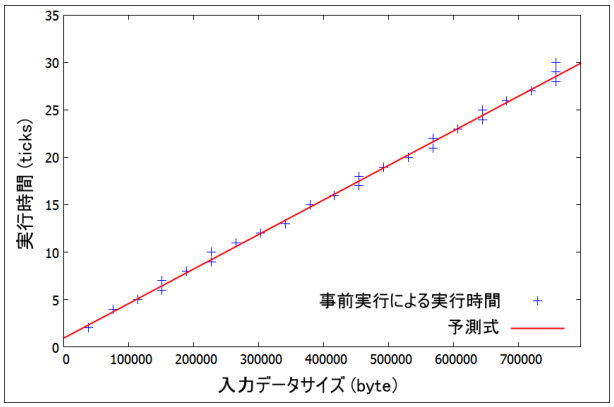

図 5.20: Sec./sha

入力値は 2 のべき乗でなければならない．2 つのアプリケーションにおいて，予測要因は入力値とした．事前実行による実行時間と生成された予測式を，図 5.21 と 5.22 に示す．

ADPCM-encode：PCM 形式の音声ファイルを ADPCM アルゴリズムによって圧縮するアプリケーションである．予測要因は入力データサイズ (音声ファイルサイズ) とする．事前実行による実行時間と生成された予測式を図 5.23 に示す．

ADPCM-decode：ADPCM アルゴリズムによって圧縮された音声ファイルを復号するアプリケーションである．予測要因は入力データサイズ (圧縮された音声ファイルサイズ) とする．事前実行による実行時間と生成された予測式を図 5.24 に示す．

GSM-encode：音声ファイルの圧縮を行うアプリケーションである．音声ファイルの形式を GSM 形式に変換する．予測要因は，入力データサイズ (音声ファイルサイズ) と入力音声形式とした．使用する入力音声形式を au 形式と linear 形式の 2 種類とし，それぞれの入力音声形式ごとに入力データサイズを変数とする予測式を生成した．事前実行による実行時間と生成された予測式を図 5.25 に示す．入力音声形式が au 形式の場合は予測式_1 を，linear 形式の場合は予測式_2 を使用して実行時間の予測を行う．

GSM-decode：圧縮された音声ファイルの復号を行うアプリケーションである．GSM 形式の音声ファイルを他の形式の音声ファイルに変換する．予測要因は，入力データサイズ (圧縮された音声ファイルサイズ) と出力音声形式とした．使用する出力音声形式を au 形式と linear 形式の 2 種類とし，それぞれの出力音声形式ごとに入力データサイズを変数とする予測式を生成した．事前実行による実行時間と生成された予測式を図 5.26 に示す．出力音声形式が au 形式の場合は予測式_1 を，linear 形式の場合は予測式_2 を使用して実行時間の予測を行う．

CRC32：CRC 値の計算を行うアプリケーションである．音声ファイルを入力とし，CRC 値を求める．予測要因は入力データサイズ (音声ファイルサイズ) とした．事前実行による実行時間と生成された予測式を図 5.27 に示す．

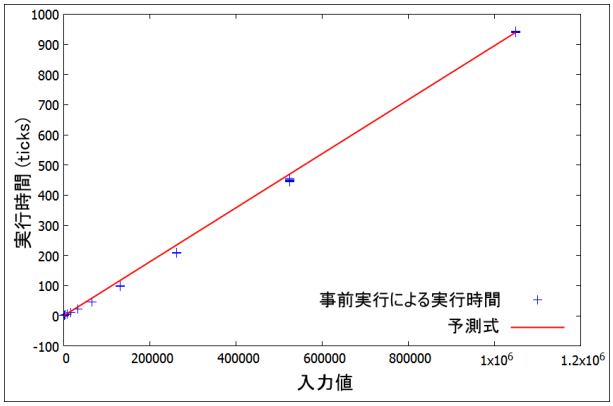

図 5.21: Tel./FFT

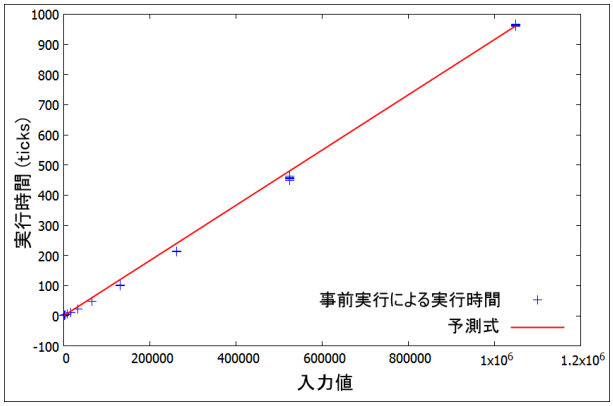

図 5.22: Tel./IFFT

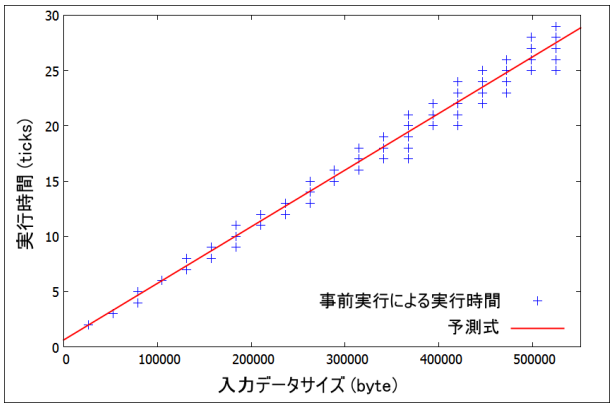

図 5.23: Tel./ADPCM-encode

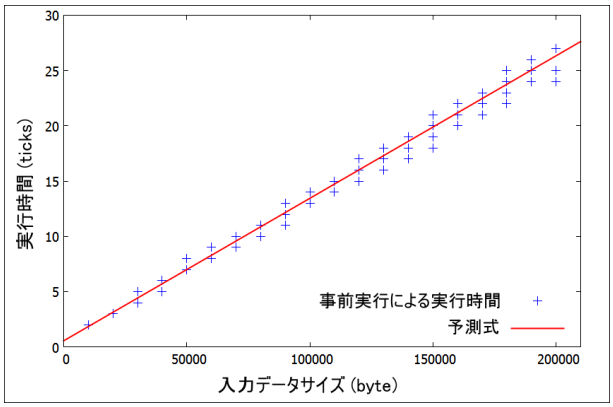

図 5.24: Tel./ADPCM-decode

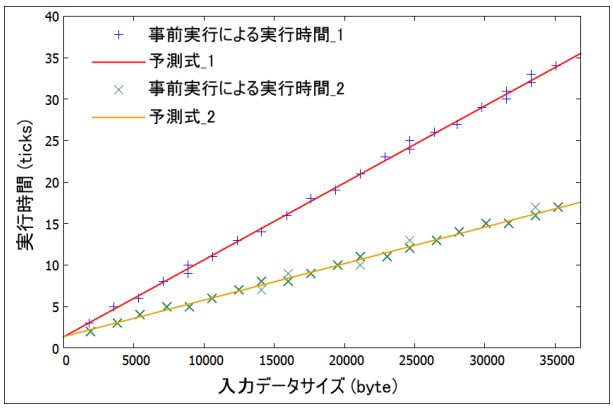

図 5.25: Tel./GSM-encode

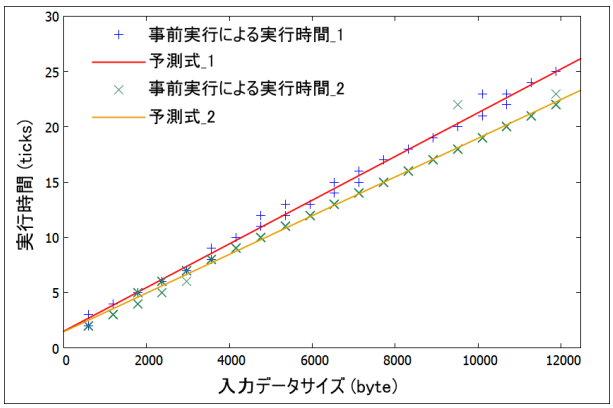

図 5.26: Tel./GSM-decode

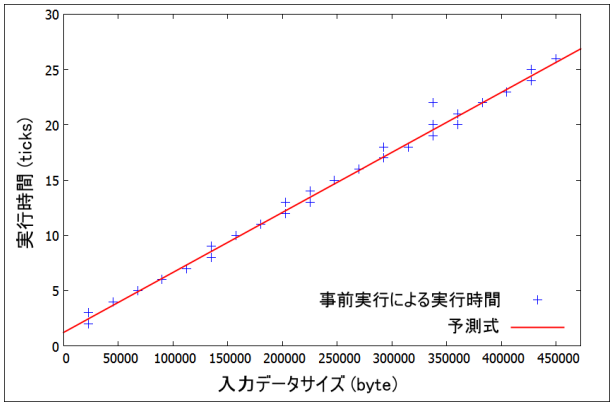

図 5.27: Tel./CRC32

第6章　評価

本章では，シミュレーションによって各スケジューリング方式の性能を比較した結果について述べる．性能として，非周期タスクの平均応答時間と，周期タスクの絶対ジッタおよび相対ジッタを対象とする．

本研究において，実際のシステムにおける提案手法の有用性を示すために，実際のアプリケーションの実行時間を反映した上での評価を行う．実際のアプリケーションとして，組込み向けベンチマーク集である MiBench を使用する．MiBench は無償で提供されており，かつ現在でも様々な文献において使用されている [5][6][7][8]．対象タスクの実行時間として，前章で述べた 27 アプリケーションを用いて実際に計測した実行時間を使用した．各アプリケーションにおいて，対象タスクに用いる実行時間の計測に使用した入力と，前章で示した（予測式生成のための）事前実行における計測に使用した入力は全て異なる．

実行時間の計測において，使用した環境は Intel Xeon E3-1280 3.6GHz，Linux 2.6.32 である．各アプリケーションごとに，主要な処理 [1] を実行するために費やした CPU 時間を実行時間とした．実行時間の計測には clock_gettime() 関数を使用した．これは，Linux の高精度クロックによってナノ秒単位の時刻を取得する関数である．

6.1　評価1：非周期タスクの平均応答時間

MiBench の各アプリケーションを非周期タスクとして，その非周期タスクと周期タスクが混在するタスクセットにおけるスケジューリングのシミュレーションを行う．このシミュレーションによって，各スケジューリング方式による非周期タスクの応答時間を比較する．

6.1.1　評価方法

5 つの手法，TBS，ATBS，ATBSM，ATBSM+dwcet，タスクの実行時間が既知である理想の TBS を比較する．タスクの実行時間が既知である理想の TBS を，ORACLE とする．TBS アルゴリズムを用いた手法による非周期タスクの応答時間は，ORACLE の場合に最短となる．従って，上記の ORACLE 以外の 4 つの手法による応答時間が ORACLE による応答時間よりも短くなることはない．

[1] 初期化や結果の出力等を除いた処理を，主要な処理とする．

対象タスクセットとして，CPU 使用率が $U_p = 60\%, 65\%, \cdots, 95\%$ となる周期タスクセットと，観測期間内において WCET による CPU 使用率が平均 5%となる非周期タスクセットを組み合わせたものを使用する．観測期間は，非周期タスクの 100 回のインスタンスが終了するまでとする．ただし Office/ghostscript アプリケーションを使用する場合の観測期間は，シミュレーションの時間の関係上，非周期タスクの 10 回のインスタンスが終了するまでとする．周期タスクセットは各 U_p ごとに 30 個，非周期タスクセットは 10 個用意する．従って，各 U_p ごとに，30 個の周期タスクセットと 10 個の非周期タスクセットの全ての組み合わせ (300 個のタスクセット) をシミュレーションし，その平均値を結果とする．非周期タスクの実行を管理するサーバーのバンド幅 U_s は，$U_s = 1 - U_p$ とする．

周期タスクについて，周期は平均 100 ティックの指数分布に基づく乱数によって決定する．WCET は，平均 10 ティックの指数分布に基づく乱数によって決定する．また，WCET と実行時間は等しいする．

非周期タスクについて，実行時間は MiBench の各アプリケーションにおいて実際に計測した実行時間を使用する．このとき，予測要因の値は指数分布に基づく乱数によって決定する．指数分布の平均値は，(予測式生成のための) 事前実行にて指定した予測要因の最大値の 1/5 の値とする．WCET は，観測期間内に実行する全ての非周期タスク実行において，最も長い実行時間の 1.5 倍の値とする．各タスクの到着間隔は，ポアソン分布に基づく乱数によって決定する．このとき，観測期間内において WCET による CPU 使用率を平均 5%とするために，平均で $20 \times WCET$ ティック内に 1 回到着するとした．

