Vol.2009-ARC-183 No.26
Vol.2009-OS-111 No.26
2009/4/23

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

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．また，実行速度の調整により，DoS 攻撃の影響を抑制できると考えられる．このため，著者らは，これまでにハードウェア性能の範囲内でプログラムの実行速度を自由に調整する制御法をライブラリとして実現した．この制御法は，プロセスの発行するシステムコールに着目し，システムコールの発行直後と終了直前にプロセスを停止することにより，実行速度を調整する．この制御法は，プロセスを停止する処理に問題があり，調整の精度が悪くなる．そこで，本稿では，これまでに提案した実行速度調整法の特徴と問題を詳細に述べ，対処法を提案する．また，対処の実装と評価により，本制御法の特徴と有効性を明らかにする．

Evaluation for a Mechanism of Regulating Execution Speed that Considered the Run Mode of Program

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 will become better. In addition, regulating execution speed inhibits influence of DoS attack. Before now, we proposed a mechanism of regulating execution speed in library. This mechanism regulates execution speed by stopping processes at just after or before the issue of system call. This mechanism has a problem in stop processing. This paper describes the problem in detail and the response of the problem. Furthermore, we implement and evaluate the proposed mechanism to clarify the characteristic and the effectiveness of it.

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，例えば，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り，利便性が低下する．また，ネットワークの普及と共に，ネットワークを介したサービス妨害攻撃 (例えば，DoS 攻撃) が増加し，深刻な問題となっている．このような攻撃を受けた場合，サービスを提供するプログラムが使用するプロセッサ資源の量を自由に調整 (保証あるいは制限) できれば，DoS 攻撃の影響を抑制できる．例えば，Web サーバの管理者は，攻撃を受けている Web サーバの実行速度を低速にする (制限) ことにより，プロセッサ資源の占有を抑制できる．

上記背景を基に，文献 1) では，プログラムの走行モードを考慮したプログラムの実行速度調整法を提案し，OS カーネル外のライブラリとして実現する方式を提案した．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．

ここでは，提案したプログラムの実行速度調整法の特徴と問題を述べ，その対処を述べる．既存手法は，要求した停止時間と実際の停止時間の差分による調整の誤差が問題となる．具体的には，停止から復帰したプロセスが再走行するまでに遅延が発生する．これにより，プログラムは，利用者の期待する実行速度よりも遅く実行される．そこで，この問題への対処手法を提案し，ライブラリとして実装した．また，提案手法を 3 つの段階に分けて比較評価することにより，本対処の特徴と有効性を明らかにする．

2. 実行速度調整法

2.1 基本方式

著者らは，これまでに，走行モードを考慮したプログラムの実行速度調整法[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．速度調整法の基本方式を図 1 に示

†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

© 2009 Information Processing Society of Japan

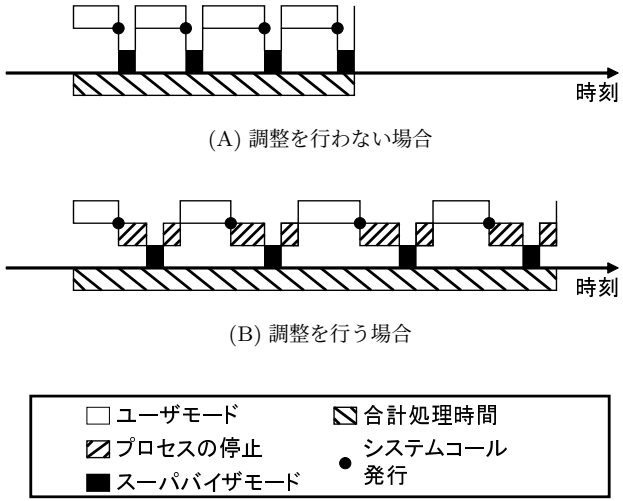


図 1 走行制御の様子

す．図 1(A) は調整を行わない場合のプロセスの走行の様子を示す．また，図 1(B) は調整を行う場合の調整制御の様子を示す．

基本的な方式として，プロセスがユーザモードとスーパバイザモードで走行した時間をそれぞれ計測し，その時間を調整することでプログラム実行速度を調整する．具体的には，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，システムコール処理終了から次回システムコール発行までの時間をユーザモードでの走行，OS が応用プログラム (以降，AP と略す) から依頼されたシステムコール処理を実行する時間をスーパバイザモードでの走行として，経過時間を計測する．計測した経過時間を基に，利用者の指定する要求性能から制御量を算出する．算出した制御量 (以降，要求停止時間と略す) 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．なお，以降では，システムコール発行直前に行う調整をユーザ調整，システムコール終了直後に行う調整をカーネル調整と略す．

