Vol.2010-OS-113 No.14
2010/1/28

ライブラリ実装したプログラム実行速度調整法における処理の均一性の評価

境　講一[†1]　田端利宏[†1]
谷口秀夫[†1]　箱守聰[†2]

計算機ハードウェアの性能に左右されないでソフトウェアの実行速度を調整できれば，サービスの利便性は向上する．このため，これまでに，サービスを実現するソフトウェアの処理速度を調整する制御法として，プログラム実行速度調整法をライブラリに実現した．本論文では，提案制御法における評価の観点として，処理を均一に伸縮できるかについて議論する．具体的には，これまでに提案した処理の均一性を示す指標を利用して，提案制御法を評価する．評価により，提案制御法は，共存プロセスが存在せずシステムコール発行間隔が長い場合には処理の均一性が非常に高いこと，被調整プロセスを高優先度に設定することにより処理の均一性を高くできること，および共存プロセスの与える影響を明らかにした．

Evaluation of Uniformity of Processing in Mechanism for Regulating Program Execution Speed in Library

Koichi Sakai,[†1] Toshihiro Tabata,[†1]
Hideo Taniguchi[†1] and Satoshi Hakomori[†2]

If execution speed of software is regulated without concerning by performance of the computer hardware, convenience of the service better. We proposed the mechanism for regulating program execution speed in library. This paper describes whether the mechanism can improve processing uniformly. Specifically, we evaluate the mechanism based on processing uniformity. Our evaluation show high uniformity in case of non coexistence process and long interval between system call. In addition, it shows high uniformity in case of process with high priority and influence of coexistence process.

1. はじめに

近年，計算機ハードウェアの性能は著しく向上し，処理時間の短縮や複雑な処理の実行が可能になっている．一方，ソフトウェアの処理時間は，ハードウェア性能に大きく依存する．このため，高性能な計算機と低性能な計算機では，同じソフトウェアでも表示速度が大きく異なる．例えば，現在の処理内容や利用者にとって重要なメッセージを実行中に表示するプログラムを実行する場合，高速に実行されると表示内容を確認できない．これに対し，本制御機構を利用することにより，表示内容をリアルタイムに確認することができる．例えば，プログラムのデバックにおいて，現在の処理内容や返り値などの重要な値，警告メッセージの表示をリアルタイムに確認できる．また，処理内容をリアルタイムに表示し確認することは，プログラムの動作を理解することに役立つ．

上記背景を基に，文献 1) では，ライブラリでプログラム実行速度を調整制御する制御法を実装し，評価した．この方式は，ライブラリで取得可能なシステムコールの発行と終了に着目し，プログラムの各走行モードでの走行時間を基に，システムコール発行直前，またはシステムコール終了直後にプロセスを停止することにより，プログラムの実行速度を調整する．また，停止時間くりこし値の導入により停止時間のずれによる影響を抑制し，精度の高い調整を可能にした．具体的には，共存プロセスが存在しない場合には 1%以下の誤差で実行速度を調整できることを示した．しかし，プログラム実行速度の調整は，処理時間の評価だけでは有効ではない．これは，処理時間は，プログラムの開始と終了の時刻差だけで決定されるためである．実行速度の調整は，処理時間に加え，実行途中の処理が均一に調整できることも重要である．例えば，アニメーションの実行時，表示が滑らかに行われなければ，利用者の利便性は低下する．

そこで，ここでは，提案したプログラムの実行速度調整法を評価する観点として，処理を均一に伸縮できるかについて議論する．評価においては，サービス処理が均一に調整されているか否か，つまり，サービス処理を均一に伸縮できているか否かに着目する．実行速度の調整は，ゴムやバネの伸縮のように，全体が均一に伸縮していることが重要である．そこで，処理の均一性を示す指標[2] を利用して，提案制御法を評価する．評価により，提案制御

†1 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University
†2 （株）NTT データ
NTT Data Corporation

法は，共存プロセスが存在せずシステムコール発行間隔が長い場合には処理の均一性が非常に高いこと，被調整プロセスを高優先度に設定することにより処理の均一性を高くできること，および共存プロセスの与える影響を明らかにする．

従来，ハードウェア性能の範囲内で性能を変化させる研究は，I/O 要求の順番を入れ替えることでスループットを調整し，ディスク性能を向上させる研究[3]，実時間性を保証する I/O スケジューリング法の研究[4)–6)]，および，CPU 使用の予約と使用時間の制御によりスケジューリングを予想可能にする研究[7] が行われている．また，並列ファイルシステムのための I/O スケジューリング法の研究[8] や，分散システムのための I/O スケジューリング法の研究[9] が行われている．これらの研究は，ハードウェア性能を最大限に引き出すことを目的としている．これに対し，本研究は，要求された処理性能分だけの性能をプロセスに提供し，実行速度をできるだけ均一に調整する．