ATBS は平均実行時間を PET として使用する．ATBSM と ATBSM+dwcet においては，前章で示した各アプリケーションの予測式を使用して PET を計算する．また，ATBSM+dwcet において，5 段階の dwcet を使用する．

6.1.2 結果

非周期タスクとして Automotive/qsort アプリケーションを使用したときの，TBS における平均応答時間を 1 として正規化した平均応答時間を図 6.1 に示す．横軸が周期タスクの使用率，縦軸が正規化した平均応答時間を示す．全ての U_p の場合において，ATBSM による平均応答時間は既存の手法による平均応答時間よりも短くなっていることが分かる．TBS と比較した際，ATBSM によって $U_p = 75\%$ のときに最大 62.3%，ATBS と比較した際，ATBSM によって $U_p = 75\%$ のときに最大 40.9%短縮された．また，ATBSM+dwcet による平均応答時間は ATBSM による平均応答時間よりも短くなっていることが分かる．ATBSM と比較した際，ATBSM+dwcet によって $U_p = 75\%$ のときに最大 3.6%短縮された．

全 27 アプリケーションにおける平均値を図 6.2 に示す．全ての U_p の場合において，ATBSM による平均応答時間は既存の手法よりも短くなっており，かつ ATBSM+dwcet による平均応答時間は ATBSM よりも短くなっていることが分かる．TBS と比較した際，ATBSM によって $U_p = 75\%$ のときに最大 52.5%，ATBSM+dwcet によって $U_p = 75\%$ のときに最大 53.2%短縮された．また，ATBS と比較した際，ATBSM によって $U_p = 75\%$

のときに最大 30.2%，ATBSM+dwcet によって $U_p = 75\%$ のときに最大 31.3%短縮された．

Aut./qsort アプリケーション以外の各アプリケーションにおける結果を，図 6.3〜図 6.28 に示す．これらの図において，平均応答時間を ART(Average Response Time) と表記している．

ATBS と ATBSM における PET について考察する．まず，非周期インスタンスの実際の実行時間と 2 つの手法による PET の誤差を比較する．Aut./qsort アプリケーションを使用したとき，シミュレーションで使用した全ての非周期インスタンス（1 セットにつき 100 回× 10 セット=計 1000 回の非周期インスタンス）において，ATBS による PET と実際の実行時間の誤差の平均値は 4.47 ティック，ATBSM による PET と実行時間の誤差の平均値は 0.33 ティックとなった．これは，提案手法の予測式を使用した実行時間予測によって，ATBS の場合よりも誤差を 92.6%小さくすることができたことを示す．全 27 アプリケーションの平均においては，提案手法によって ATBS の場合よりも誤差を 77.2%小さくすることができた．この結果は，提案手法によって ATBS よりも高精度に実行時間予測を行えていることを示す．次に，PET 内に終了する非周期インスタンスの割合を比較する．Aut./qsort アプリケーションを使用したとき，ATBS において 70.0%，ATBSM において 96.4%の非周期インスタンスが PET 内に終了した．全 27 アプリケーションの平均においては，ATBS において 66.1%，ATBSM において 92.0%の非周期インスタンスが PET 内に終了した．この結果は，提案手法によって ATBS よりも過小見積の数を減らすことができていることを示す．

次に，dwcet の効果について考察する．ATBSM と ATBSM+dwcet における，過小見積時のデッドラインの長さを比較する．すなわち，過小見積時のデッドライン計算において，WCET を使用する場合と dwcet を使用する場合を比較する．Aut./qsort アプリケーションにおいて，ATBSM+dwcet による過小見積時のデッドラインの長さが，ATBSM の場合よりも平均 75.0%短縮された．全 27 アプリケーションの平均においては，平均 42.2%短縮された．以上の結果は，ATBSM+dwcet によって過小見積時の影響を小さくできていることを示す．

