社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2009－OS－110 (14)
2009／1／29

プログラムの走行モードを考慮した実行速度調整

境 講一[†] 田端利宏[†] 谷口秀夫[†] 箱守 聰[††]

[†] 岡山大学大学院自然科学研究科
[††](株)NTT データ

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，DoS 攻撃からの復旧に役立つと考えられる．一方，プログラムは，ユーザモードとスーパバイザモードの 2 つのモードで走行する．このため，プログラムの走行モードを考慮することで，精度の高い実行速度の調整が可能である．本稿では，プログラムの走行モードを考慮した実行速度調整の実現方式を述べる．具体的には，ユーザモードで走行するプログラムの実行速度調整の基本方式とスーパバイザモードで走行するプログラムの実行速度調整の基本方式を述べ，両モードの基本方式を組み合わせた調整方式を述べる．また，プロセスの停止方法と，調整割合の指定法について述べる．さらに，ライブラリとして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．

A Mechanism of Regulating Execution Speed that Considered the Run Mode of Program

Koichi Sakai[†] Toshihiro Tabata[†] Hideo Taniguchi[†] Satoshi Hakomori[††]

[†] Graduate School of Natural Science and Technology, Okayama University
[††] NTT Data Corporation

If execution speed of software is regulated without concerning by performance of the computer hardware, Convenience of the service better. In addition, regulating execution speed inhibits influence of DoS attack, and help restoration from DoS attack. On the other hand, program runs in user mode and supervisor mode. Therefore, considering run mode of program enables high accuracy regulating execution speed. We propose a mechanism of regulating execution speed in consideration of the run mode of program. Specifically, our method can regulate the execution speed of program in user mode, supervisor mode, and both modes. This paper describes a method of stopping process and a method of setting regulated performance. Furthermore, we implement and evaluate the proposed mechanism in library 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，ソフトウェアの実行性能は，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るため，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 (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，プログラムの実行速度を自由に調整できれば，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ことにより，CPU 資源の占有を抑制できる．

そこで，著者らは，計算機ハードウェア性能の範囲で，利用者が求める速度でプログラム実行速度を自由に調整する方法について，研究を進めている．現在までに，オペレーティングシステム (以降，OS と略す) カーネル内でプログラム実行速度を調整する方式[1),2)]を実現した．しかし，OS カーネル内に実現する場合，OS カーネルを再構築する必要があり，実行速度を調整できる環境が限られる．そこで，OS カーネル外のライブラリとして実現する方式を文献 3) で提案した．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，システムコール終了から次回システムコール発行までのユーザプログラムの処理時間をプロセッサ使用量として調整を行う．この方式では，ユーザプログラムの処理時間を基に調整のための待ち時間を決定するため，調整の精度は応用プログラム (以降，AP と略す) のプロセッサ使用量に依存する．このため，例えば，システムコールを発行し続けるプログラムの実行速度をうまく調整することができない．一方，OS カーネルが処理を行う時間を調整す

ることができれば，システムコールを発行し続けるプログラムの実行速度も調整可能となる．

そこで，本稿では，プログラムの走行モードを考慮した実行速度調整の実現方式ついて述べ，ライブラリとして実装する．具体的には，プロセスはユーザプログラムの処理を行うユーザモードと OS カーネルが処理を行うスーパバイザモードの 2 つのモードで走行することに着目し，システムコールの発行直前と直後に強制的に走行を停止させることにより，各走行モードにおいてプログラムの実行速度を調整する．この方法をライブラリとして実装した．また，試作したライブラリについて，オーバヘッドと調整の精度の観点で評価した．

2. プログラム実行速度の調整

2.1 基本方式

プログラムを実行する場合，プロセスは 2 つのモードで走行する．1 つは，ユーザプログラムの処理を行うユーザモードである．もう 1 つは，OS が処理を行うスーパバイザモードである．このため，プログラム実行速度を調整する場合，2 つのモードを考慮して調整を行う必要がある．