2.2 要求性能の指定法

利用者は，実行速度を調整したいプロセス (以降，被調整プロセスと呼ぶ) を希望する実行速度で実行したい．また，プロセス実行途中で実行速度を調整したいという要求もある．このため，プロセス実行途中に他のプロセス (以降，性能指定プロセスと呼ぶ) から自由に速度調整を行うことが要求される．

この要求を満足するため，要求性能の指定法として，共有メモリを利用する．利用者は，共有メモリに要求性能を設定することにより，実行速度の調整が可能となる．共有メモリを利用することにより，被調整プロセスと性能指定プロセス間で要求性能情報の共有が可能となる．また，共有メモリを介して，値の設定や参照を自由に行うことができる．実行速度の調整は，被調整プロセスがライブラリ内で共有メモリにアクセスし，要求性能を取得することで行う．また，システムコールの発行ごとに共有メモリの値を参照することで，実行途中にプログラム実行速度を変更できる．

2.3 プロセス停止処理

2.3.1 プロセス停止法

プロセスの停止は，nanosleep システムコールを利用して行う．これは，以下の 2 つの利点が挙げられる．1 つは，プロセスの停止中に CPU 資源を消費しないことである．このため，共存プロセスへの影響を抑制できる．もう 1 つは，停止時間を細かく指定できることである．nanosleep システムコールは，停止時間の指定が 1 ナノ秒と細かく指定できるため，調整の精度の向上が見込まれる．

2.3.2 nanosleep システムコールの精度による問題

nanosleep システムコールは，以下の仕様を持つ．

(仕様 1) 要求停止時間がソフトウェアクロックの倍数になっていない場合，実際の停止時間 (以降，実停止時間と略す) は一番近い倍数に切り上げられる．

(仕様 2) 指定された時間経過後，CPU が呼び出し元のプロセスを再び実行できるようになるまでに遅延が入る．

(仕様 1) より，nanosleep システムコールの精度は，ソフトウェアクロックの分解能 (以降，分解能と略す) に等しい．一方，要求停止時間は 1 ナノ秒から指定できる．このため，算出した要求停止時間が分解能よりも小さい場合，制御を行うことができない．分解能による実停止時間への影響を図 2 に示し，以下に説明する．図 2(a) は理想的な調整の様子，(b) は実際の調整の様子を示す．

例えば，分解能が 10 ミリ秒であり，基本方式に従い算出した要求停止時間が 5 ミリ秒の場合，nanosleep システムコールにより 5 ミリ秒停止することが望ましい (図 2(a))．しかし，(仕様 1) により，実停止時間は分解能に切り上げられる．このため，実際は分解能と等しい時間だけ停止する．(図 2(b))．

なお，ソフトウェアクロックの分解能は，カーネルのバージョンとハードウェアプラット

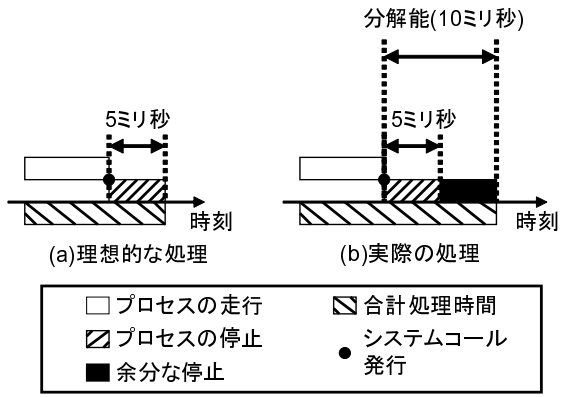


図 2 分解能による実停止時間への影響

フォームに依存する．例えば，i386 系アーキテクチャにおいて Linux 2.4.x 以前のソフトウェアクロックの分解能は 10 ミリ秒であり，Linux 2.6.0 以降は 1 ミリ秒である．