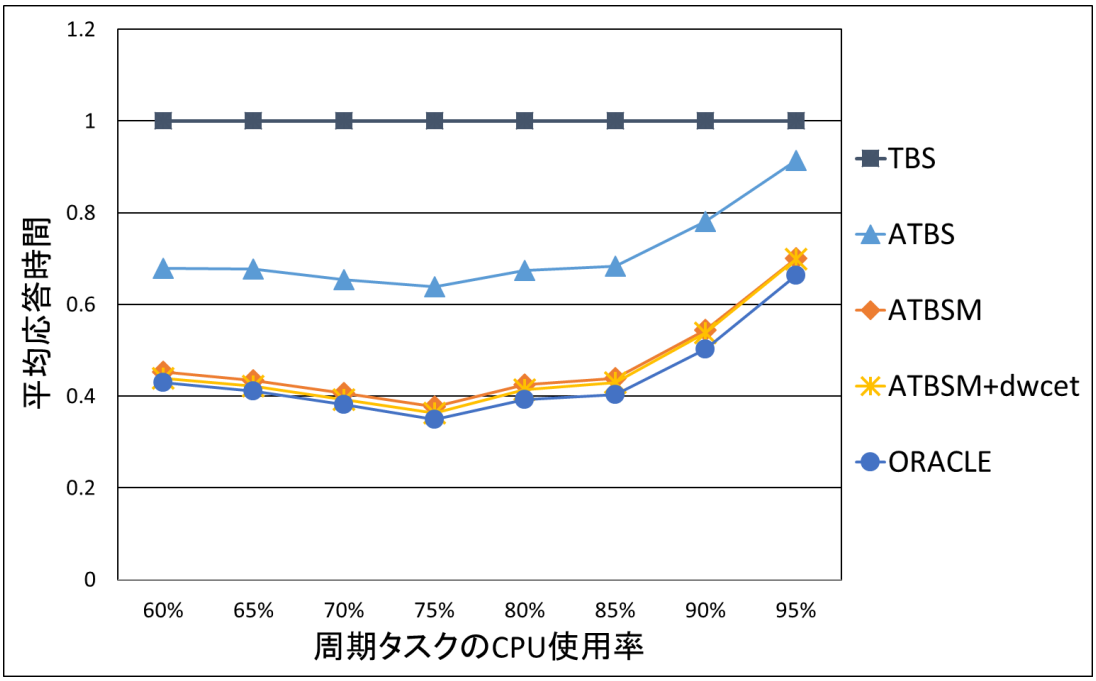

図 6.1: Aut./qsort における平均応答時間

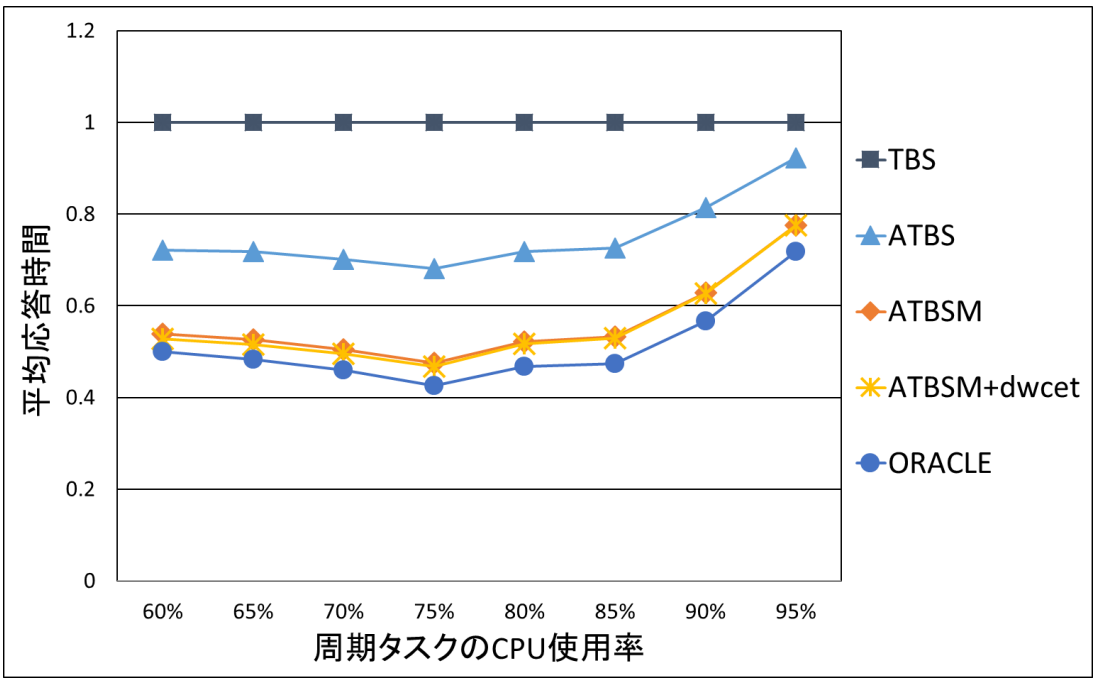

図 6.2: 27 アプリケーションの平均値

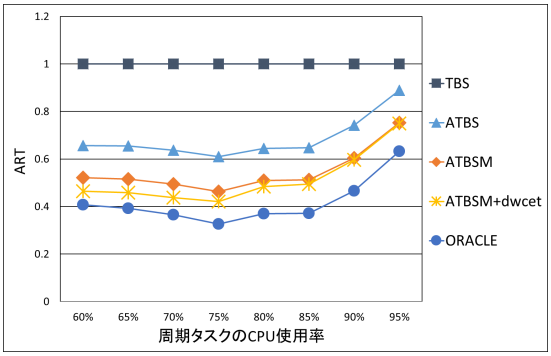

図 6.3: Aut./susan-ed. における ART

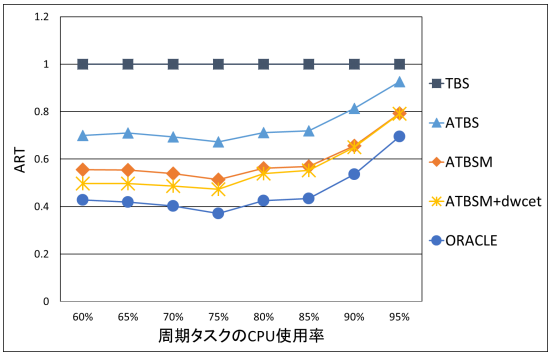

図 6.4: Aut./susan-co. における ART

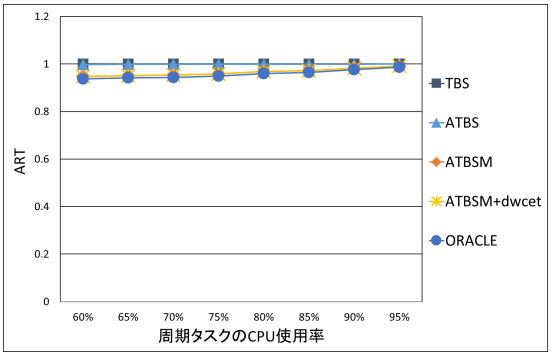

図 6.5: Aut./susan-sm. における ART

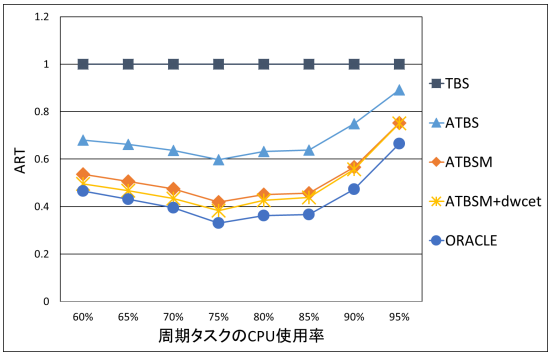

図 6.6: Con./jpeg-enc. における ART

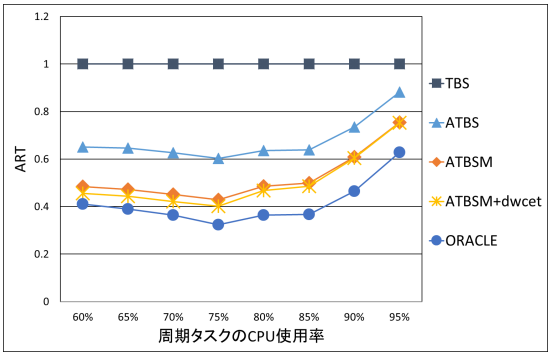

図 6.7: Con./jpeg-dec. における ART

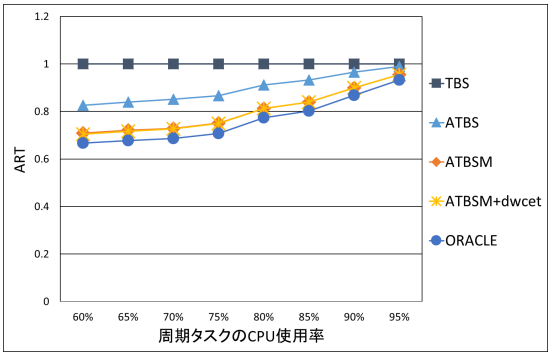

図 6.8: Con./lame における ART

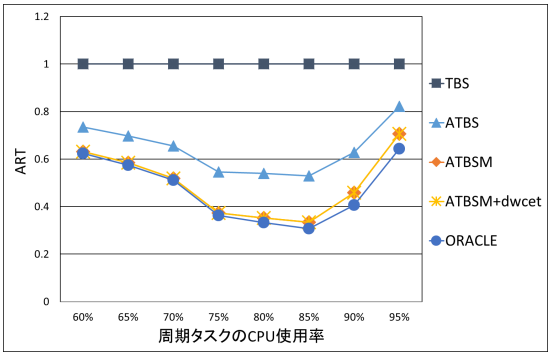

図 6.9: Con./mad における ART

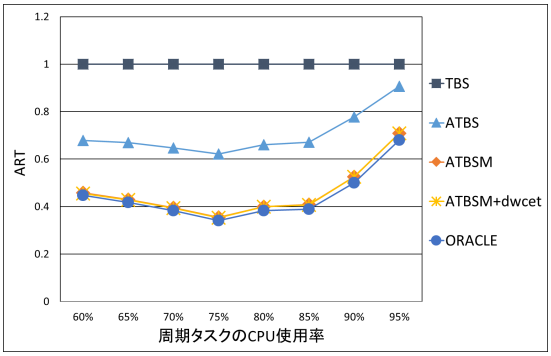

図 6.10: Con./tiff2bw における ART

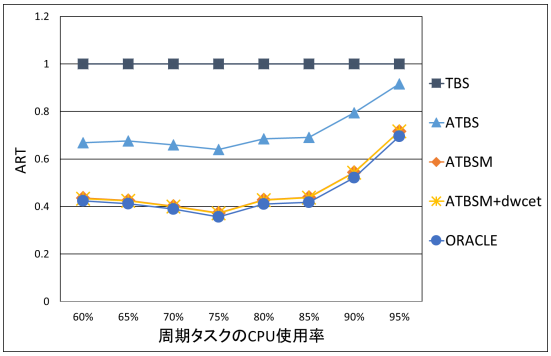

図 6.11: Con./tiff2rgba における ART

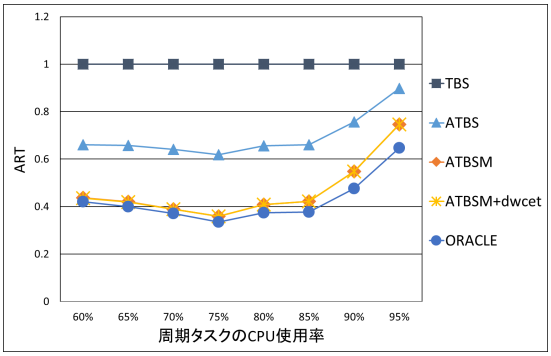

図 6.12: Con./tiff2dither における ART

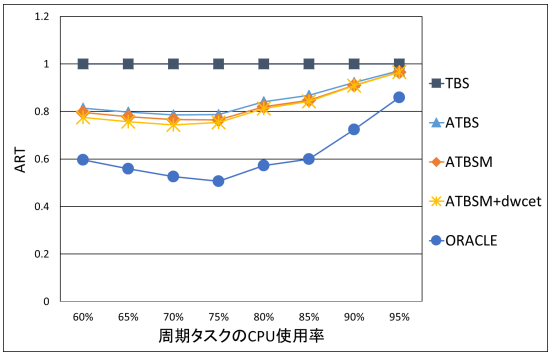

図 6.13: Con./tiff2median における ART

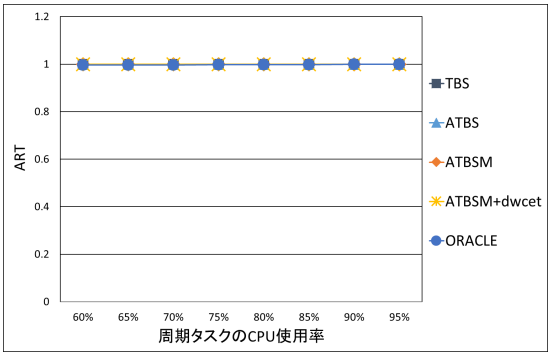

図 6.14: Off./ghostscript における ART

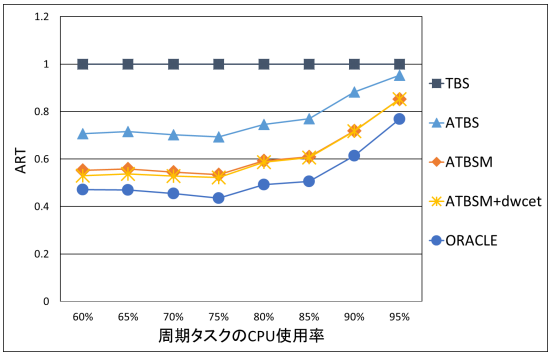

図 6.15: Off./rsynth における ART

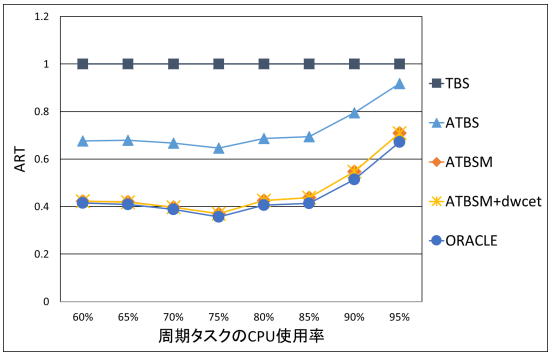

図 6.16: Net./patricia における ART

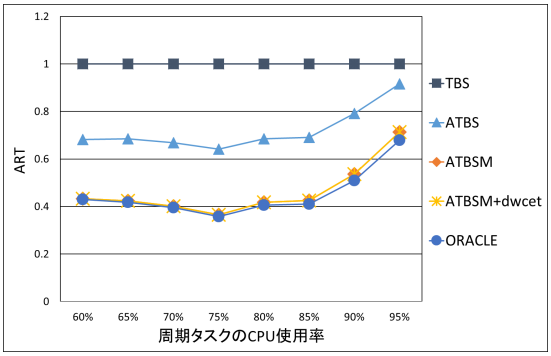

図 6.17: Sec./blowfish-enc. における ART

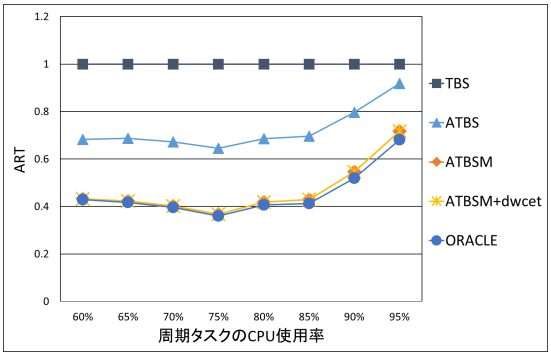

図 6.18: Sec./blowfish-dec. における ART

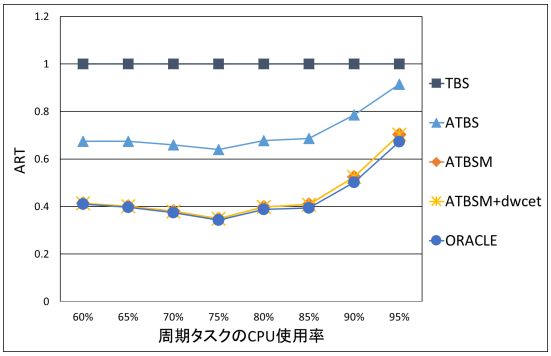

図 6.19: Sec./rijndael-enc. における ART

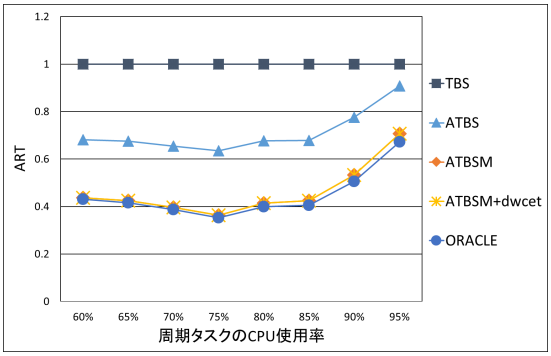

図 6.20: Sec./rijndael-dec. における ART

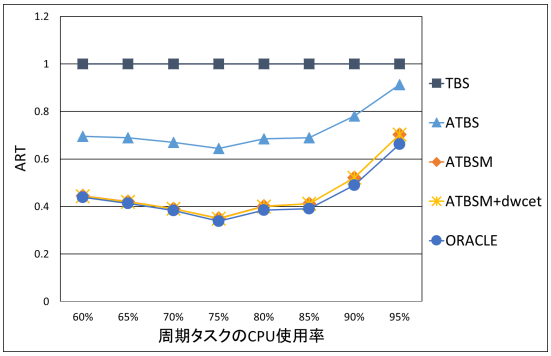

図 6.21: Sec./sha における ART

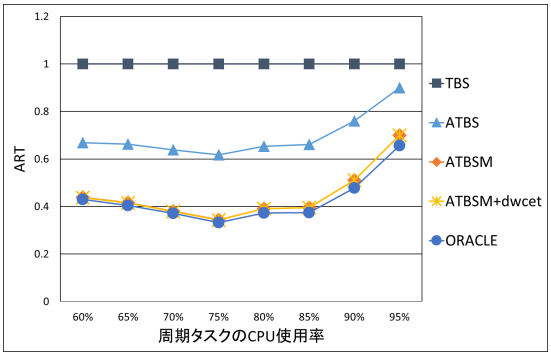

図 6.22: Tel./CRC32 における ART

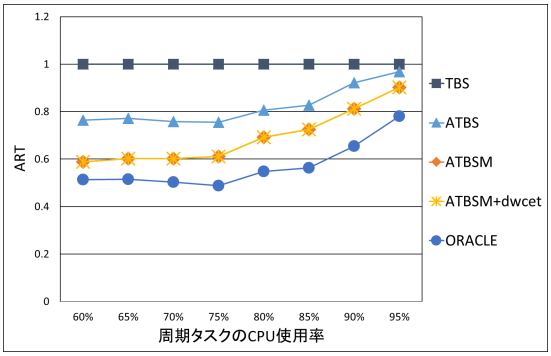

図 6.23: Tel./FFT における ART

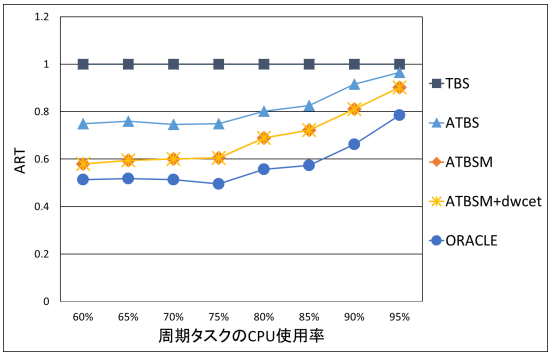

図 6.24: Tel./IFFT における ART

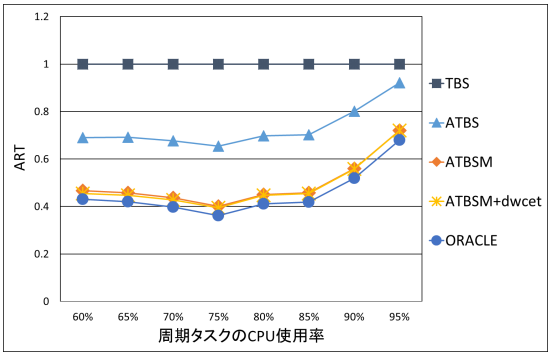

図 6.25: Tel./adpcm-enc. における ART

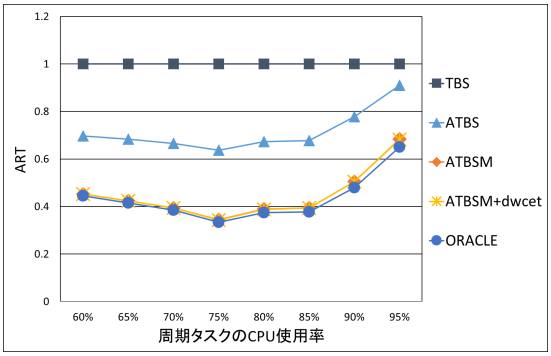

図 6.26: Tel./adpcm-dec. における ART

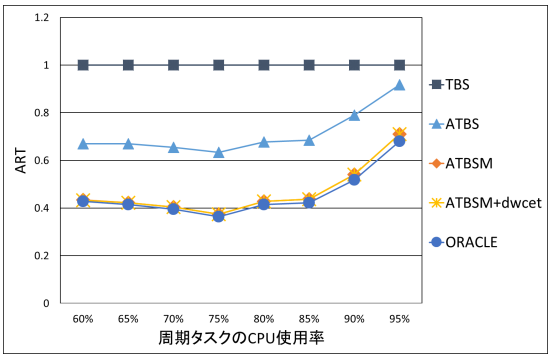

図 6.27: Tel./gsm-enc. における ART

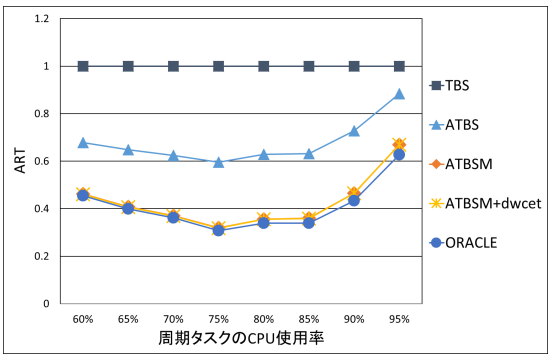

図 6.28: Tel./gsm-dec. における ART

6.2　評価 2：周期タスクのジッタ

MiBench の各アプリケーションを周期タスクとして，MiBench による周期タスクと他の周期タスクが混在するタスクセットにおけるスケジューリングのシミュレーションを行う．このシミュレーションによって，各スケジューリング方式を使用したときの MiBench による周期タスクの絶対ジッタおよび相対ジッタを比較する．