基本方式を図 1 に示す．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．図 1(B) はユーザモード走行時の調整制御の様子を示し，図 1(C) はスーパバイザモード走行時の調整制御の様子を示す．また，図 1(D) は両方式を組み合わせた場合の調整制御の様子を示す．各モード走行時の調整について，以下で説明する．

[ユーザモード走行時の調整]

ユーザモード走行時は，ユーザプログラムの処理を実行する．そこで，ユーザモード走行時の調整 (以降，ユーザ調整と呼ぶ) の基本的な方式として，プロセスがユーザプログラムを実行した時間を測定し，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，ライブラリ内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のシステムコール発行まで連続でユーザプログラムを実行したと仮定し，この時間を測定する．測定した時間を基に，利用者の指定する要求性能から制御量を算出する．算出した制御量を基に，ライブラリがシステムコールを発行する直前にプロセスを停止させる．

また，プロセスの停止には，nanosleep システムコールを使用する．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することができる．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，最小停止時間が 1 ナノ秒と短いため，調整精度の向上が見込まれる．

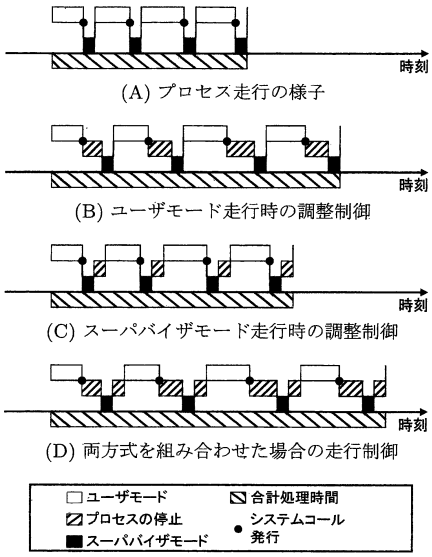


図 1 走行制御の様子

[スーパバイザモード走行時の調整]

スーパバイザモード走行時は，OS がシステムコール処理を実行する．そこで，スーパバイザモード走行時の調整 (以降，カーネル調整と呼ぶ) の基本的な方式として，OS がプロセスに依頼されたシステムコール処理を実行する時間を測定し，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，ユーザ調整と同様に，ライブラリ内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開始からシステムコール処理終了までの時間を測定する．測定した時間を基に，利用者の指定する要求性能から制御量を算出する．算出した制御量を基に，システムコール処理終了直後にプロセスを停止させる．

また，プロセスの停止には，ユーザ調整と同様に，nanosleep システムコールを使用する．

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，

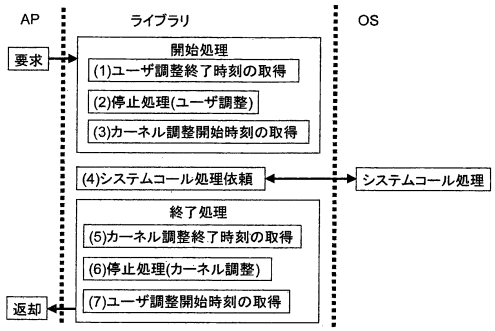


図 2 制御処理の流れ

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 制御処理の流れ

プロセスの走行モードを意識したプログラム実行速度の調整制御の流れを図 2 に示し，以下に説明する．

(1) ユーザ調整のために，プロセッサのシステムカウンタを用いて現システムカウンタ値 (以降，ユーザ調整終了時刻と呼ぶ) を取得する．

(2) 共有メモリから，ユーザ調整のために保存したユーザ調整開始時刻 (後述する) と要求性能を取得し，停止時間を算出する．また，算出した停止時間を基に，停止処理を行う．(詳細は 3.2 節で述べる)