プロセスの資源利用量を制御する方法として，カーネル等のプログラムが処理を実行する流れをパスとして表し，この単位で資源利用量を制御する方法[10] や Resource Container を用いて資源利用量を制御する方法[11] が提案されている．文献 10) では，プログラムモジュールやメモリ資源の利用量を制御の対象としているものの，プロセッサについては対象としていない．これに対し，本研究は，プロセスを強制的に止めることによりプロセスのプロセッサ利用量を調整制御する．一方，文献 11) は，プロセッサも制御の対象としている．この機構は，OS カーネル内で制御するのに対し，提案した制御機構は，OS カーネル外のライブラリで制御を行う．

2. 実行速度調整法

2.1 基 本 方 式

実行速度を調整する方法として，これまでに，ライブラリでプログラム実行速度を調整制御する制御法[1](以降，速度調整法と略す) を提案し，ライブラリとして実装した．

基本的な調整法を**図 1** に示し，その考え方を述べる．図 1（A）は，調整を行わない場合である．処理は，プログラム処理部分（ユーザモードで走行）と OS 内処理部分（スーパバイザモードで走行）を繰り返し実行する．このプログラム走行モード（ユーザモードまたはスーパバイザモード）の変化は，システムコールの発行と終了で把握できる．つまり，ユーザモード走行したプロセッサ利用量（以降，ユーザ利用量と略す）は，システムコール終了時刻と次システムコール発行時刻の差分で推定できる．同様に，スーパバイザモード走行したプロセッサ利用量（以降，カーネル利用量と略す）は，システムコール発行時刻とそのシステムコール終了時刻の差分で推定できる．

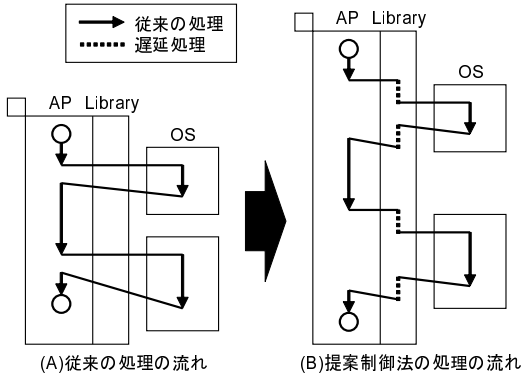


図 1 基本的な調整法

これらを利用して調整を行う．図 1（B）は，調整を行う場合である．システムコール発行時，ユーザ利用量を算出し，要求性能を考慮して制御量（以降，要求停止時間と呼ぶ）を算出する．この要求停止時間に基づき，プロセスを停止させて性能を調整する．また，同様に，システムコール終了時，カーネル利用量を算出し，プロセスを停止させて性能を調整する．なお，本制御機構は，ライブラリ部分で制御を行うため，プロセスの停止はシステムコールを利用する．

2.2 実効単位と実効精度

プロセスを停止する場合，実際にプロセスを停止する単位（以降，実効単位と呼ぶ）が問題となる．具体的には，実効単位が 10 ミリ秒の場合には，1 ミリ秒や 5 ミリ秒だけ停止することはできず，停止時間は 10 ミリ秒まで切り上げられる．この問題に対処するため，文献 1) では，停止可否閾値を導入した．これにより，要求停止時間が実効単位より小さい場合も制御できる．

また，プロセスを停止させた際の精度（以降，実効精度と呼ぶ）は，調整の精度に大きな影響を与える．具体的には，要求停止時間と実際の停止時間の誤差の積算が問題となる．文献 1) では，停止時間くりこし値を導入することにより，実効精度の影響を抑制した．

2.3 プロセッサ利用量の保証と制限

プログラムの実行速度調整は，保証と制限の観点がある．保証は，プログラムの単位時間あたりのプロセッサ利用量を保証する．これに対し，制限は，プログラムの単位時間あたり

のプロセッサ利用量を制限する．本制御手法は，ライブラリに実現するため，共存プロセスの影響を受ける．したがって，プロセッサ利用量を保証することは難しい．一方，プロセッサ利用量を制限することは可能である．具体的には，実行速度の調整に与える影響は，ディスク I/O 中の待ち時間によるカーネル利用量の増加と CPU 処理の競合によるプロセッサ利用量の低下である．これらは，被調整プロセスのプロセッサ利用量を減少させる．したがって，利用者の指定する要求性能以上にプロセッサを利用することはないため，プロセッサ使用量を制限することができる．

3. 処理の均一性

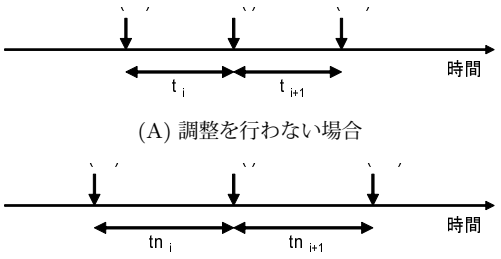
プログラム実行速度を調整する場合，処理の均一性を処理時間のみでは評価できない．これは，処理時間は，プログラムの開始と終了の時刻差だけで決定されるためである．評価においては，処理が均一に調整されているか否か，つまり，処理を均一に調整できているか否かに着目する必要がある．ゴムやバネの伸縮のように，全体が均一に伸縮していることが重要である．