6.2.1　ジッタ短縮用の予測式について

評価 2 において使用する，ジッタを短縮するための予測式について説明する．ジッタの短縮を考慮する場合，応答時間の場合とは異なり，必ずしも対象タスクの全てのインスタンスにおいて高精度に実行時間の予測を行えれば良いわけではない．どれだけ応答時間が短縮されても対象タスクにおけるインスタンス間の応答時間の差が広がるならば，ジッタは長くなる．ジッタを短縮するためには，実行時間が長いインスタンスに対してより高精度に実行時間予測を行い，応答時間を短くすることが有効である．実行時間が長いインスタンスは実行時間が短いものよりも応答時間が長くなりやすいため，実行時間が長いインスタンスの応答時間を短縮することで対象タスクにおけるインスタンス間の応答時間の差を狭める．そのため，実行時間が長いインスタンス，すなわち予測要因の値が大きいインスタンスに対してより高精度に実行時間予測を行える式を生成する．これは 3 章で述べたように，予測式の生成の際に事前実行にて得られた 200 個の計測結果の全ては使用せず，予測要因の値における上位の計測結果のみを使用することによって実現できる．ここでは，予測要因の値における上位 100 個の計測結果のみを使用して，ジッタを短縮するための予測式を生成した．ATBSM と ATBSM+dwcet において，ジッタを短縮するための予測式を使用した手法を，それぞれ ATBSM(2) と ATBSM(2)+dwcet とする．この 2 つの手法によって，予測要因の値が大きいインスタンスに対してより高精度に実行時間を予測し応答時間を短くすることで，ATBSM と ATBSM+dwcet よりも短いジッタとなることを狙う．

6.2.2　評価方法

評価 2 において，7 つの手法，TBS，ATBS，ATBSM，ATBSM+dwcet，ATBSM(2)，ATBSM(2)+dwcet，ORACLE を比較する．

対象タスクセットとして，CPU 使用率が $U_p = 60\%, 65\%, \cdots, 95\%$ となる周期タスクセットと，観測期間内において WCET による CPU 使用率が平均 5%となる MiBench による周期タスクセットを組み合わせたものを使用する．観測期間は，MiBench による周期タスクの 100 回のインスタンスが終了するまでとする．ただし Office/ghostscript アプリケーションを使用する場合の観測期間は，シミュレーションの時間の関係上，MiBench による周期タスクの 10 回のインスタンスが終了するまでとする．周期タスクセットは各 U_p ご

とに 30 個，MiBench による周期タスクセットは 10 個用意する．従って，各 U_p ごとに，30 個の周期タスクセットと 10 個の MiBench による周期タスクセットの全ての組み合わせ (300 個のタスクセット) をシミュレーションし，その平均値を結果とする．MiBench による周期タスクの実行を管理するサーバーのバンド幅 U_s は，$U_s = 1 - U_p$ とする．

周期タスクは，評価 1 で使用した周期タスクと同一のものを使用する．MiBench による周期タスクとして，評価 1 で使用した非周期タスクが周期的に到着するように変更したものを使用する．到着間隔は，観測期間内において WCET による CPU 使用率を平均 5%とするために，$20 \times WCET$ ティックとする．

6.2.3 結果（絶対ジッタ）

MiBench による周期タスクとして Aut./qsort アプリケーションを使用したときの，TBS における絶対ジッタを 1 として正規化した絶対ジッタを図 6.29 に示す．横軸が周期タスクの使用率，縦軸が正規化した絶対ジッタを示す．全ての U_p の場合において，平均応答時間の場合と異なり，ATBS よりも TBS の方が良い結果となっている．ATBS のデッドライン計算において，実行時間が長いインスタンスに対しては過小見積となり WCET を使用する場合が多いが，実行時間が短いインスタンスに対しては PET を使用する場合が多い．そのため，TBS と比較した際，ATBS によって最小応答時間は短くなるが，最長応答時間は短くならない場合が出てくる．従ってジッタにおいて，ATBS よりも TBS の方が良い結果となる傾向がある．全ての U_p の場合において，ATBSM と ATBSM+dwcet による絶対ジッタは既存の手法による絶対ジッタよりも短くなっていることが分かる．TBS と比較した際，ATBSM によって $U_p = 75\%$ のときに最大 16.6%，ATBSM+dwcet によって $U_p = 75\%$ のときに最大 16.8%短縮された．ATBS と比較した際，ATBSM によって $U_p = 75\%$ のときに最大 16.9%，ATBSM+dwcet によって $U_p = 75\%$ のときに最大 17.1%短縮された．また，ATBSM(2) と ATBSM(2)+dwcet による絶対ジッタは，それぞれ ATBSM と ATBSM+dwcet による絶対ジッタよりも短くなっていることが分かる．ATBSM と比較した際，ATBSM(2) によって $U_p = 85\%$ のときに最大 5.2%短縮された．ATBSM+dwcet と比較した際，ATBSM(2)+dwcet によって $U_p = 70\%$ のときに最大 7.8%短縮された．

全 27 アプリケーションにおける平均値を図 6.30 に示す．全ての U_p の場合において，ATBSM と ATBSM+dwcet による絶対ジッタは既存の手法よりも短くなっており，かつ ATBSM(2) による絶対ジッタは ATBSM よりも，ATBSM(2)+dwcet による絶対ジッタは ATBSM+dwcet よりも短くなっていることが分かる．TBS と比較した際，ATBSM によって $U_p = 70\%$ のときに最大 15.1%，ATBSM(2) によって $U_p = 70\%$ のときに最大 15.8%，ATBSM+dwcet によって $U_p = 60\%$ のときに最大 15.9%，ATBSM(2)+dwcet によって $U_p = 70\%$ のときに最大 17.0%短縮された．

Aut./qsort アプリケーション以外の各アプリケーションにおける結果を，図 6.31〜図 6.56 に示す．これらの図において，絶対ジッタを AJ（Absolute Jitter）と表記している．

ジッタ短縮用の予測式の効果について考察する．ATBSM と ATBSM(2) における，予測

要因が大きいインスタンスにおける PET を比較する．予測要因が大きいインスタンスとして，シミュレーションで使用した全ての MiBench による周期インスタンス（1 セットにつき 100 回× 10 セット=計 1000 回の周期インスタンス）の内，予測要因の値における上位 10%のものを対象とし，それらのインスタンスにおいて比較を行う．まず，Aut./qsort アプリケーションを使用した場合について述べる．ATBSM による PET と実際の実行時間の誤差の平均値は 0.59 ティック，ATBSM(2) による誤差の平均値は 0.36 ティックとなった．これは，予測要因が大きいインスタンスにおいて，ジッタ短縮用の予測式によって ATBSM の場合よりも誤差を 39.0%小さくすることができたことを示す．また，PET 内に終了するインスタンスの割合は，ATBSM において 97.0%，ATBSM(2) において 99.0%となった．これは，予測要因が大きいインスタンスにおいて，ジッタ短縮用の予測式によって ATBSM の場合よりも過小見積の数を減らすことができていることを示す．次に，全 27 アプリケーションの平均の場合について述べる．ジッタ短縮用の予測式による実際の実行時間と PET の誤差の減少率は，−9.9%，すなわち ATBSM の場合よりも誤差が 9.9%大きくなった．これは Con./tiff2dither アプリケーションにおける結果に大きく影響されているためである．Con./tiff2dither アプリケーションにおいてジッタ短縮用の予測式による誤差の減少率は，−310.5%となった．そのため，Con./tiff2dither アプリケーションを除いた 26 アプリケーションの平均においては，誤差は 1.7%小さくなった．また，PET 内に終了するインスタンスの割合は，ATBSM にて 87.8%，ATBSM(2) にて 90.0%となった．