(3) カーネル調整のために，プロセッサのシステムカウンタを用いて現システムカウンタ値 (以降，カーネル調整開始時刻と呼ぶ) を取得し，共有メモリに保存する．

(4) AP に依頼されたシステムコール処理を呼び出す．

(5) カーネル調整のために，プロセッサのシステムカウンタを用いて現システムカウンタ値 (以降，カーネル調整終了時刻と呼ぶ) を取得する．

(6) 共有メモリから，カーネル調整のために保存したカーネル調整開始時刻と要求性能を取得し，停止時間を算出する．また，算出した停止時間を基に，停止処理を行う．(詳細は 3.2 節で述べる)

(7) ユーザ調整のために，プロセッサのシステムカウンタを用いて現システムカウンタ値 (以降，ユーザ調整開始時刻と呼ぶ) を取得し，共有メモリに保存する．

3. 実装と評価

3.1 実装内容

プロセスの走行モードを考慮したプログラム実行速度調整法を FreeBSD 4.3-RELEASE(以降，FreeBSD と略す) のライブラリに実現した．具体的には，FreeBSD の既存ライブラリに大きく以下の 4 つの処理を追加した．

(1) 時刻の獲得

これは，rdtsc 命令でシステムクロックを取得することで実現した．

(2) 共有メモリのアタッチとデタッチ

アタッチは，ftok 関数，shmget システムコール，および shmat システムコールを利用した．共有メモリへのアタッチが失敗した場合，つまり，共有メモリが存在しない場合，停止処理を行わず，AP で依頼されたシステムコール処理を OS に依頼する．また，デタッチは，shmdt システムコールを利用した．

(3) 停止時間の算出

取得したシステムクロックと要求性能を用いて，停止時間を計算するプログラムを追加した．要求性能が 100%の場合，停止処理の算出と停止処理は行わず AP で依頼されたシステムコール処理を OS に依頼する．

(4) 停止処理 (nanosleep 処理の強制呼び出し)

ソフトウェア割込 (int0x80 命令) を発行し，nanosleep システムコールを発行する処理を追加した．また，nanosleep システムコール処理を依頼する直前の処理は，3.2 節で述べる．

3.2 停止処理の流れ

nanosleep システムコールを利用しプロセスを停止させる場合，以下の 2 つの問題がある．

(1) nanosleep システムコールの最小停止時間

nanosleep システムコールを利用したプロセスの停止は，算出した停止時間 (s) が最小停止時間 (W) より小さい場合，制御ができない．このため，nanosleep システムコールの最小停止時間 (W) を考慮し，閾値 (L) と合計停止時間 (S) を設け，制御を行う．ただし，閾値 (L) は最小停止時間 (W) よりも大きいことが必要である．

(2) nanosleep システムコールの精度

nanosleep システムコールの精度の問題により，閾値 (L) を 10 ミリ秒以下とした場合，十分な速度調整を行うことができない．このため，最小停止時間 (W)

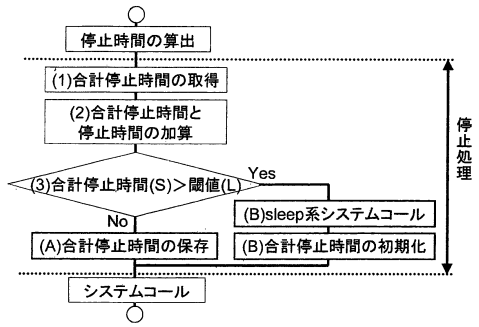


図 3 停止処理の流れ

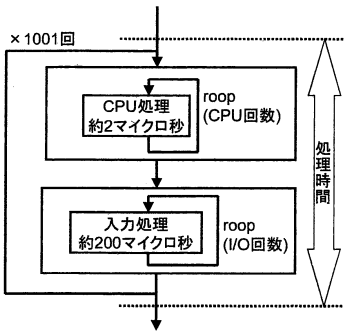


図 4 評価プログラム

を 10 ミリ秒とし，合計停止時間 (S)> 閾値 (L) となる場合は，合計停止時間 (S) から 10 ミリ秒減算し，停止処理を行う．

上記の問題を考慮した停止処理の流れを図 3 に示す．これにより，停止時間が小さい場合も制御ができ，nanosleep システムコールの精度による影響を抑制することができる．