一方，プログラム実行速度は，プログラムの何らかのデータ入出力により具現化される．例えば，プログラムのプロセッサ処理部分が非常に早くても，データ入力後に結果が出力されなければプログラムは動作しなかったことに等しい．

そこで，速度調整法を評価する方法として，速度調整度[2)]を利用する．速度調整度は，プログラムの入出力に着目した処理の均一性の評価尺度である．プログラム実行速度を調整したとき，プログラムの入出力に着目した処理の様子を**図 2** に示す． 実行速度を調整しない場合の入出力時刻の間隔を t，要求性能を $n\%$ としたときの入出力時刻の間隔を tn とするとき，速度調整度（η_i）を以下に示す．

$$\eta_i = \frac{tn_i}{\frac{t_i}{\frac{n}{100}}} = \frac{tn_i}{t_i} \times \frac{n}{100} \tag{1}$$

速度調整度は，1 に近いほど処理を要求性能で調整していることを示し，1 を超えると要求性能以下，1 未満では要求性能を超える性能に調整していることを指す．つまり，η_i が 1 に近いほど，i 点での調整が要求性能に合わせてうまく行えているといえる．また，速度調整度の分散が小さいほど，処理の均一性は高い．したがって，処理の均一性の評価尺度として，速度調整度の分散を利用する．つまり，i 点付近での処理の均一性は，隣接した入出力



(A) 調整を行わない場合

(B) 調整を行う場合

図 2 プログラムの入出力処理の流れ

についての速度調整度の分散が小さいほど高いといえる．

したがって，プログラム全体についての処理の均一性は，全入出力についての速度調整度の分散により判断できる．また，処理全体についての要求性能の調整程度は，全入出力についての速度調整度の平均値により判断できる．

4. 評　　価

4.1 評価環境と評価プログラム

Celeron(2.8GHz) プロセッサを搭載したマシンで FreeBSD を走行させ，実装したライブラリを用いてプログラムの処理時間を測定した．他プロセスの影響を避けるため，FreeBSD をシングルユーザモードで起動した．また，共有メモリは 64 バイト分の領域を持つものを 1 つだけ作成した．

評価プログラムを**図 3** に示す．評価プログラムは，特定のメモリ領域の値のインクリメントを繰り返す CPU 処理と read システムコールにより DK から 512 バイト読み込む入力処理を繰り返し行うものである．CPU 処理は，約 25 マイクロ秒の処理を単位とし，その繰り返し回数 (以降，CPU 回数と略す) を測定パラメータとした．入力処理は，1 回のデータ入力時間（約 200 マイクロ秒）を単位とし，CPU 処理後に 1 回だけ実行する．また，速度調整度を算出するため，データ入力終了後にクロックサイクルを取得した．なお，入力時刻の間隔数を 1000 とし，停止可否閾値を 10 ミリ秒に設定した．

4.2 基 本 評 価

本制御機構を用いた実行速度調整の処理の均一性を明らかにするため，共存プロセスは存

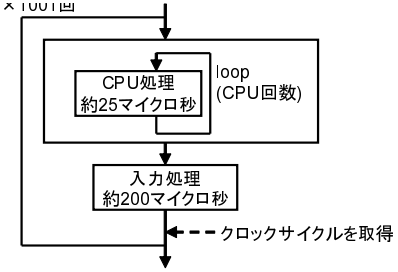


図 3　評価プログラムの処理の流れ

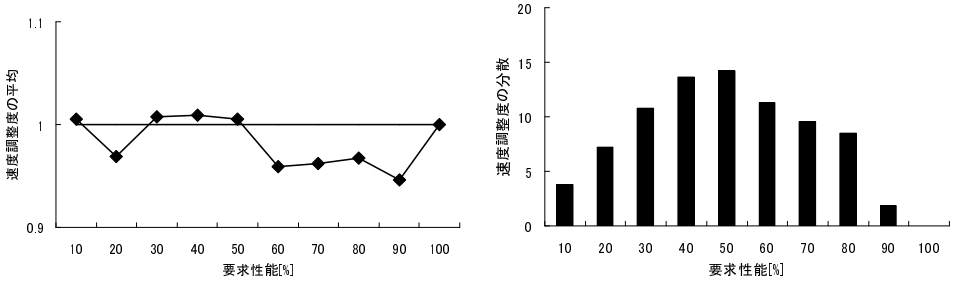
在せず，被調整プロセスが 1 つだけ存在する場合において，CPU 処理時間を I/O 処理と同じ時間だけ実行する場合，停止可否閾値と同じ時間だけ実行する場合，および停止可否閾値よりも長い時間実行する場合について評価した．具体的には，CPU 処理時間を 0.2 ミリ秒，10 ミリ秒，100 ミリ秒とした評価プログラムを用いて速度調整度を評価した．なお，要求性能は 10%から 100%まで 10%刻みで変化させた．速度調整度の平均と分散を**図 4** に示す．