以上の結果は予測要因が大きいインスタンスにおいて，ジッタ短縮用の予測式によって ATBSM よりも誤差を小さくし，かつ過小見積の数を減らす傾向があることを示す．

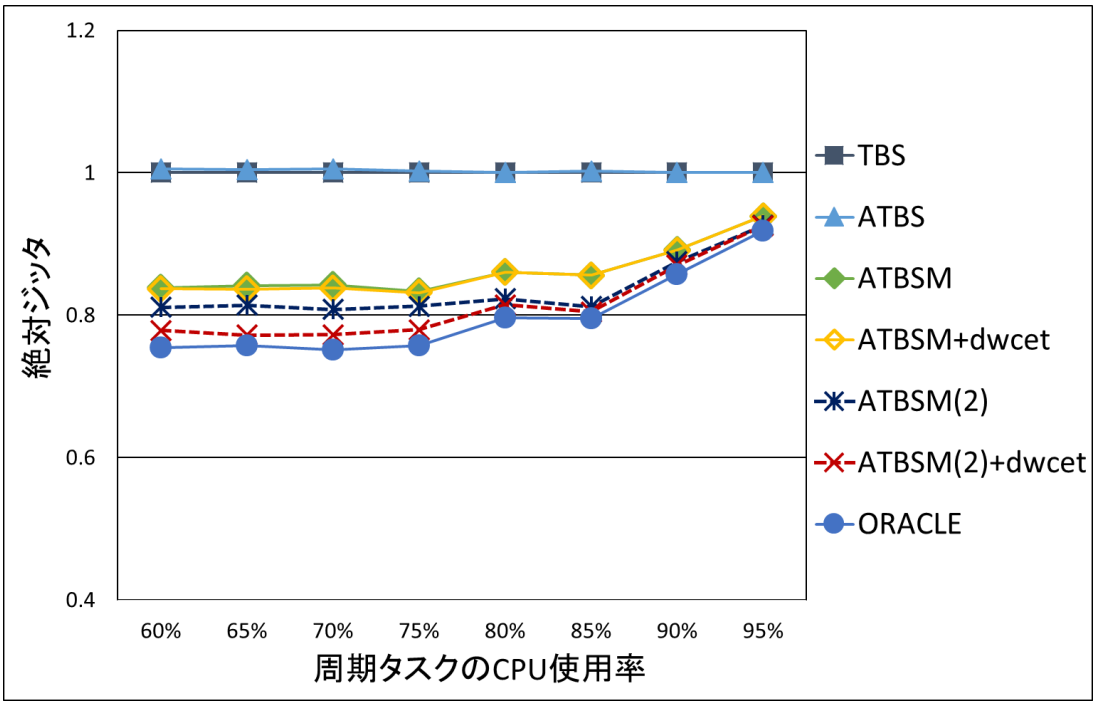

図 6.29: Aut./qsort における絶対ジッタ

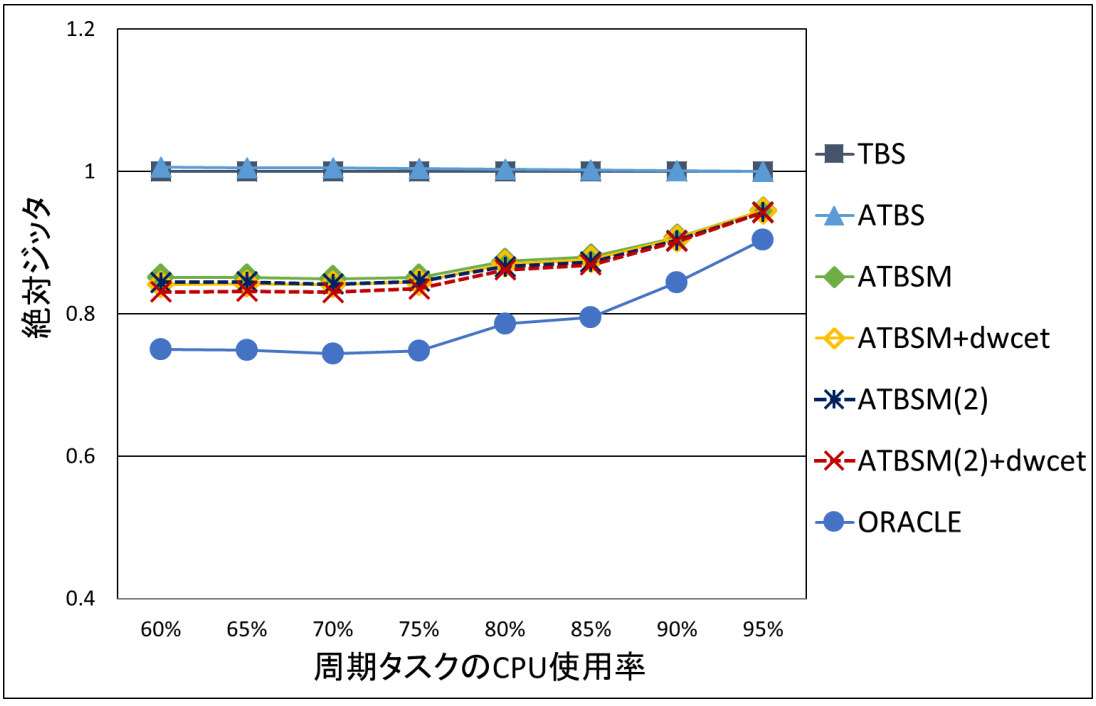

図 6.30: 27 アプリケーションにおける絶対ジッタの平均値

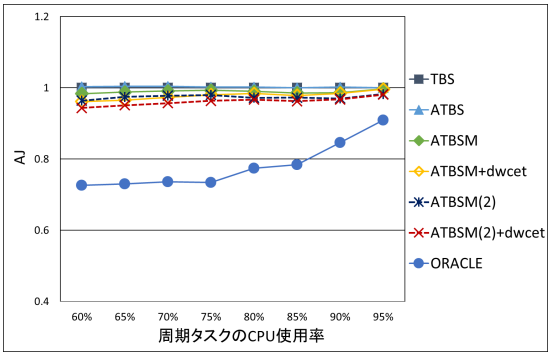

図 6.31: Aut./susan-ed. における AJ

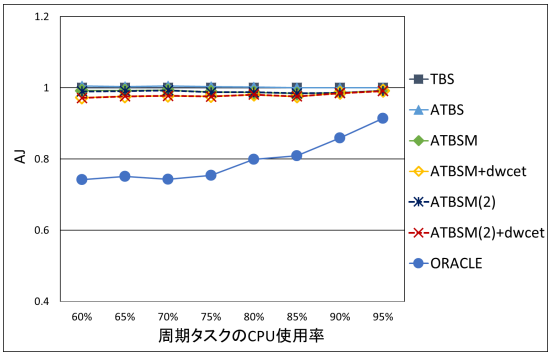

図 6.32: Aut./susan-co. における AJ

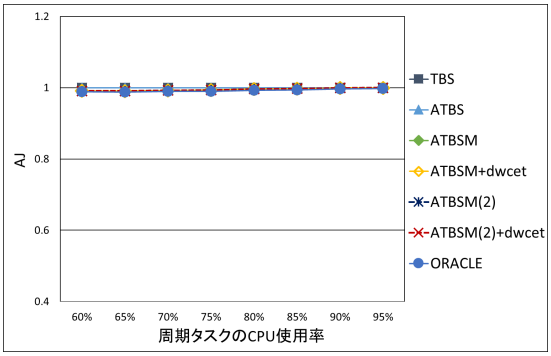

図 6.33: Aut./susan-sm. における AJ

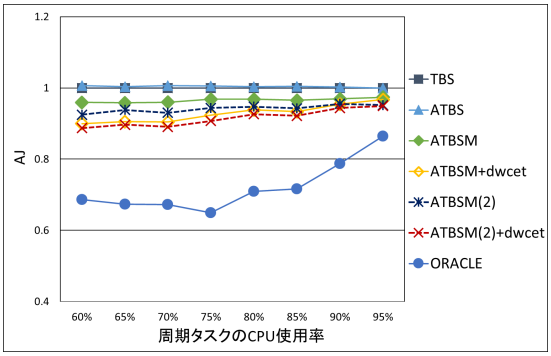

図 6.34: Con./jpeg-enc. における AJ

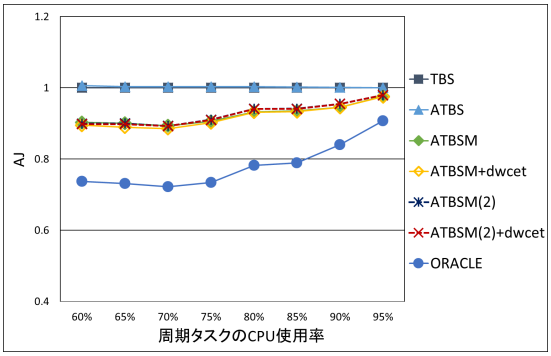

図 6.35: Con./jpeg-dec. における AJ

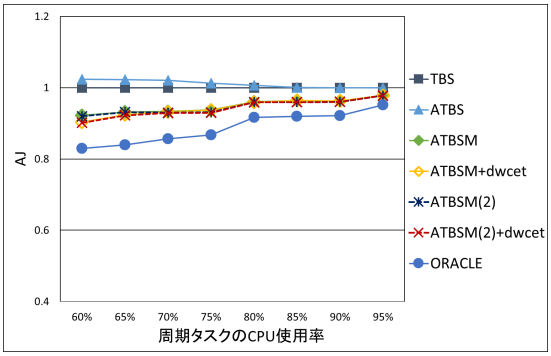

図 6.36: Con./lame における AJ

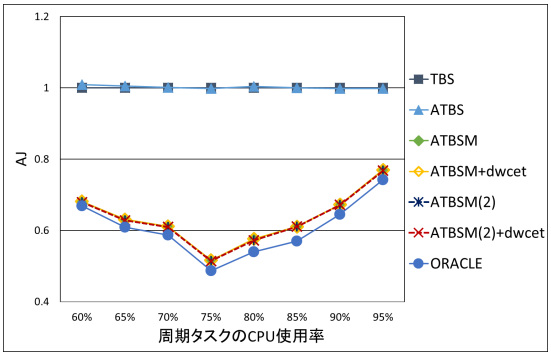

図 6.37: Con./mad における AJ

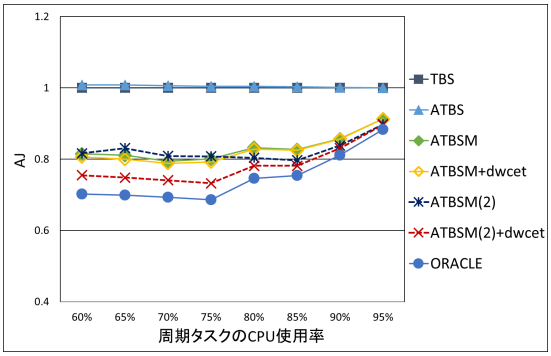

図 6.38: Con./tiff2bw における AJ

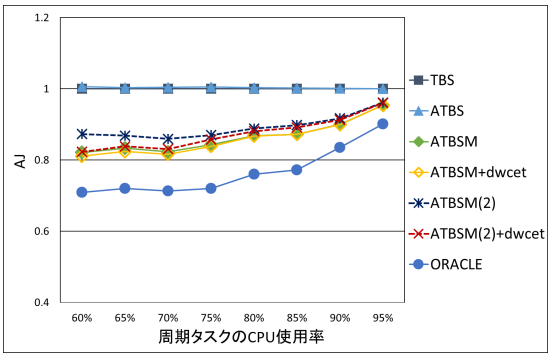

図 6.39: Con./tiff2rgba における AJ

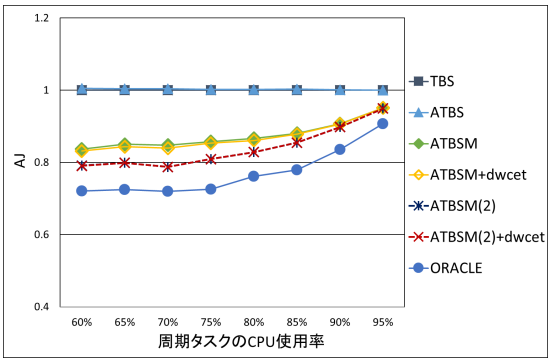

図 6.40: Con./tiff2dither における AJ

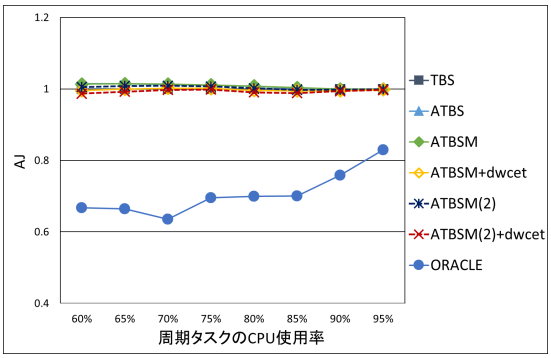

図 6.41: Con./tiff2median における AJ

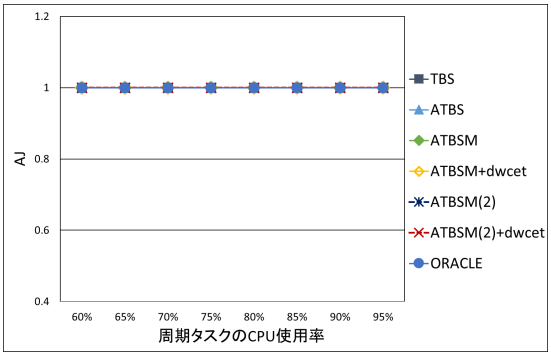

図 6.42: Off./ghostscript における AJ

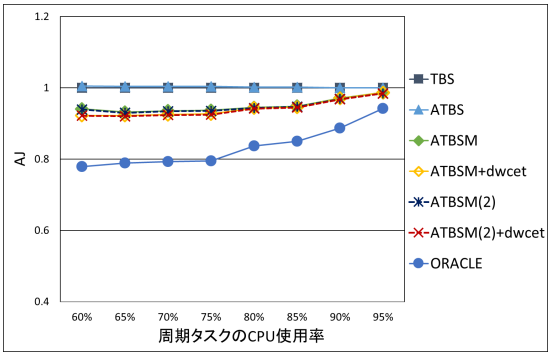

図 6.43: Off./rsynth における AJ

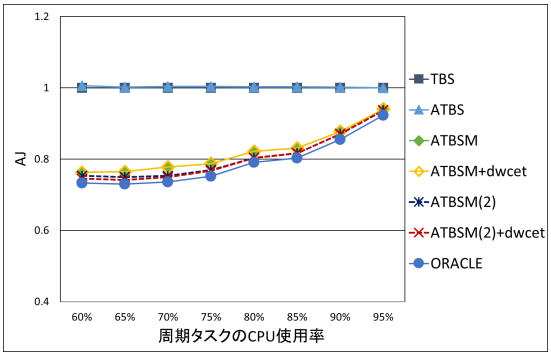

図 6.44: Net./patricia における AJ

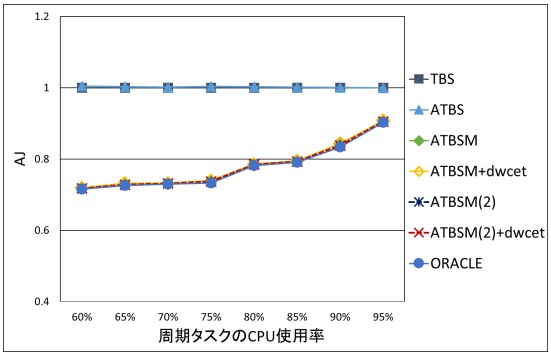

図 6.45: Sec./blowfish-enc. における AJ

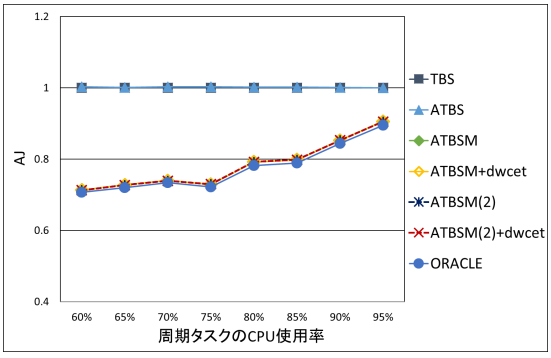

図 6.46: Sec./blowfish-dec. における AJ

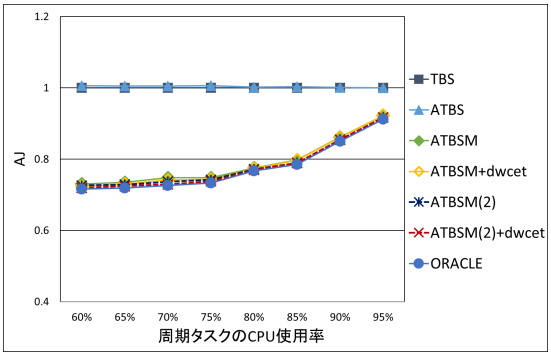

図 6.47: Sec./rijndael-enc. における AJ

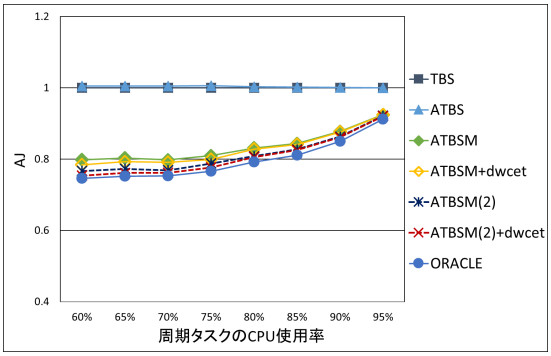

図 6.48: Sec./rijndael-dec. における AJ

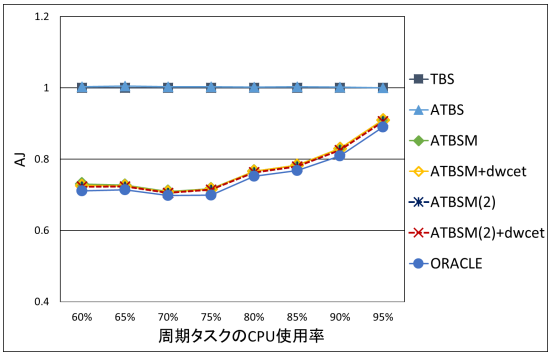

図 6.49: Sec./sha における AJ