2.3.3 閾値の導入

2.3.2 項で述べた問題に対処するため，文献 1) では，要求停止時間を (s) とした場合に，最小停止時間 (W)，閾値 (L)，合計要求停止時間 (S) を設けて制御を行った．閾値を導入した停止処理の流れと共有メモリの関係を図 3 に示す．ここで，最小停止時間 (W) を分解能とし，閾値 (L) は最小停止時間 (W) よりも大きいとする．また，合計要求停止時間 (S) は，n 回分の要求停止時間 (s) を合計したものである．具体的には，算出した要求停止時間 (s) と共有メモリに保存してある合計要求停止時間 (S) を加算し，閾値 (L) と比較する．合計要求停止時間 (S) が閾値 (L) よりも小さい場合，合計要求停止時間を共有メモリに保存する．一方，合計要求停止時間 (S) が閾値 (L) よりも大きい場合，合計要求停止時間 (S) から分解能を減算し，nanosleep システムコールを発行する．また，nanosleep システムコール処理終了後，共有メモリの合計要求停止時間 (S) を初期化する．

これにより，要求停止時間が分解能よりも小さい場合も制御することが可能になる．また，合計要求停止時間 (S) から分解能を減算することにより，実停止時間の切り上げによる影響を抑制できる．

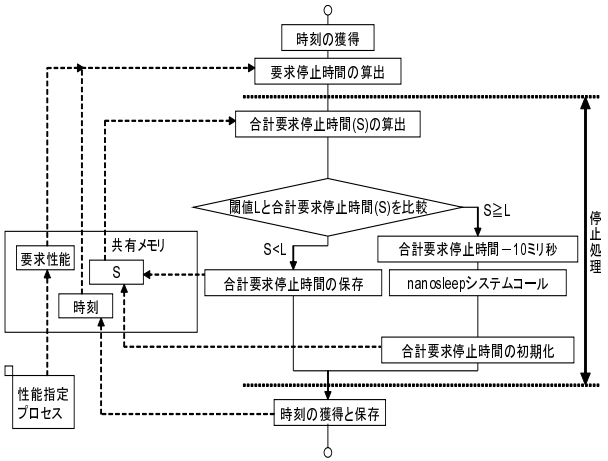


図 3 既存の停止処理の流れ

3. 既存手法の問題と対処

既存手法では，閾値を導入することにより，実停止時間の切り上げによる影響を抑制した．一方，この方法では，(仕様 2) による影響を抑制することはできない．

本章では，(仕様 2) による問題を詳細に述べ，その対処法を提案する．

3.1 問　　題

nanosleep システムコールを利用しプロセスを停止した場合，(仕様 2) により再走行するまでに遅延が入る．具体的には，停止から復帰したプロセスは，タイマ割り込みにより呼び出されたスケジューラが再スケジューリングすることにより走行を再開する．このため，停止から復帰してからタイマ割り込みが発生するまで，プロセスの再走行は遅延される．この様子を図 4 に示し，以下に説明する．図 4(a) は理想的な調整の様子，(b) は実際の調整の様子を示す．

例えば，タイマ割り込み間隔が 20 ミリ秒であり，実停止時間が 10 ミリ秒の場合，nanosleep システムコールにより 10 ミリ秒停止することが望ましい (図 4(a))．しかし，(仕様 2) により，タイマ割り込みの発生により再スケジューリングされるまで，プロセスは再走行することができない．このため，実際はタイマ割り込みが発生するまで再走行できない．(図 4(b))．また，再走行するまでの遅延時間は，復帰処理とタイマ割り込みの間隔に依存する．