図 4 より，以下のことがわかる．

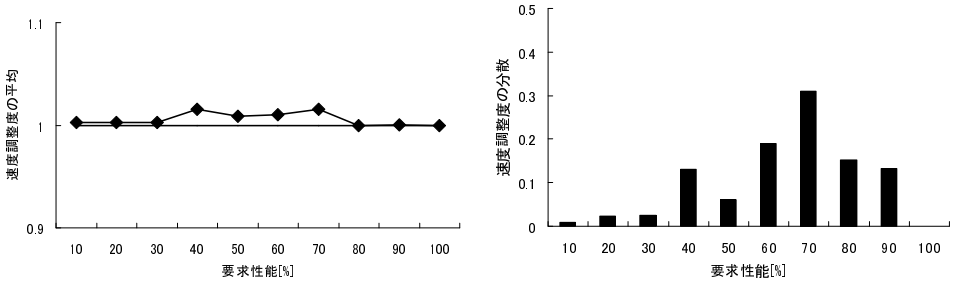
(1)　CPU 処理の割合に関わらず，速度調整度の平均は 1 に近い．具体的には，全ての場合において，処理時間比は 0.94 から 1.02 の間にある．したがって，CPU 処理の割合に関係なく，要求性能に合わせてうまく実行速度を調整している．これは，停止時間くりこし値の導入により，要求停止時間と実際の停止時間の誤差の影響を抑制しているためである．

(2)　CPU 処理の割合が小さい場合（図 4（A））, 速度調整度の分散は非常に大きい．具体的には，要求性能を 50%にした場合，速度調整度の分散は 15.1 になる．これは，実効単位の問題に起因する．CPU 処理時間が短い場合，システムコール発行ごとの要求停止時間が停止可否閾値を超えないため，停止処理は数回に 1 度まとめて行われる．したがって，停止処理を行わない場合と行う場合の I/O 発行間隔の差が大きくなり，速度調整度の分散は大きくなる．一方，停止可否閾値により，まとめて停止処理を実行するため，要求停止時間が実効単位よりも小さい場合も制御でき，速度調整度の平均は 1 に近くなる．つまり，処理全体の調整精度は高い．

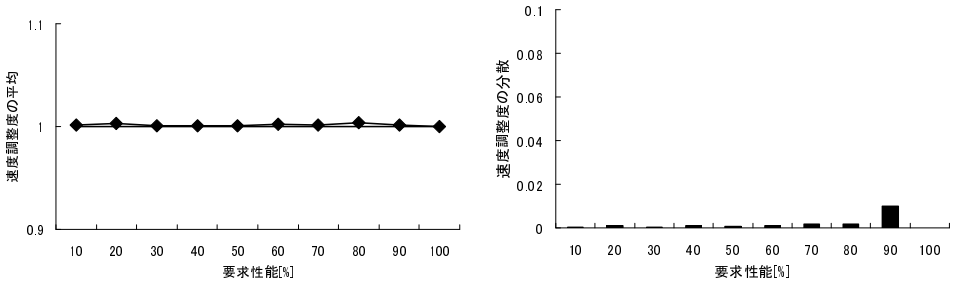
(3)　CPU 処理の割合が大きい場合（図 4（C））, 速度調整度の分散は小さく，最大で 0.01 である．これは，CPU 処理の時間が長いため，システムコール発行ごとに停止処理を実行するためである．



(A)CPU 処理時間 0.2 ミリ秒

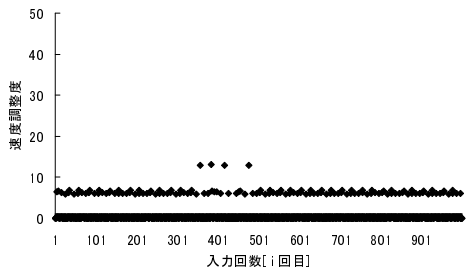


(B)CPU 処理時間 10 ミリ秒

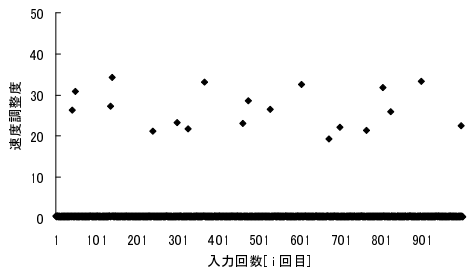


(C)CPU 処理時間 100 ミリ秒

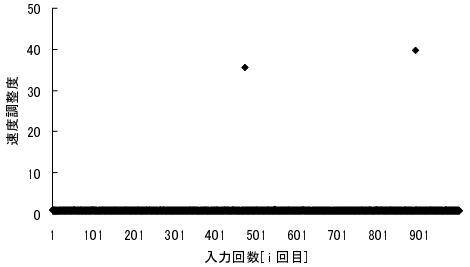
図 4　CPU 処理時間を変化させた場合の速度調整度の平均と分散



(A) 要求性能 10%



(B) 要求性能 50%



(C) 要求性能 90%

図 5 i 回目の I/O 発行時の速度調整度（η_i）（要求性能が 10%，50%，および 90%の場合）

プログラムの実行速度を調整した際，処理の均一性は，各々の入力処理の均一性が重要な尺度である．したがって，速度調整度の平均が 1 に近いこととともに，速度調整度の分散が小さいことが非常に重要である．このことから，本制御機構は，CPU 処理の割合が大きい場合には処理の均一性の非常に高い調整を行えるといえる．