3.3 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使っ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 4 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．

測定は，rdtsc 命令でシステムクロックを取得し行った．

3.4 評価の観点

試作したライブラリ (以降，試作ライブラリと略す) を使用してプロセスを走行させるうえで，追加した処理が与える影響は小さく，調整の精度が高いことが望ましい．そこで，試作ライブラリのオーバヘッドを測定した．試作ライブラリでは，実行途中で要求性能を変更可能にするため，要求性能の指定法として共有メモリを利用する．共有メモリへのアタッチ，およびデタッチは，4 つの関数を発行して行うため，オーバヘッドは大きくなると考えられる．また，閾値 (L) による調整の精度の影響を評価した．

次に，さまざまな AP の実行速度を調整する場合を考慮し，CPU 処理と入力処理の比率を変化させ，走行モードを意識した速度調整について評価を行った．具体的には，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，ユーザ調整，カーネル調整，および両モードの調整 (以降，両調整と略す) を行い，調整の精度を評価した．入力処理時間と比率を変化させた場合の CPU 処理時間を表 1 に示す．

例えば，入力処理時間を 20 ミリ秒 (I/O 回数は 100 回) とし，CPU 処理:I/O 処理を 1:2 として評価を行う場合，CPU 処理時間が 10 ミリ秒となるように CPU 回数を変化させる．評価は，ユーザ調整を行う場合，カーネル調整を行う場合，および両調整を行う場合について行った．

なお，特に説明のない限り，以降の各図の相対処理時間は，被調整プログラムに対し各要求性能で両調整を行った場合の理論値を 1 とした場合の処理時間である．理論値とは，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．相対処理時間が 1 より大きい場合，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，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．

3.5 評価考察

3.5.1 オーバヘッド

試作ライブラリに追加した処理がプロセスに与える影響を明らかにするため，試作ライブラリのオーバヘッドを評価した．具体的には，既存ライブラリと試作ライブラリにおいて，read システムコール 1 回に

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	1:2	1:4
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

表 2　オーバヘッド

既存ライブラリ	試作ライブラリ (A)	試作ライブラリ (B)
203.6[マイクロ秒]	251.6[マイクロ秒]	266.3[マイクロ秒]

かかる時間の測定を 10 回行い，その平均値を評価した．ただし，試作ライブラリは，共有メモリがある場合とない場合で処理が異なる．このため，共有メモリがない場合を試作ライブラリ (A)，ある場合を試作ライブラリ (B) とし，それぞれについて評価した．また，要求性能は 100%とした．評価結果を表 2 に示す．

表 2 より，read システムコール 1 回にかかるオーバヘッドは，共有メモリがない場合は約 25%(48 マイクロ秒)，ある場合は 30%(62.7 マイクロ秒) であり，既存ライブラリと比べて大きいといえる．しかし，nanosleep システムコールを用いてプロセスを停止させる場合，停止時間は 10 ミリ秒以上である必要がある．このため，実行速度を調整する場合，追加処理のオーバヘッドは無視できる．

3.5.2　閾値による影響

実行速度の調整を行う際，利用者の希望する速度に近い速度でプログラムが実行されることが望ましい．文献 3) では，ユーザ調整を行う場合における閾値による影響を評価し，閾値を 10 ミリ秒にした場合にもっとも精度がよいことを明らかにした．そこで，本稿では，カーネル調整を行う場合における閾値による影響について評価を行った．