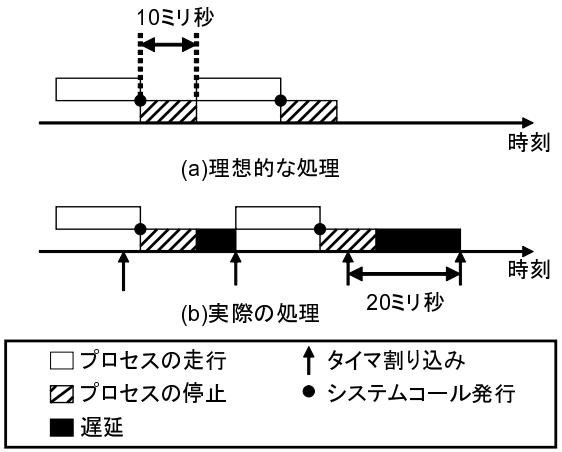


図 4 再走行までの遅延の様子

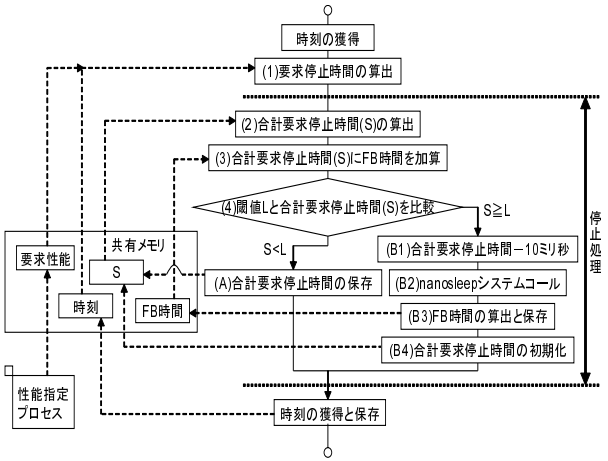


図 5 停止処理の流れと共有メモリの関係

このため，遅延時間を事前に知ることは難しく，遅延をなくすことは難しい．一方，プログラムの実行速度を調整するうえで，要求停止時間と停止時間の差分は，調整の精度に大きな影響を与える．

3.2 フィードバック法の導入

3.1 項で述べた (仕様 2) による遅延の影響を抑制するため，フィードバックを利用した制御法 (以降，FB 法と略す) を提案する．FB 法は，算出した合計停止時間 (S) と実際の停止時間の差分を次回合計停止時間 (S) にフィードバックさせる．具体的には，FB 時間 = 合計停止時間 (S)− 実際の停止時間とし，次回停止処理時に算出する合計停止時間 (S) に FB 時間を加算する．これにより，遅延の影響を抑制でき，調整精度の向上が見込まれる．

4. 実装と評価

4.1 実　　装

4.1.1 実 装 内 容

3.2 項で述べた対処を，FreeBSD 4.3-RELEASE(以降，FreeBSD と略す) のライブラリに実現した．時刻の獲得は，rdtsc 命令でシステムクロックを取得することにより実現した．また，要求停止時間を算出するプログラムと，ソフトウェア割込み (int0x80 命令) を発行して nanosleep システムコールを発行するプログラムを追加した．要求停止時間の算出は，要求性能が 100%の場合は行わない．これにより，要求性能を 100%とした場合のオーバヘッドを削減できる．さらに，要求性能の設定と合計要求停止時間，FB 時間，および取得した時刻の保存を行うため，共有メモリのアタッチとデタッチを行うプログラムを追加した．

なお，閾値と FB 法の有効性を明らかにするため，以下の 3 つのライブラリを実現した．

(1) 要求停止時間の値に関係なく nanosleep システムコールを発行する場合 (以降の各図では基本と略す)

(2) 閾値 (L) を設けて制御を行う場合 (以降の各図では基本+閾値法と略す)

(3) 閾値 (L) を設け，FB 法を利用し制御を行う場合 (以降の各図では基本+閾値法+FB 法と略す)

また，上記に示したように，実装したライブラリは，システムクロック，ソフトウェア割込み，nanosleep システムコール，および共有メモリのみを使用している．このため，これらに相当する機能を有する OS 上であれば容易に利用可能である．

4.1.2 停 止 処 理

停止処理の流れと共有メモリの関係を図 5 に示し，以下に説明する．ここで，本制御法を実現した FreeBSD の分解能は 10 ミリ秒である．このため，最小停止時間 (W) は 10 ミリ秒とする．また，タイマ割り込みは 10 ミリ秒間隔 (デフォルト) で発生する．

(1) 共有メモリ内のユーザ調整開始時刻またはカーネル調整開始時刻，および要求性能を

取得し，要求停止時間を算出する．

(2) 共有メモリ内の合計要求停止時間 (S) と算出した要求停止時間 (s) を加算する．また，加算結果を合計要求停止時間 (S) とする．

(3) 合計要求停止時間 (S) に FB 時間を加算する．また，加算結果を合計要求停止時間 (S) とする．

(4) 合計要求停止時間 (S) と閾値 (L) を比較し，以下の処理を行う．

(A) 合計要求停止時間 (S)< 閾値 (L) の場合，合計要求停止時間 (S) を共有メモリに保存する．

(B) 合計要求停止時間 (S) ≧閾値 (L) の場合，以下の処理を行う．

(B1) 合計要求停止時間 (S) から 10 ミリ秒減算する．

(B2) nanosleep システムコールを発行する．

(B3) 実停止時間を計測し，FB 時間を算出する．また，算出した FB 時間を共有メモリに保存する．

(B4) 共有メモリ内の合計要求停止時間 (S) を初期化する．

4.2 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 6 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 2 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，その繰り返し回数 (以降，I/O 回数と略す) を測定パラメータとした．また，入力時刻の間隔数を 1000 とした．