これに対し，CPU 処理の割合が小さい場合，処理の均一性は低くなる．具体的には，要求性能が 50% 以下の場合，要求性能が大きいほど速度調整度の分散は大きくなる．一方，要求性能が 50% よりも大きい場合，要求性能が小さいほど速度調整度の分散は大きくなる．要求性能を 10% ，50% ，および 90% とした場合における i 回目の I/O 発行時の速度調整度（η_i）を**図 5** に示し，原因を以下で述べる．

要求性能を 10%にした場合（図 5（A）），理想の入力処理の間隔は長くなる．具体的には，100%での入力処理の間隔が 0.4 ミリ秒なので，理想の入力処理の間隔は 4 ミリ秒になる．また，要求停止時間の増加に伴い，停止処理を行う回数も増加するため，停止可否閾値を超えて 10 ミリ秒程度停止する回数が増加する．このため，速度調整度の分散は小さくなる．次に，要求性能を 90%にした場合（図 5（C）），要求停止時間の増加がほとんどないため，停止処理はほとんど行われない．しかし，理想の入力処理の間隔は 0.44 ミリ秒であり，停止処理が行われない場合の速度調整度の分散は小さくなる．最後に，要求性能を 50%にした場合（図 5（B）），10%の場合と 90%の場合の中間の特徴を持つ．つまり，理想の入力処理の間隔（0.8 ミリ秒）に対して停止処理を行う場合の分散は 10%の場合に比べ大きくなる．一方，停止処理を行わない場合の分散は 90%の場合に比べ大きくなる．したがって，要求性能が 50%以下の場合，要求性能が大きいほど速度調整度の分散は大きくなり，要求性能が 50%よりも大きい場合，要求性能が小さいほど速度調整度の分散は大きくなる．

以上のことから，処理の均一性は，CPU 処理時間，要求性能，および停止可否閾値に依存する．つまり，CPU 処理時間と要求性能から算出した要求停止時間が停止可否閾値よりも小さい場合，停止処理を行う回数が減少し，処理の均一性は低くなる．

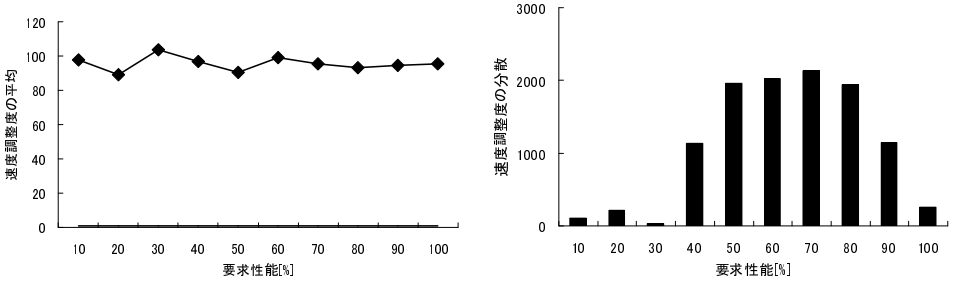
4.3 共存プロセスの影響

共存プロセスによる処理の均一性への影響を明らかにするため，共存プロセスが 3 つ存在し，被調整プロセスが 1 つだけ存在する場合において，CPU 処理時間を 0.2 ミリ秒，10 ミリ秒，100 ミリ秒とした評価プログラムを用いて速度調整度を評価した．実行速度調整法は，プロセッサ利用量の保証と制限の観点がある．本制御手法は，ライブラリに実現するため，共存プロセスの影響を受ける．したがって，プロセッサ利用量を要求性能以下に制限することはできても，要求性能分のプロセッサ利用量を保証することは難しい．しかし，被調整プロセスの優先度を高く設定し，プロセッサを優先的に割り当てることにより，プロセッサ利用量を保証できると考えられる．そこで，被調整プロセスの優先度を高く設定した場合と共存プロセスと同じに設定した場合について評価した．なお，共存プロセスの CPU 処理時間は 10 ミリ秒とし，要求性能は 10%から 100%まで 10%刻みで変化させた．速度調整度の平均と分散を**図 6** と**図 7** に示す．図 6 は，被調整プロセスと共存プロセスの優先度を同じにした場合である．図 7 は，被調整プロセスの優先度を共存プロセスよりも高く設定した場合である．