評価プログラムにおいて，入力処理時間を 2 ミリ秒とし，CPU 処理と入力処理の割合を変化させた．また，閾値が，10 ミリ秒，30 ミリ秒，および 50 ミリ秒の場合について，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 5 に示す．図 5(A) は CPU 処理時間が 0 である．図 5(B) は入力処理の割合が高く，図 5(F) は CPU 処理の割合が高い．なお，本項の相対処理時間は，被調整プログラムに対し各要求性能でカーネル調整を行った場合の理論値を 1 とした場合の処理時間である．

図 5 より，以下のことがわかる．

(1) 閾値 (L) を小さくした場合 (L=10 ミリ秒) は，CPU 処理と入力処理の比率に関わらず，調整の精度が悪いことがわかる．これは，nanosleep システムコールの精度に起因する．具体的には，算出した停止時間に対し，実際の停止時間が大きくなる．また，閾値を小さくした場合の停止処理は，閾値を大きくした場合に比べ，頻繁に発生する．このため，算出した停止時間と実際の停止時間の差の総和が大きくなり，調整の精度が悪くなる．

(2)CPU 処理の割合が大きい場合 (図 5(F)) は，CPU 処理の割合が小さい場合 (図 5(B)) に比べ，調整の精度がよいことがわかる．これは，CPU 処理の割合が大きくなると停止時間のずれは相対的に小さくなるためである．つまり，閾値の値は，入力処理の割合が高い場合に精度がよい値とすればよい．

(3) 入力処理のみを行う場合 (図 5(A)) は，閾値を 50 ミリ秒とした場合にもっとも精度がよいことがわかる．一方，閾値を大きくする場合，閾値を小さくする場合に比べプロセス停止処理間の時間が長くなるため，きめの細かい調整が行えない．