4.3 評価の観点

閾値を導入したことによる調整への影響を明らかにするため，基本と基本+閾値法において，調整の精度について比較評価した．また，FB 法の有効性を明らかにするため，基本+閾値法と基本+閾値法+Fb 法において，調整の精度について比較評価した．

各評価では，様々な AP が走行することを考慮し，CPU 回数と I/O 回数を変化させることにより，CPU 処理と入力処理の比率を変化させた．また，ユーザモードとスーパバイザモードの両走行モードの調整（以降，両調整と略す）について評価を行った．

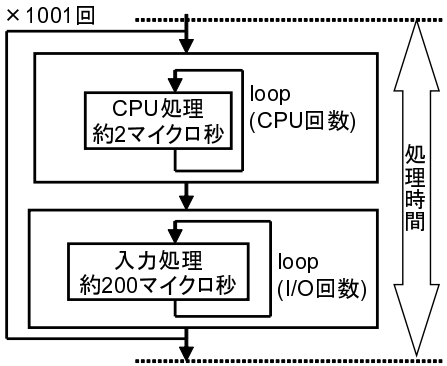


図 6 評価プログラム

表 1 入力処理時間と比率を変化させた場合の CPU 処理時間

	比率		
入力処理時間	0:1	1:4	1:2
2 ミリ秒	0 ミリ秒	0.5 ミリ秒	1 ミリ秒
20 ミリ秒	0 ミリ秒	5 ミリ秒	10 ミリ秒
200 ミリ秒	0 ミリ秒	50 ミリ秒	100 ミリ秒
	比率		
入力処理時間	1:1	2:1	4:1
2 ミリ秒	2 ミリ秒	4 ミリ秒	8 ミリ秒
20 ミリ秒	20 ミリ秒	40 ミリ秒	80 ミリ秒
200 ミリ秒	200 ミリ秒	400 ミリ秒	800 ミリ秒

入力処理時間と比率 (CPU 処理時間:I/O 処理時間) を変化させた場合の CPU 処理時間を表 1 に示す．例えば，入力処理時間が 2 ミリ秒の場合に比率を 1:2 とする場合，CPU 処理時間が 1 ミリ秒となるように CPU 回数を変化させる．

なお，以降の各図の処理時間比は，

$$\text{処理時間比} = \frac{\text{実測値}}{\text{理論値}} \tag{1}$$

である．実測値は測定した処理時間であり，理論値は当該の要求性能で理想的に実行速度の調整を行ったと仮定して算出した処理時間である．したがって，処理時間比が 1 より大きい

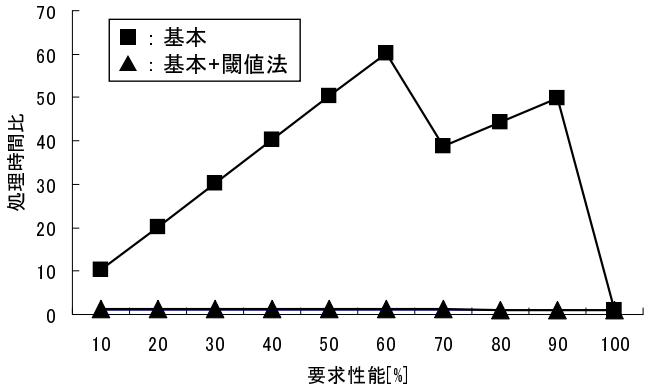


図 7 閾値の導入による調整の精度への影響

場合は，プログラムが理論値よりも長時間で処理を行った場合である．一方，1 より小さい場合は，プログラムが理論値よりも短時間で処理を行った場合である．

4.4 評 価 考 察

4.4.1 閾値の有効性

閾値を導入したことによる調整の精度への影響を明らかにするため，基本と基本+閾値法において，調整の精度について比較評価した．具体的には，基本と基本+閾値法それぞれにおいて，入力処理時間を 2 ミリ秒，比率を 1:1 とした評価プログラムを用いて，要求性能を 10%から 100%まで 10%刻みで変化させて評価を行った．評価結果を図 7 に示す．

図 7 より，以下のことが分かる．

(1) 閾値を導入しない場合 (基本)，処理時間比は最大で理論値の約 60 倍になる．これは，分解能の影響による実停止時間の切り上げを考慮せず，システムコールの発行，または終了ごとに毎回停止処理を行うためである．