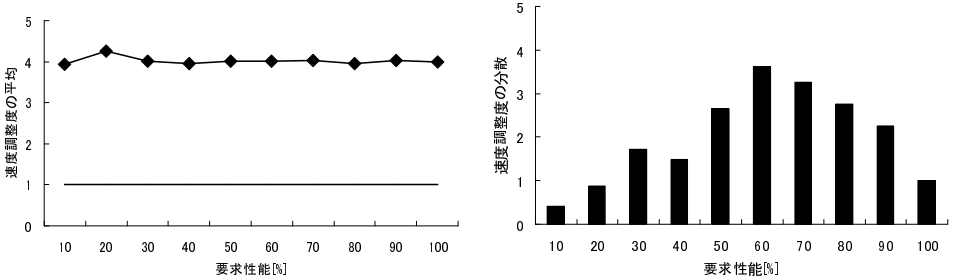
図 6 と図 7 より，以下のことがわかる．

(1) 図 6 より，被調整プロセスと共存プロセスの優先度を同じにした場合，以下のことがわかる．

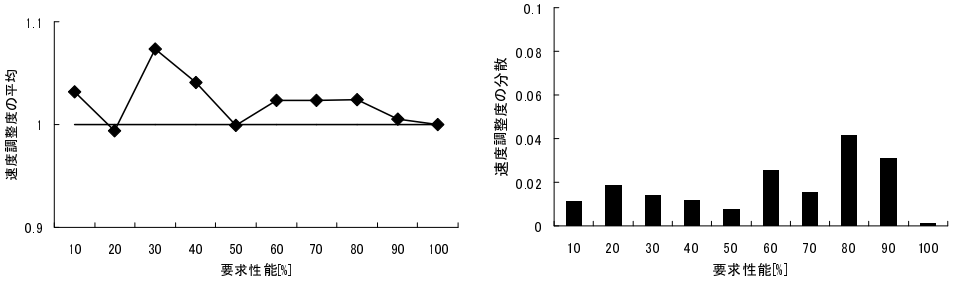
Vol.2010-OS-113 No.14
2010/1/28



(A)CPU 処理時間 0.2 ミリ秒

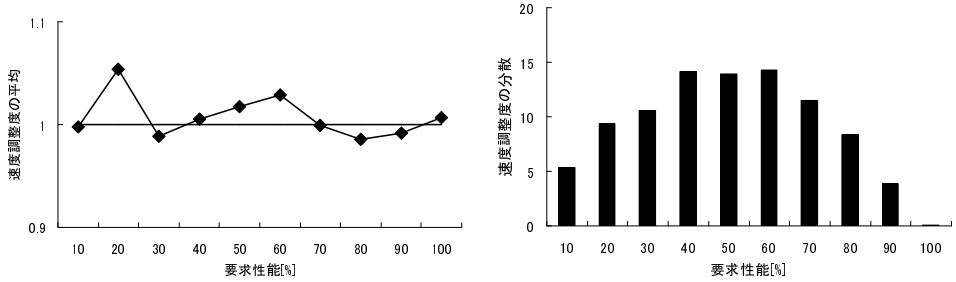


(B)CPU 処理時間 10 ミリ秒

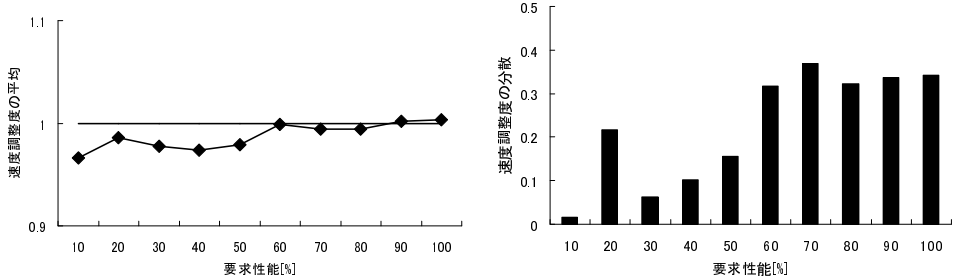


(C)CPU 処理時間 100 ミリ秒

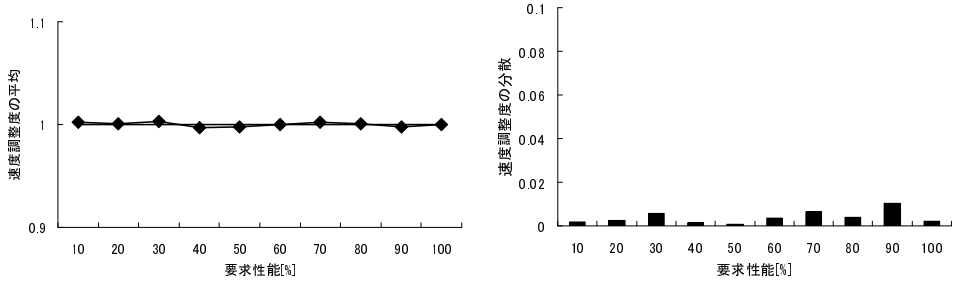
図 6　被調整プロセスと共存プロセスの優先度が同じ場合



(A)CPU 処理時間 0.2 ミリ秒



(B)CPU 処理時間 10 ミリ秒



(C)CPU 処理時間 100 ミリ秒

図 7　被調整プロセスが高優先度の場合

（**A**） 共存プロセスの CPU 処理に対して被調整プロセスの CPU 処理の割合が小さい場合（図 6（A））, 速度調整度の平均と分散は大きくなる．具体的には，平均は約 100，分散は最大で 2200 である．したがって，共存プロセスの CPU 処理に対して被調整プロセスの CPU 処理の割合が小さい場合，共存プロセスの影響を大きく受ける．