以上のことから，カーネル調整は閾値を 50 ミリ秒として実行速度の調整を行う．

3.5.3　走行モードを意識した実行速度の調整

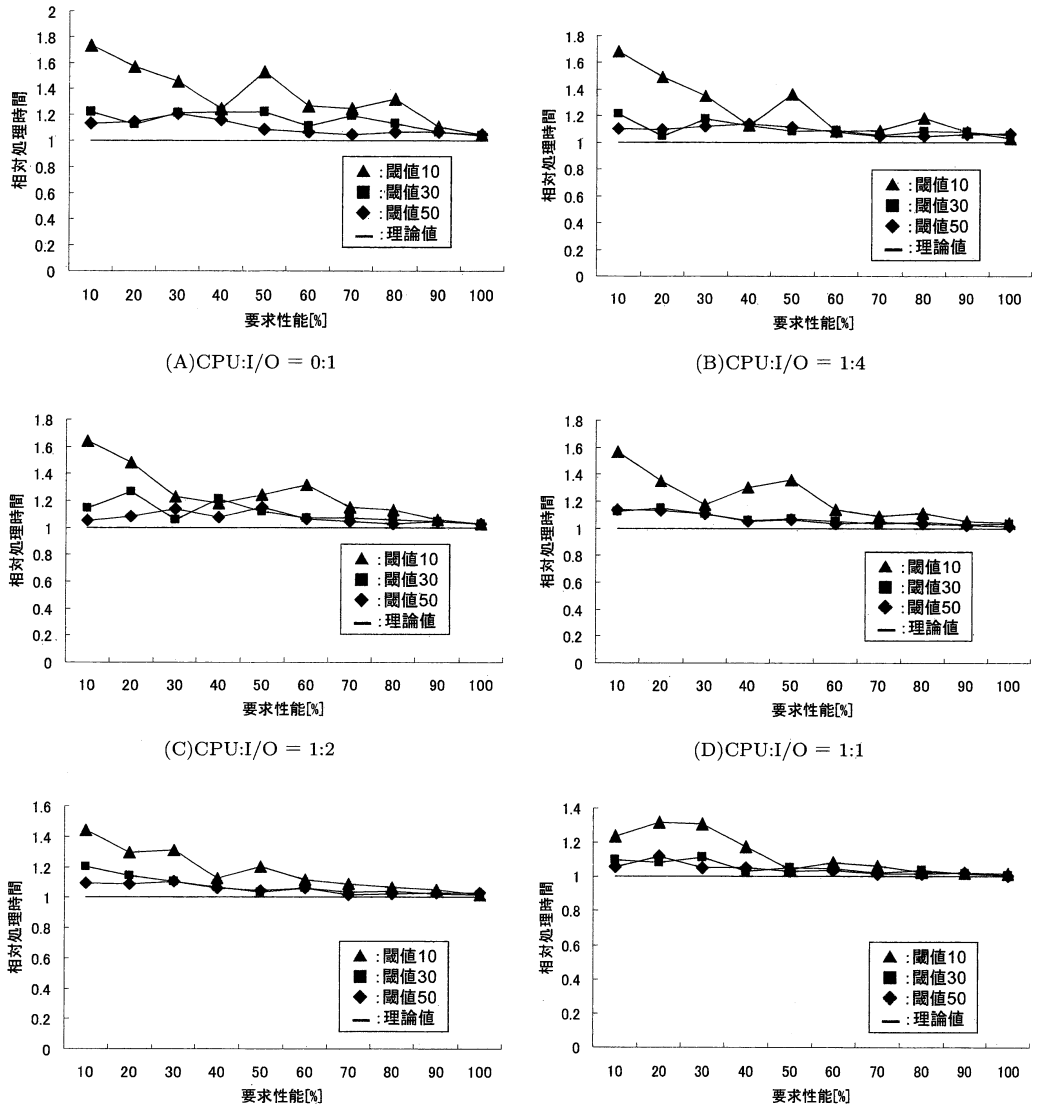
計算機上で走行するプロセスには，さまざまなものが存在し，走行モードも変化する．例えば，read システムコールを多く発行するプロセスは，read システムコールをほとんど発行しないプロセスに比べ，スーパバイザモードで走行する割合が高い．そこで，走行モードの違いによる調整の精度への影響を明らかにするため，CPU 処理と入力処理の割合を変化させ評価を行った．

評価プログラムにおいて，入力処理時間を 200 ミリ秒とし，CPU 処理と入力処理の割合を変化させた．また，要求性能を 10%から 100%まで 10%刻みで変化させ，ユーザ調整，カーネル調整，および両調整を行う場合について評価した．評価結果を図 6 に示す．

図 6 より，以下のことがわかる．

(1) ユーザ調整を行う場合，CPU 処理の割合が高い (図 6(F)) と理論値に近づく．一方，CPU 処理の割合が低い (図 6(A)) とうまく調整できていないことがわかる．これは，ユーザ調整はユーザモード走行時の調整を行うものであり，スーパバイザモード走行時の調整を行わないためである．