(2) 閾値を導入しない場合 (基本)，要求性能が 70%以上の場合と 60%以下の場合で処理時間比が大きく異なる．これは，要求停止時間の算出精度に起因する．具体的には，要求性能が 70%以上の場合，ユーザ調整の際に算出した要求停止時間は算出精度 (1 マイクロ秒) 未満となり，停止処理は行われない．一方，要求性能が 60%以下の場合，算出した要求停止時間は算出精度 (1 マイクロ秒) よりも大きくなる．このため，停止処理は実停止時間を 10 ミリ秒に切り上げて行われる．これにより，要求性能が 60%の場合，70%の場合に比べ，処理時間比が大きくなる．

(3) 閾値を導入する場合 (基本+閾値法)，処理時間比はほぼ 1 になる．これは，合計要求停止時間が 10 ミリ秒以上になるまで停止処理を行わず，停止処理を行う場合は合計要求停止時間から 10 ミリ秒減算することにより，実停止時間の切り上げによる処理時間の増加を抑制したためである．

以上のことから，閾値を導入することにより，分解能の影響を抑制できる．

4.4.2 FB 法の有効性

FB 法の有効性を明らかにするため，基本+閾値法と基本+閾値法+FB 法において，調整の精度について比較評価した．具体的には，基本+閾値法と基本+閾値法+FB 法それぞれにおいて，入力処理時間を 200 ミリ秒とし，比率を 0:1 から 4:1 まで変化させた評価プログラムを用いて，要求性能を 10%から 100%まで 10%刻みで変化させて評価を行った．評価結果を図 8 に示す．

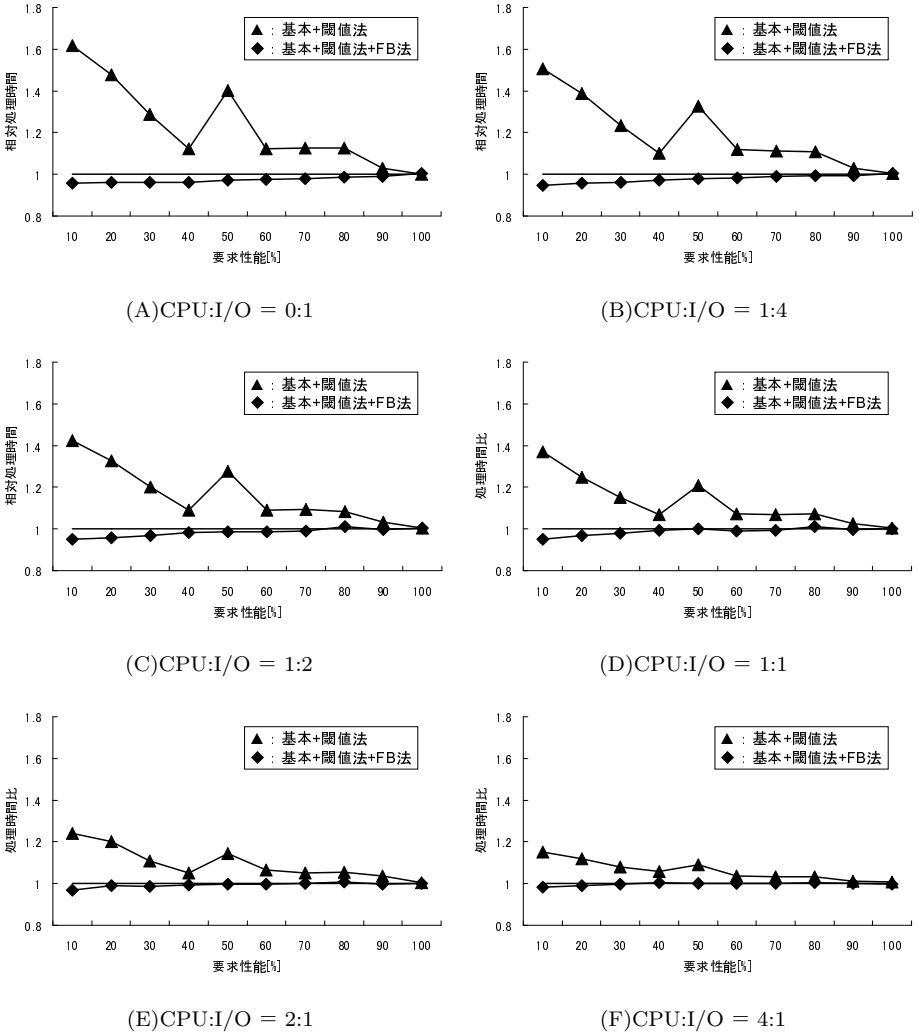
図 8 より，以下のことが分かる．

(1) 閾値のみで制御を行う場合 (基本+閾値法)，FB 法を利用して制御を行う場合 (基本+閾値法+FB 法) に比べ，調整の精度は悪くなる．特に，要求性能が低い場合 (要求性能 50%以下) に精度が悪い．これは，要求性能が低い場合，停止処理の回数が多くなり，再走行までの遅延の影響を大きく受けるためである．