（**B**） 共存プロセスの CPU 処理に対して被調整プロセスの CPU 処理の割合が大きい場合（図 6（C））, 速度調整度の平均も分散も小さい．したがって，共存プロセスの CPU 処理に対して被調整プロセスの CPU 処理の割合が大きい場合，共存プロセスの影響を受けずに処理の均一性の高い調整を行える．

（**2**） 図 7 より，被調整プロセスの優先度を共存プロセスよりも高く設定した場合，以下のことがわかる．

（**A**） 被調整プロセスの CPU 処理の割合に関わらず，速度調整度の平均は小さい．具体的には，全ての場合において，処理時間比は 0.94 から 1.02 の間にある．したがって，被調整プロセスの優先度を高く設定した場合，CPU 処理の割合に関係なく要求性能に合わせてうまく実行速度を調整している．

（**B**） CPU 処理の割合が小さい場合（図 7（A））, 速度調整度の分散は約 15 になる．しかし，被調整プロセスと共存プロセスの優先度を同じにした場合（図 6（A））に比べて非常に小さい．具体的には，要求性能を 50%にした場合，速度調整度の分散は約 1/130 になる．

（**C**） CPU 処理の割合が大きい場合（図 7（C））, 速度調整度の分散は小さく，最大で 0.01（要求性能を 90%にした場合）である．これは，被調整プロセスと共存プロセスの優先度を同じにした場合（図 6（C））の約 1/3 である．

以上のことから，本制御機構は，共存プロセスの影響を大きく受け，処理の均一性は低下する．しかし，被調整プロセスの優先度を共存プロセスよりも高く設定することにより，共存プロセスの影響を抑制でき，処理の均一性の高い調整を行える．

5. 利用範囲と有効事例

5.1 実装方式による違い

プログラム実行速度を調整する制御法を実装する方式として，大きく 2 つある．OS カーネル内へ実装する方式[2]（以降，OS 方式と略す）と OS カーネル外にライブラリとして実装する方式[1]（以降，ライブラリ方式）である．OS 方式は，プロセスのスケジューラを工夫するため，共存プロセスの影響を受けず，調整の精度は高い．しかし，OS カーネルに手を加える場合，開発者のスキルが重要となる．例えば，開発時にバグが混入した場合，システム全体に影響を与えてしまう．一方，ライブラリ方式は，OS 方式と逆の特徴を持つ．つまり，スケジューラに手を加えないため，共存プロセスの影響を大きく受ける．また，制御のための停止時間は，OS の提供するシステムコールの精度に依存する．したがって，調整の精度は低い．しかし，バグが混入した場合，影響を受けるのは，実現したライブラリを使用するプロセスのみであり，システム全体に与える影響は小さい．また，モジュール化することにより，様々な環境で利用可能になる．

5.2 ライブラリ方式の利用可能範囲

ライブラリで実現する場合，実行速度調整の精度は，共存プロセスとプロセスを停止させるシステムコールの精度に大きな影響を受ける．一方，実行速度の調整を行う上で，調整精度は高いことが望まれる．具体的には，本制御機構を利用する場合，10%刻みで要求性能を指定することを想定している．このため，プロセッサ利用量を制限する場合，期待する速度に対して 10%以上高速に実行されないことが望ましい．また，プロセッサ利用量を保証する場合，期待する速度に対して 10%以上高速，または低速で実行されないことが望ましい．

評価により，共存プロセスの有無に関わらず，処理全体のプロセッサ利用量を要求性能以下に制限できる．また，被調整プロセスの優先度を高く設定することにより，共存プロセスが存在する場合でも，処理全体のプロセッサ利用量を保証できる．一方，処理の均一性は，CPU 処理時間，要求性能，および停止可否閾値に依存する．つまり，CPU 処理時間と要求性能から算出した要求停止時間が停止可否閾値よりも小さい場合，停止処理を行う回数が減少し，処理の均一性は低くなる．しかし，算出した要求停止時間が停止可否閾値よりも大きい場合，処理の均一性は高くなり，処理全体のプロセッサ利用量を保証または制限できる．

5.3 優先度制御との違い

優先度制御は，共存プロセスとの相対的な制御であり，走行中のプロセス数に依存する．つまり，共存プロセスが存在しない場合，優先度制御では，実行速度の調整を行うことができない．また，優先度制御は，プロセスのプロセッサ利用量に着目しないため，利用者の期待する速度に調整できない．一方，本制御機構は，プロセスのプロセッサ利用量に着目した絶対的な制御であり，走行中のプロセスの個数に依存しない．したがって，本制御機構は，共存プロセスの有無に関わらず，利用者の期待する速度に実行速度を調整することができる．