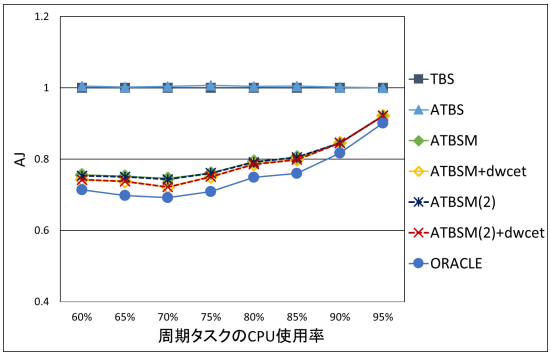

図 6.50: Tel./CRC32 における AJ

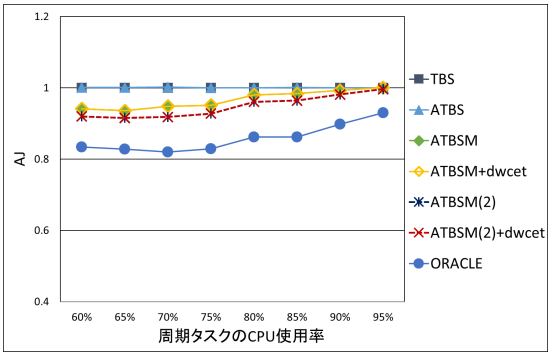

図 6.51: Tel./FFT における AJ

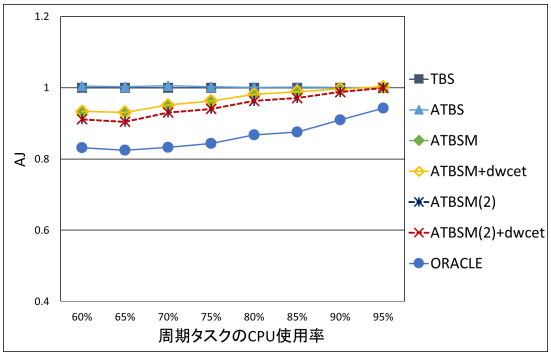

図 6.52: Tel./IFFT における AJ

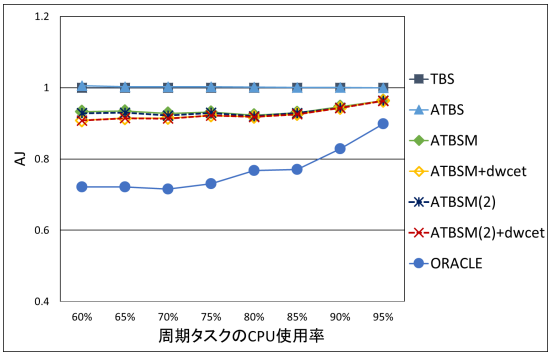

図 6.53: Tel./adpcm-enc. における AJ

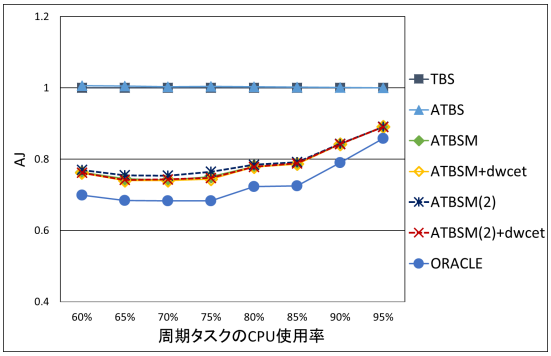

図 6.54: Tel./adpcm-dec. における AJ

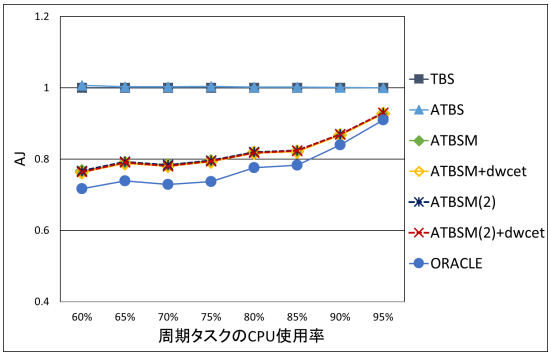

図 6.55: Tel./gsm-enc. における AJ

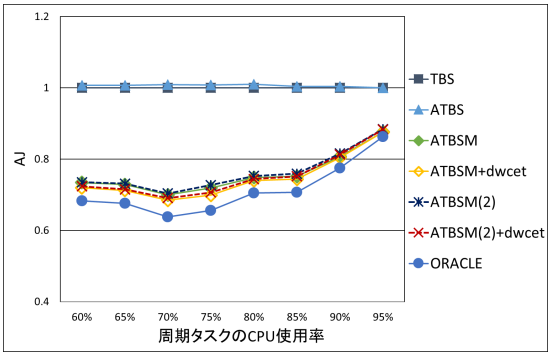

図 6.56: Tel./gsm-dec. における AJ

6.2.4 結果（相対ジッタ）

Aut./qsort アプリケーションを使用したときの，TBS における相対ジッタを 1 として正規化した相対ジッタを図 6.57 に示す．横軸が周期タスクの使用率，縦軸が正規化した相対ジッタを示す．全ての U_p の場合において，ATBSM と ATBSM+dwcet による相対ジッタは既存の手法による相対ジッタよりも短くなっていることが分かる．TBS と比較した際，ATBSM によって $U_p = 60\%$ のときに最大 12.6%，ATBSM+dwcet によって $U_p = 60\%$ のときに最大 12.7%短縮された．ATBS と比較した際，ATBSM によって $U_p = 60\%$ のときに最大 16.9%，ATBSM+dwcet によって $U_p = 65\%$ のときに最大 17.0%短縮された．また，ATBSM(2) と ATBSM+dwcet(2) による相対ジッタは，それぞれ ATBSM と ATBSM+dwcet による相対ジッタより短くなっていることが分かる．ATBSM と比較した際，ATBSM(2) によって $U_p = 85\%$ のときに最大 8.3%短縮された．ATBSM+dwcet と比較した際，ATBSM(2)+dwcet によって $U_p = 70\%$ のときに最大 8.5%短縮された．

全 27 アプリケーションにおける平均値を図 6.58 に示す．全ての U_p の場合において，ATBSM と ATBSM+dwcet による相対ジッタは既存の手法よりも短くなっており，かつ ATBSM(2) による相対ジッタは ATBSM よりも，ATBSM(2)+dwcet による相対ジッタは ATBSM+dwcet よりも短くなっていることが分かる．TBS と比較した際，ATBSM によって $U_p = 60\%$ のときに最大 11.6%，ATBSM(2) によって $U_p = 60\%$ のときに最大 12.9%，ATBSM+dwcet によって $U_p = 60\%$ のときに最大 12.7%，ATBSM(2)+dwcet によって $U_p = 60\%$ のときに最大 13.9%短縮された．

Aut./qsort アプリケーション以外の各アプリケーションにおける結果を，図 6.59〜図 6.84 に示す．これらの図において，相対ジッタを RJ（Relative Jitter）と表記している．

絶対ジッタの結果にて述べたように，ジッタ短縮用の予測式を使用することにより，予測要因の値が大きいインスタンスに対して実行時間予測の精度の向上および過小見積の数の減少を実現し，ジッタが短縮されている．