(2) 閾値のみで制御を行う場合 (基本+閾値法)，I/O 処理の割合が小さい場合 (図 8(F)) に比べ，大きい場合 (図 8(A)) のほうが調整の精度が悪くなる．具体的には，基本+閾値法の処理時間比は，I/O 処理の割合が大きい場合 (図 8(A)) に最大で理論値の 1.6 倍，I/O 処理の割合が小さい場合 (図 8(F)) に最大で理論値の 1.15 倍となる．本制御法は，システムコールに着目して調整を行うため，再走行までの遅延の総時間はシステムコールの発行割合に影響を受ける．このため，CPU 処理に対し I/O 処理の割合が大きくなると，調整の制度が悪くなる．

(3) 閾値のみで制御を行う場合 (基本+閾値法)，要求性能が 50%の場合に調整の精度が悪くなる．これは，停止処理を行う直前までの処理時間に依存する．要求性能を 50%とした場合の調整の様子を図 9 に示し，以下に説明する．

(仕様 2) より，前回のプロセス再走行の契機は，タイマ割り込み発生時である．要求性能を 50%とした場合，要求停止時間は約 200 マイクロ秒となるため，合計要求停止時間が閾値よりも大きくなるまで，つまり，50 回の I/O 処理を行う．このとき，I/O 処



(A)CPU:I/O = 0:1　(B)CPU:I/O = 1:4

(C)CPU:I/O = 1:2　(D)CPU:I/O = 1:1

(E)CPU:I/O = 2:1　(F)CPU:I/O = 4:1

図 8　FB 法の導入による調整の精度への影響

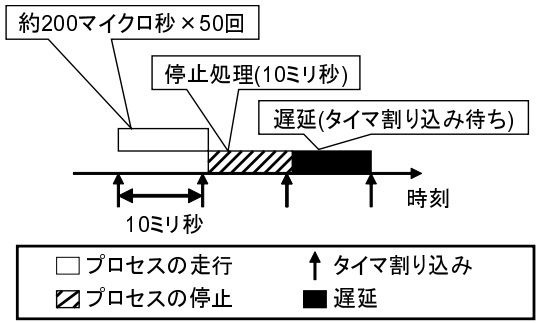


図 9　要求性能を 50%とした場合の調整の様子

理の総時間は約 10 ミリ秒である．一方，(仕様 1) より，実停止時間は 10 ミリ秒に切り上げられるため，プロセスは 10 ミリ秒停止する．ここで，プロセス再走行の契機は，タイマ割り込みによる再スケジュールである．このため，次回のタイマ割り込みまで再走行できず，調整の精度が悪くなる．

(4)　FB 法を利用して制御を行う場合 (基本+閾値法+FB 法)，I/O 処理の割合に関わらず，処理時間比は最大で理論値の 1.02 倍，最小で理論値の 0.95 倍となる．また，処理内容に関わらず，調整の精度は安定している．

以上のことから，FB 法を導入することにより，再走行までの遅延の影響を抑制でき，被調整プロセスの処理内容に関わらず，安定して精度のよい調整を行える．

5. 関 連 研 究

5.1 性 能 調 整

プログラムの性能を調整する方法として，動的電圧スケーリング (DVS) という CPU 速度をハードウェアレベルで動的に変更する機能[2] や，仮想化技術を利用し CPU 速度を制限する方法[3] がある．

DVS は，CPU チップを動作させるクロックシグナルを周期的に止めることで CPU 速度を遅くし，プログラムの実行速度を調整する．このため，利用者の希望する速度に正確に調整できる．しかし，DVS は実現できる速度の種類が限られており，自由に調整を行うことができない．

仮想化技術を利用する方法として，FoxyLargo がある．これは，仮想マシンに与える CPU

資源を制限することにより，VM 上で走行するプログラムの実行速度を調整する．このため，VM 上で走行するプログラムは，全て実行速度調整の対象となる．一方，本研究は，共有メモリを利用することにより，被調整プロセスと共存プロセスを区別する．このため，プロセスごとの実行速度調整を可能にする．また，停止処理を行うシステムコールを選別することにより，プロセスの性質に合わせて実行速度を調整することができる．