5.4 非重要サービスへの適用

計算機上では，重要サービスを提供するプロセス，比較的重要なサービスを提供するプロセス，および非重要サービスを提供するプロセスが同時に複数個走行する．例えば，各プ

ロセスが 1 つずつ存在する場合に優先度制御を行うと，重要サービスを提供するプロセスは他のプロセスに比べプロセッサを多く利用できる．しかし，比較的重要なサービスを提供するプロセスは優先度の高いプロセスにプロセッサを奪われ，走行できない．一方，本制御機構は，各プロセスに対し，利用可能なプロセッサ利用量の上限を指定（制限）できる．例えば，比較的重要なサービスを提供するプロセスに与えるプロセッサ利用量を 30%，非重要サービスを提供するプロセスに与えるプロセッサ利用量を 10%に制限することができる．この場合，重要サービスを提供するプロセスは他のプロセスに比べプロセッサを多く利用できる．また，比較的重要なサービスを提供するプロセスの優先度を高く設定することにより，指定した性能分だけプロセッサを利用（保証）することができる．

6. おわりに

プログラムの走行モードを考慮した実行速度調整法について，処理の均一性の観点から評価を行った．実行速度調整法の評価においては，処理が均一に調整されているか否か，つまり，処理を均一に調整できているか否かに着目した．速度調整度は，プログラムの入出力に着目した処理の均一性の評価尺度であり，速度調整度の分散が小さいほど，処理の均一性は高い．したがって，処理の均一性の評価尺度として，速度調整度の分散を利用した．また，処理全体についての要求性能の調整程度は，全入出力についての速度調整度の平均値により判断した．評価により，本制御機構は，CPU 処理の割合に関わらず，10%未満の誤差で調整を行えることを示した．また，CPU 処理の割合が停止可否閾値よりも大きい場合，速度調整度の平均は 1 に近く，分散は最大で 0.01 になる．一方，CPU 処理の割合が停止可否閾値よりも小さい場合，速度調整度の平均は 1 に近いものの，分散は最大で 15 になる．このことから，CPU 処理の割合が大きい場合，処理全体の調整の精度と処理の均一性が高いことを示した．また，CPU 処理の割合が小さい場合，処理全体では精度の高い調整が行えるものの，処理の均一性は低いことを示した．

次に，共存プロセスが処理の均一性に与える影響について評価し，主に 2 つのことを明らかにした．1 つは，被調整プロセスと共存プロセスの優先度が同じ場合の共存プロセスの影響である．もう 1 つは，被調整プロセスの優先度が共存プロセスよりも高い場合の共存プロセスの影響である．被調整プロセスと共存プロセスの優先度を同じにした場合，共存プロセスの影響を大きく受けて調整の精度と処理の均一性が大きく低下する．一方，被調整プロセスの優先度を共存プロセスよりも高く設定することにより，共存プロセスの影響を抑制でき，処理の均一性の高い調整を行える．

残された課題として，被調整プロセスが複数存在する場合の評価がある．

参考文献

1) 境　講一，田端利宏，谷口秀夫，箱守　聰：プログラムの走行モードを考慮した実行速度調整法の評価，情報処理学会研究報告，2009-OS-111-26（2009）．
2) 谷口秀夫：プロセススケジュールの制御によるプログラム実行速度調整法の評価，電子情報通信学会論文誌，Vol.J83-D-I，No.1，pp.184-193（2000）．
3) Povzner, A., Kaldewey, T., Brandt, S., Golding, R., Wong, T.M. and Maltzahn, C.: Efficient Guaranteed Disk Request Scheduling with Fahrrad, ACM SIGOPS Operating Systems Review，Vol.42，No.4，pp.13-25,（2008）．
4) Chang, H.P., Chang, R.I., Shih, W.K. and Chang, R.C.: GSR: A global seek-optimizing real-time disk-scheduling algorithm, Proc. Journal of Systems and Software，Vol.80，No.2，pp.198-215,（2007）．
5) Brandt, S.A., Banachowski, S., Lin, C. and Bisson, T.: Dynamic Integrated Scheduling of Hard Real-Time, Soft Real-Time and Non-Real-Time Processes, Proc. 24th IEEE International Real-Time Systems Symposium，pp.396-407,（2003）．
6) Cho, H., Ravindran, B. and Jensen, E.D.: An Optimal Real-Time Scheduling Algorithm for Multiprocessors, Proc. 27th IEEE International Real-Time Systems Symposium，pp.101-110,（2006）．
7) Jones, M.B.，Rosu, D. and Rosu, M.C.: CPU Reservations and Time Constraints:Efficient, Predictable Scheduling of Independent Activities, Proc. Symposium on Operating Systems Principles，pp.198-211,（1997）．
8) Isaila, F., Singh, D., Carretero, J., and Garcia, F.: On Evaluating Decentralized Parallel I/O Scheduling Strategies for Parallel File Systems，Lecture Notes in Computer Science，Vol.4395/2007，pp.120-130,（2007）．
9) Chen, F.: Performance of Parallel I/O Scheduling Strategies on a Network of Workstations，Proc. the Eighth International Conference on Parallel and Distributed Systems，pp.157-164,（2001）．
10) Spatscheck, O. and Petercon, L.L.: Defending Against Denial of Service Attacks in Scout，Proc. USENIX 3rd Symposium on Operating Systems Design and Implementation，pp.59-72,（1999）．
11) Banga, G. and Druschel, P.: Resource Containers: A New Facility for Resource Management in Server Systems，Proc. USENIX 3rd Symposium on Operating Systems Design and Implementation，pp.45-58（1999）．