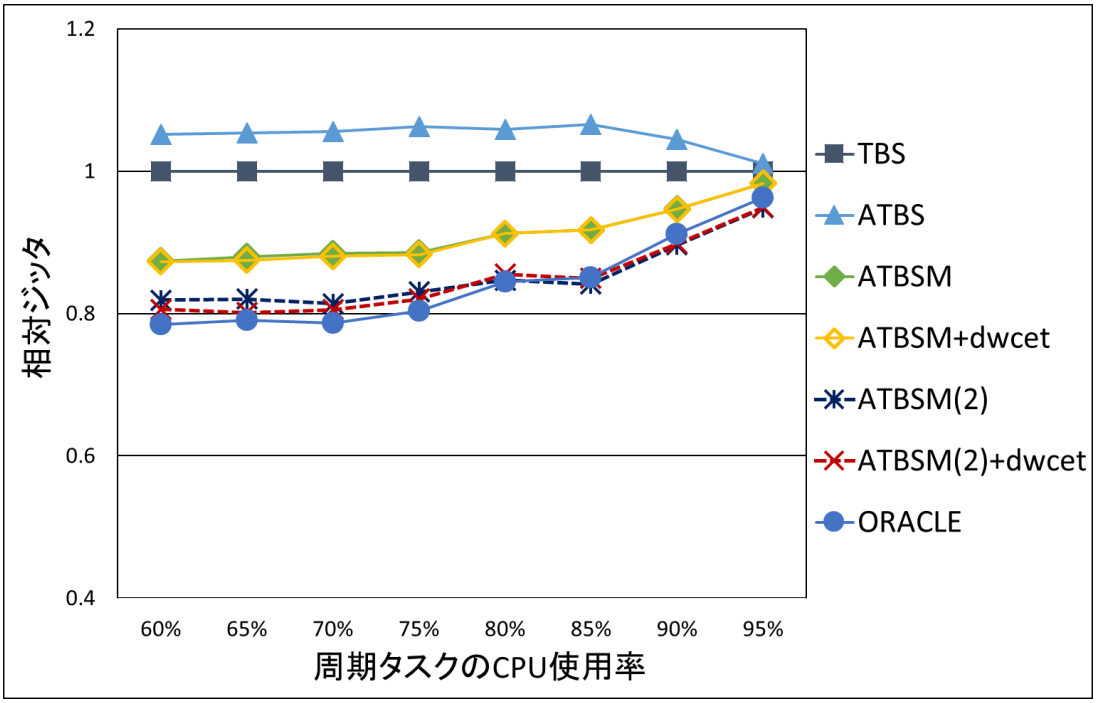

図 6.57: Aut./qsort における相対ジッタ

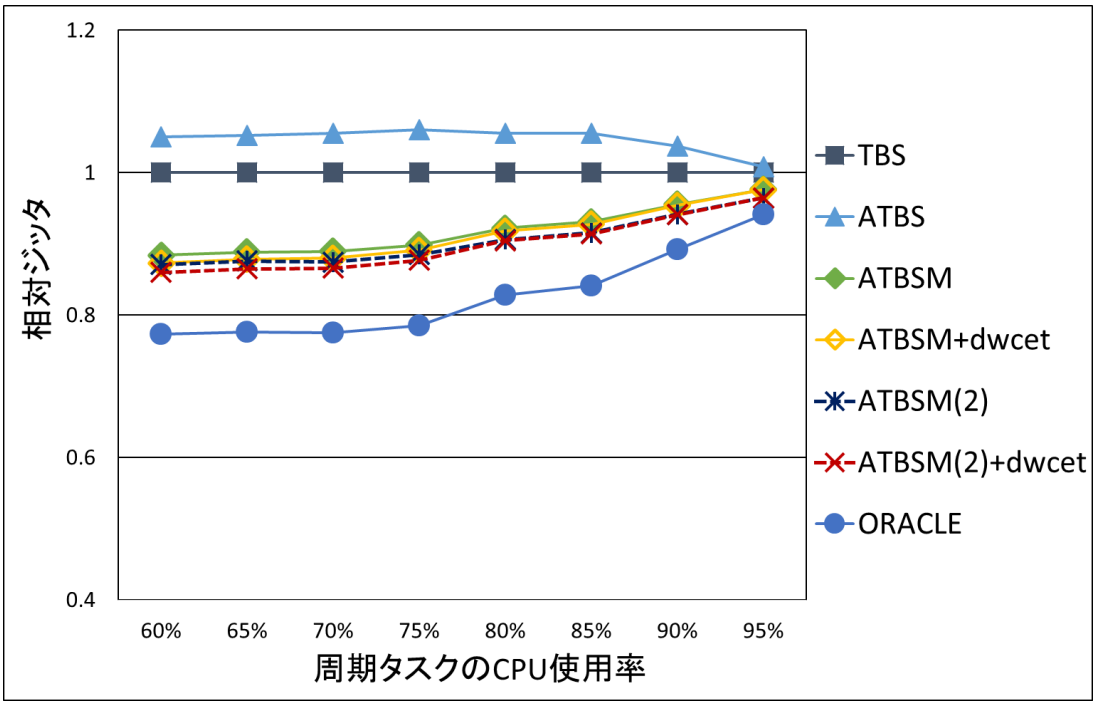

図 6.58: 27 アプリケーションにおける相対ジッタの平均値

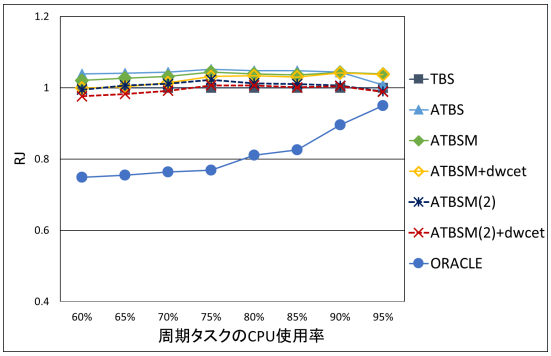

図 6.59: Aut./susan-ed. における RJ

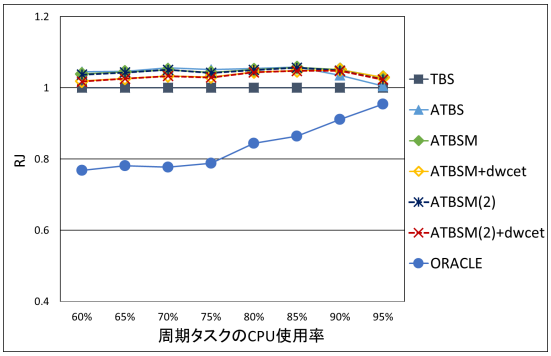

図 6.60: Aut./susan-co. における RJ

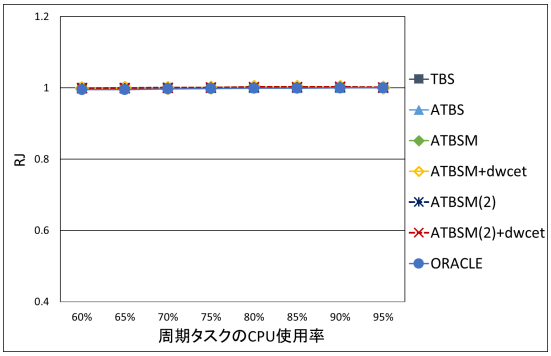

図 6.61: Aut./susan-sm. における RJ

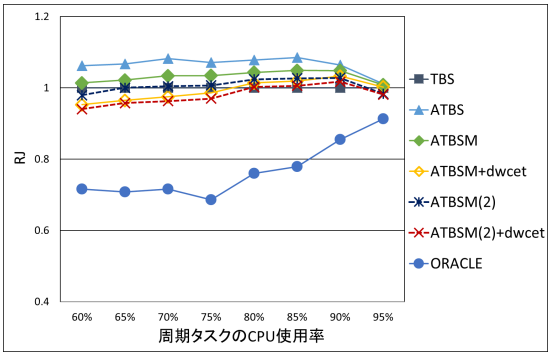

図 6.62: Con./jpeg-enc. における RJ

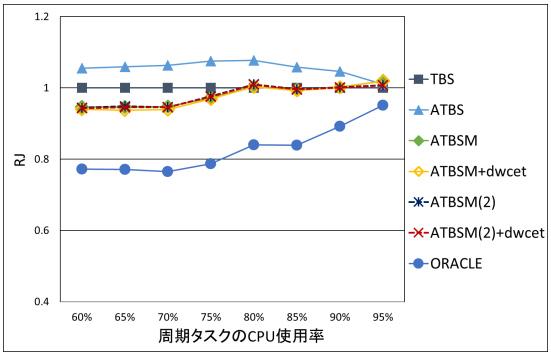

図 6.63: Con./jpeg-dec. における RJ

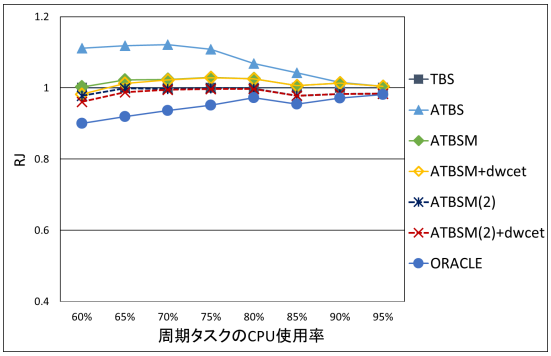

図 6.64: Con./lame における RJ

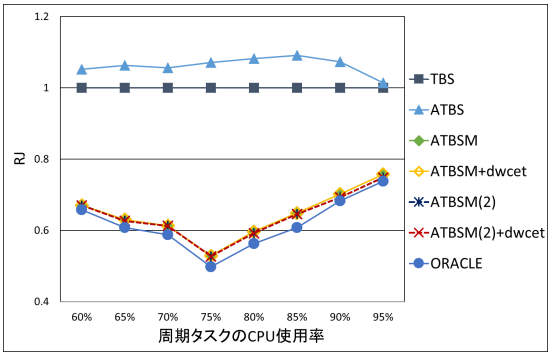

図 6.65: Con./mad における RJ

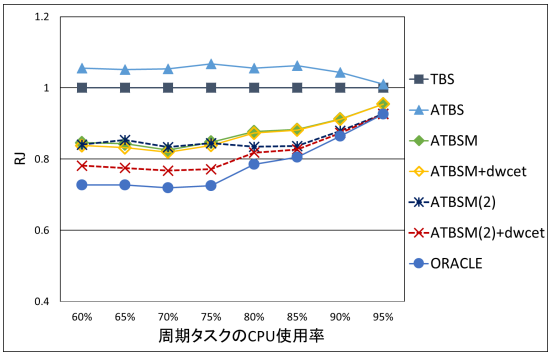

図 6.66: Con./tiff2bw における RJ

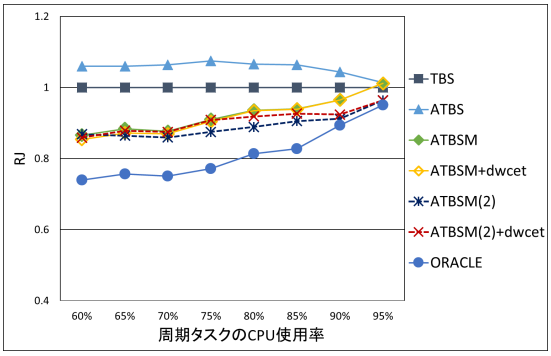

図 6.67: Con./tiff2rgba における RJ

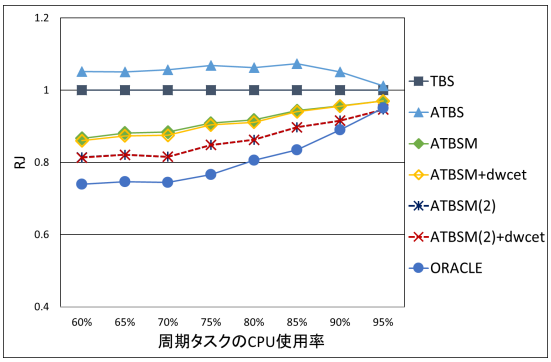

図 6.68: Con./tiff2dither における RJ

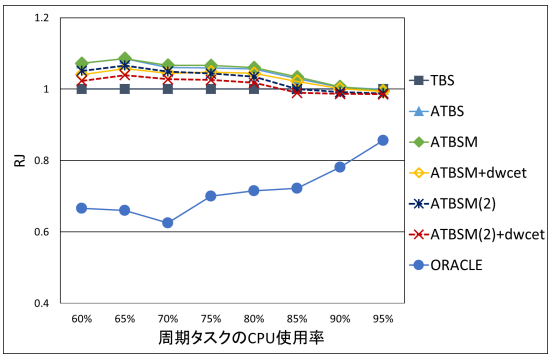

図 6.69: Con./tiff2median における RJ

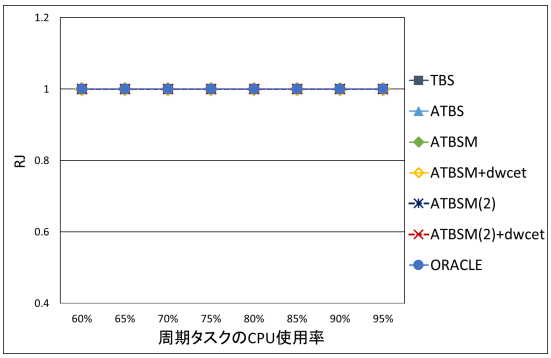

図 6.70: Off./ghostscript における RJ

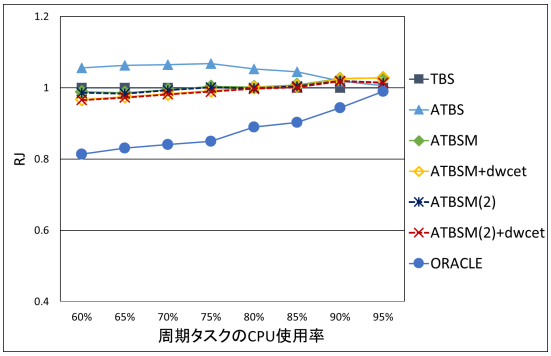

図 6.71: Off./rsynth における RJ

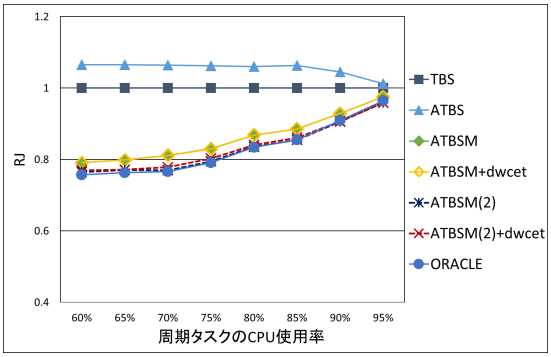

図 6.72: Net./patricia における RJ

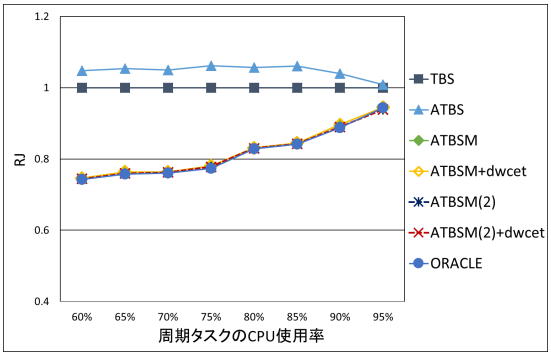

図 6.73: Sec./blowfish-enc. における RJ

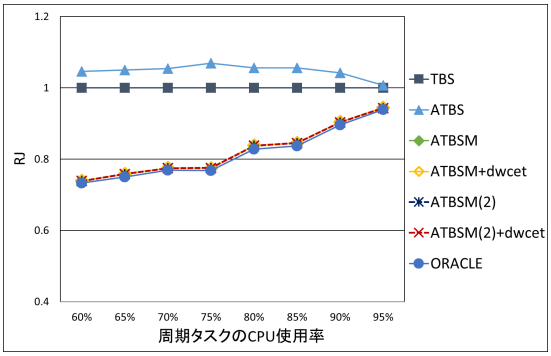

図 6.74: Sec./blowfish-dec. における RJ

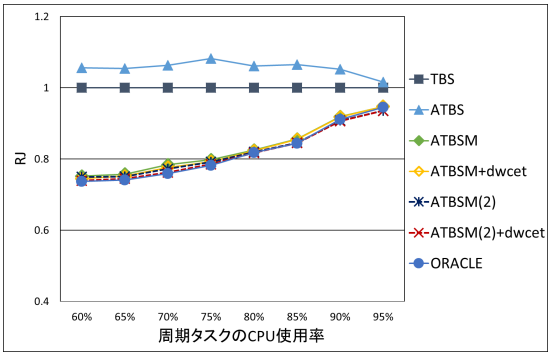

図 6.75: Sec./rijndael-enc. における RJ

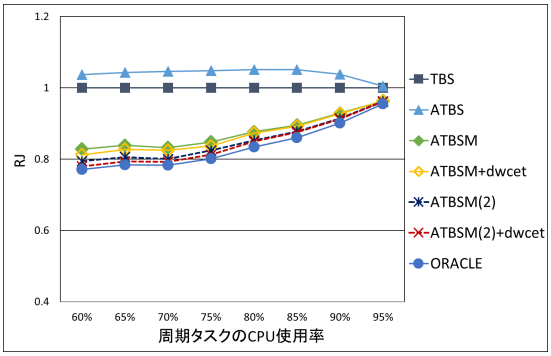

図 6.76: Sec./rijndael-dec. における RJ

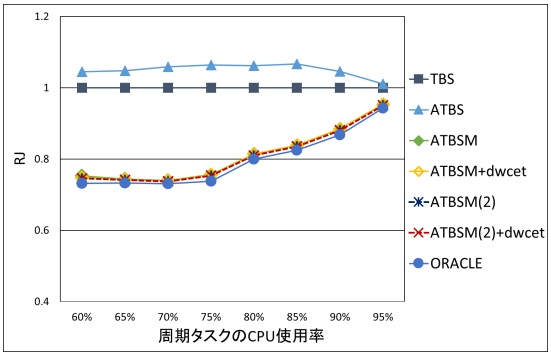

図 6.77: Sec./sha における RJ

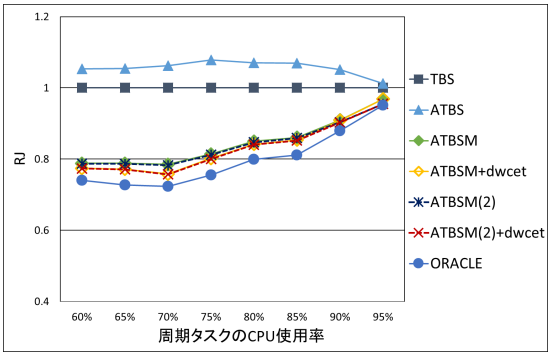

図 6.78: Tel./CRC32 における RJ

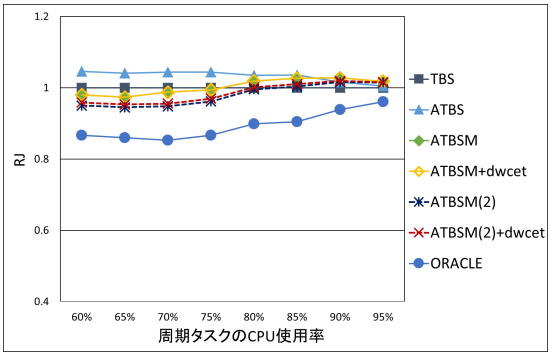

図 6.79: Tel./FFT における RJ

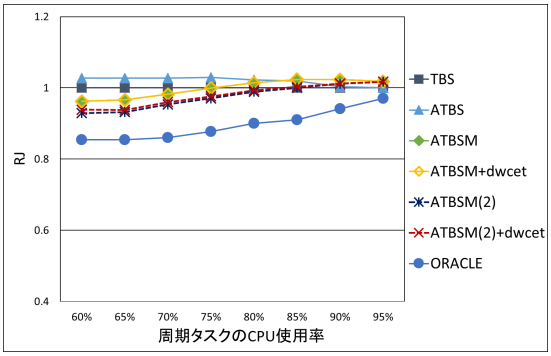

図 6.80: Tel./IFFT における RJ

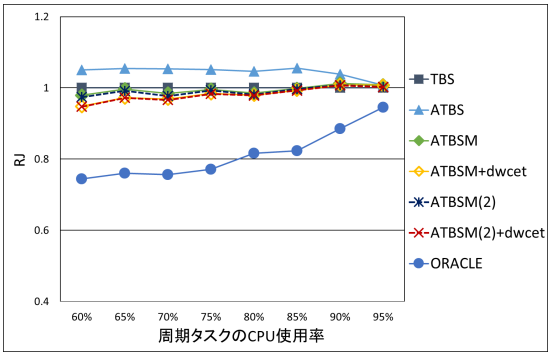

図 6.81: Tel./adpcm-enc. における RJ

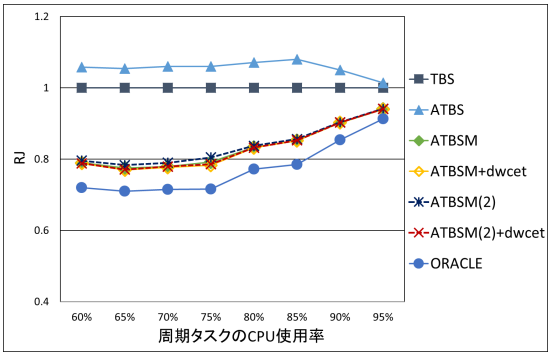

図 6.82: Tel./adpcm-dec. における RJ

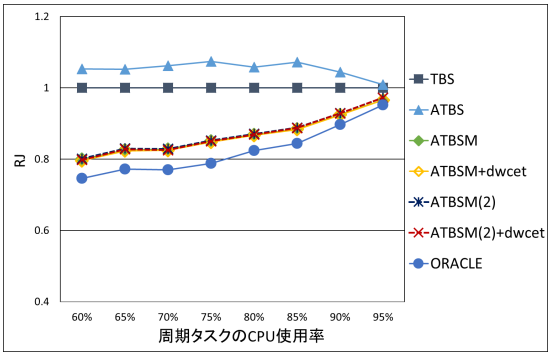

図 6.83: Tel./gsm-enc. における RJ

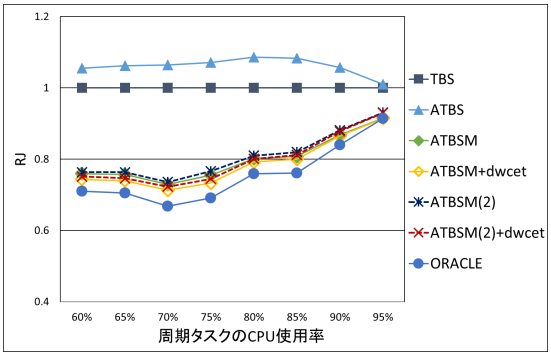

図 6.84: Tel./gsm-dec. における RJ

第7章　まとめ

7.1　結論

本研究では，Total Bandwidth Server(TBS) のデッドライン計算の際に，最悪実行時間 (WCET) の代わりに予測した実行時間 (PET) を使用することで応答時間性能を向上させる Adaptive TBS（ATBS）において，さらなる性能の向上を実現するための 2 つの手法，Adaptive TBS Modified（ATBSM）と，ATBSM+dwcet を提案した．ATBSM は，実行時間予測の高精度化によって性能の向上を狙う手法である．対象アプリケーションに対して，様々な入力による実行時間の変化を調べることから生成された予測式を使用することで，高精度な実行時間予測を実現する．PET と実際の実行時間の誤差を小さくすることで，デッドラインが短くなり，応答時間が短縮される．ATBSM+dwcet は，PET が実際の実行時間よりも短くなった (過小見積りとなった) 場合の影響を小さくすることで，性能の向上を狙う手法である．ATBS および ATBSM では過小見積となった場合，WCET を使用したデッドラインに切り替えて残りの実行をスケジュールするため，応答時間の短縮が見込めない．ATBSM+dwcet では，予測要因の値に応じた段階的な最悪実行時間（discrete WCET : dwcet）を一定の間隔ごとに用意しておき，過小見積時に予測要因の値に基づいて dwcet を選択し WCET の代わりに使用することで，過小見積時の影響を小さくする．