5.2 スケジューリング

従来，I/O スケジューリングについての研究は，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[4]，実時間性を保証する I/O スケジューリング法の研究[5),6)]，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]，および多重化 I/O の実行間隔制御において，スケジュール操作を行うことにより CPU 資源を効率的に使用する研究[8)] が行われている．また，近年，ネットワークの普及や処理高速化により，並列分散システムが多く用いられている．このため，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[9)] 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[10)] 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．これらに対し，本研究は，ハードウェア性能の範囲で要求された処理性能分だけの性能をプロセスに提供し，実行速度を調整する．

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の問題と対処について述べた．文献 1) で提案した実行速度調整法は，要求停止時間と分解能の関係による調整精度の低下を抑制するため，閾値を設けて制御を行った．しかし，閾値による制御だけでは，再走行するまでの遅延の影響を抑制することができない．この遅延の総時間が大きい場合，要求の精度が悪くなる．この問題に対処するため，フィードバックを利用し制御 (FB 法) を行った．FB 法は，合計要求停止時間と実停止時間の差分を次回要求停止時間にフィードバックさせる．これにより，再走行までの遅延の影響を抑制でき，精度の向上が見込まれる．

対処手法を実現し評価した．算出した停止時間に関わらず停止処理を行う場合，調整の精度は非常に悪くなる．これは，分解能の影響による実停止時間の切り上げを考慮せず，システムコールの発行，または終了ごとに毎回停止処理を行うためである．これに対し，閾値を導入して制御を行うことにより，分解能の影響を大きく抑制できる．しかし，再走行までの遅延による影響を抑制することはできない．そこで，さらに FB 法を導入し制御を行うことにより，調整の精度が向上する．これは，フィードバックを利用することにより，合計要求停止時間と実停止時間の差分を小さくでき，再走行までの遅延の影響を抑制することができるためである．

残された課題として，共存プロセスが存在する場合の評価と実 AP による評価がある．

謝辞 本研究の一部は，科学研究費補助金基盤研究 (B)(18300010)，および科学研究費補助金若手研究 (B)(課題番号 18700030) による．

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，”情報処理学会研究報告 2008-OS-110, vol.2009, no.6, pp.99-106 (2009.01)．
2) Tam.D.，Tsang.W，Drula.C，“ Dynamic Voltage Scaling in Mobile Devices，” CSC2228 Project Final Report，(2003)．
3) Tetsuya Yoshida, Hiroshi Yamada, and Kenji Kono, “ FoxyLargo: Slowing Down CPU Speed with a Virtual Machine Monitor for Embedded Time-Sensitive Software Testing，”International Workshop on Virtualization Technology，(2008.6)．
4) Anna Povzner，Tim Kaldewey，Scott Brandt，Richard Golding，Theodore M. Wong，Carlos Maltzahn, “ Efficient Guaranteed Disk Request Scheduling with Fahrrad，”EuroSys European Conference on Computer Systems，pp.27–40，pp.13-25，(2008.4)
5) H.P.Chang，R.I.Chang，W.K.Shih，R.C.Chang “ GSR: A global seek-optimizing real-time disk-scheduling algorithm，”The Journal of Systems and Software，pp.198-215，(2007)．
6) S.A.Brandt，S.Banachowski，C.Lin，and T.Bisson.“ Dynamic integrated scheduling of hard real-time,soft real-time and non-real-time processes，”Real-Time Systems Systems2003，pp.396-407，(2003.12)．
7) M.B.Jones，D.Rosu，and M.-C.Rosu, “ CPU reservations and time constraints: Efficient，predictable scheduling of independent activities，”Symposium on Operating Systems Principles，pp.198-211，(1997.10)．
8) 河合栄治，門林雄基，山口　英,“ 多重化 I/O の実行間隔制御におけるスケジュール操作による確定的なプロセッサ利用の実現，”情報処理学会研究報告, 2003-OS-93，vol.2003, No.42, pp.33-40 (2003)．
9) Florin Isaila，David Singh，Jesus Carretero，and Felix Garcia, “ On Evaluating Decentralized Parallel I/O Scheduling Strategies for Parallel File Systems，”Lecture Notes in Computer Science，Vol.4395/2007，pp.120-130，(2007)．
10) Fangyu Chen，and Shikharesh Majumdar,“ Performance of Parallel I/O Scheduling Strategies on a Network of Workstations，”Eighth International Conference on Parallel and Distributed Systems，pp.157-164，(2001.6)．