これらの提案手法による性能をシミュレーションによって評価した．シミュレーションにおいて，実際のシステムにおける提案手法の性能を評価するために，組込み向けベンチマーク集である MiBench の各アプリケーションを対象タスクとして使用した．MiBench を構成する 36 アプリケーションの内，27 アプリケーションを評価に使用し，それぞれ非周期タスクおよび周期タスクとする場合の評価を行った．非周期タスクとする場合は平均応答時間を，周期タスクとする場合は絶対ジッタおよび相対ジッタを評価の対象とした．評価に使用した全 27 アプリケーションの平均値において，非周期タスクの平均応答時間は，ATBS と比較した際，ATBSM によって最大 30.2%，ATBSM+dwcet によって最大 31.3%短縮された．絶対ジッタは，TBS と比較した際，ATBSM によって最大 15.1%，ATBSM+dwcet によって最大 15.9%短縮された．また，実行時間が長いインスタンスに対してより高精度に実行時間予測を行うことでジッタの短縮を狙う予測式を提案し，ATBSM，ATBSM+dwcet に適用したところ，両方ともジッタ短縮用の予測式によって絶対ジッタの短縮率が増加した．相対ジッタにおいても，絶対ジッタと同様の傾向が得られた．

7.2 今後の課題

提案手法において，実行時間予測式の生成のためにまず実行時間と比例関係にある要因（予測要因）の検出を行った．予測要因を変数とする予測式を生成し，各インスタンスの予測要因の値に合わせた実行時間を予測することで，高精度な実行時間予測が実現できた．本研究では予測要因の検出の定式化を行えていないため，各アプリケーションごとにシステム開発者が手動で予測要因を探さなければならない．システム開発者の負担を減らすため，実行時間予測式の生成を全て自動的に行う予定である．

評価方法において，実際の組込みシステムにおける提案手法の性能を評価するために，MiBench を使用して対象タスクを生成した．結果として，乱数で生成されたものではなく，実際の組込み向けアプリケーションの実行時間を反映したシミュレーションにおいて，提案手法による性能の向上を確認できた．現在は事前に計測した実行時間を対象タスクとして使用しているが，より実用上の性能を評価するために，シミュレーション上で実際のプログラムを動作させながらの評価，さらに実機上での評価をする予定である．

謝辞

本論文を作成するにあたり，熱心なご指導をして頂いた田中清史准教授に感謝致します．日々の研究においても，適切な助言を頂きながら，時間をかけて丁寧にご指導して頂いたおかげで，最後まで研究を続けることができました．また，様々な発表を場を与えてもらい，良い経験を積むことができました．本当にありがとうございました．

また，中間審査や合同ゼミ等で適切なご指導を頂いた金子峰雄教授と井口寧教授に感謝致します．

最後に，常日頃より支えて頂いた田中研究室と金子研究室の皆様，また同期の友人達に感謝致します．

参考文献

[1] C.L.Liu, James W.Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, Journal of the Association for Computing Machinery 20(1), pp.46-61, 1973.

[2] M.Spuri, G.C.Buttazzo, Efficient Aperiodic Service under Earliest Deadline Scheduling, Proc. of Real-Time Systems Symposium, pp.2–11, 1994.

[3] Kiyofumi Tanaka, Adaptive Total Bandwidth Server : Using Predictive Execution Time, Proc. of IFIP TC 10 International Embedded Systems Symposium （IESS）, Springer, pp.250–261, 2013.

[4] M.R.Guthaus, et al., MiBench : A free commercially representative embedded benchmark suite, Proc. of IEEE International Workshop on Workload Characterization（WWC-4）, pp.3–14, 2001.

[5] 住吉正人, 田辺靖貴, 天野英晴, 並列化 MiBench を利用したチップマルチプロセッサの評価, 情報処理学会研究報告システム LSI 設計技術（SLDM）, 28 号, pp.1-6, 2006.

[6] 石井 義史, 王 蔚涵, 天野 英晴, VLIW 型プロセッサにおける Mixed Power Gating の研究, 情報処理学会論文誌コンピューティングシステム, Vol.5 No.5, pp.23-32, 2012.

[7] LEI Zhao, XU Hui, IKEBUCHI Daisuke, et al., A Leakage Efficient Data TLB Design for Embedded Processors, IEICE Transactions on Information and Systems, E94-D(1), pp.51-59, 2011.

[8] Syed Muhammad Zeeshan Iqbal, et al., ParMiBench - An Open-Source Benchmark for Embedded Multiprocessor Systems, IEEE Computer Architecture Letters, Volume.9 No.2, pp.45-48, 2010.