(2) カーネル調整を行う場合，入力処理のみを行う場合 (図 6(A)) は，うまく調整できている．また，入



(A)CPU:I/O = 0:1　(B)CPU:I/O = 1:4

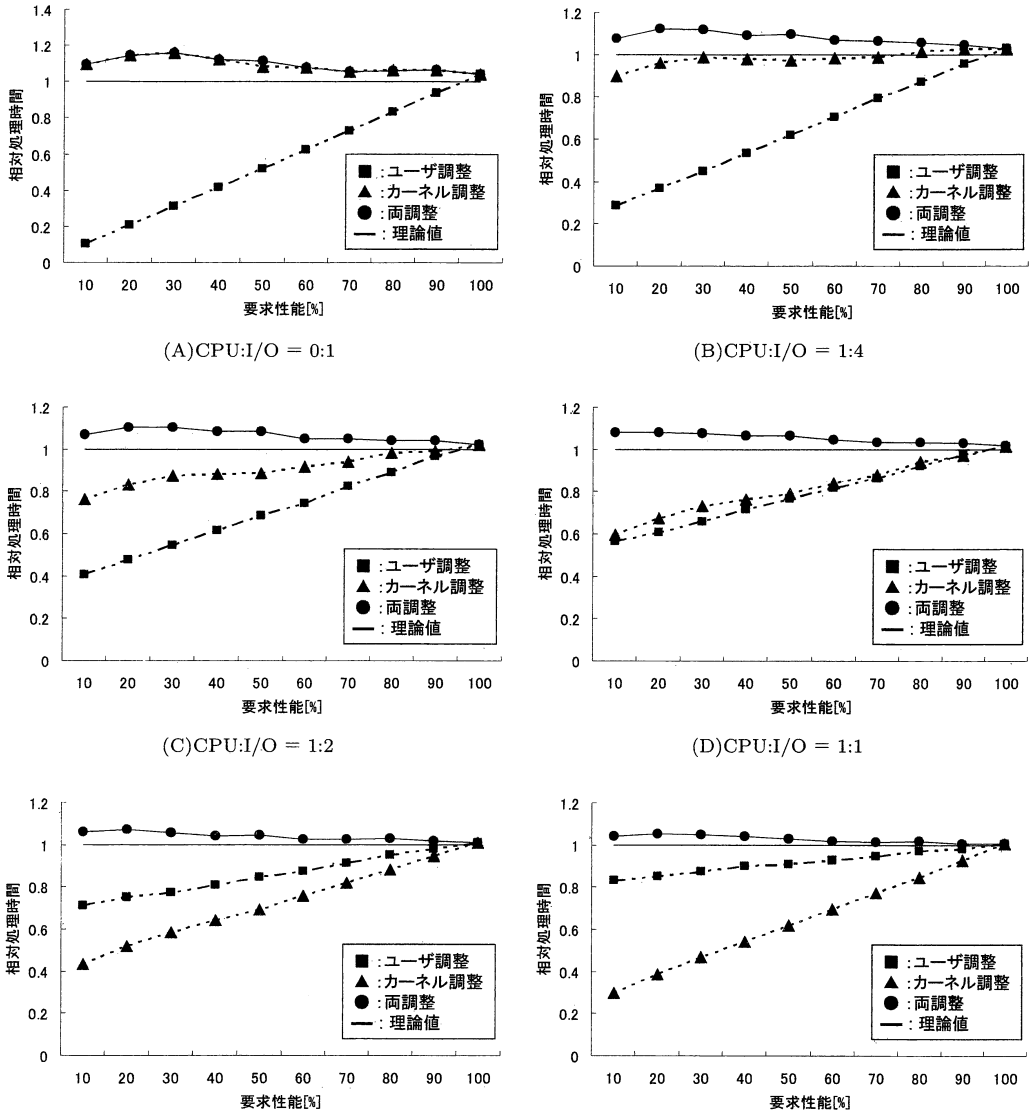
(C)CPU:I/O = 1:2　(D)CPU:I/O = 1:1

(E)CPU:I/O = 2:1　(F)CPU:I/O = 4:1

図 5　閾値による影響

力処理の割合が高い (図 6(B)) 場合，理論値に近づく．一方，入力処理の割合が低い (図 6(F)) と調整できていないことがわかる．これは，ユーザ調整のみを行う場合と逆の理由による．つまり，カーネル調整は，ユーザモード走行時の調整を行わないためである．

(3) 両調整を行う場合，CPU 処理と入力処理の割合に関わらず，うまく調整を行えていることがわかる．特に，CPU 処理の割合が高い場合 (図 6(F)) は，CPU 処理の割合が低い場合 (図 6(A)) に比べ，うまく調整を行うことができる．具体的には，CPU 処理の割合が低い場合の相対処理時間は最大で約 1.2 であるのに対し，CPU 処理の割合が高い場合の相対処理時間が約 1.05 である．この違いは，nanosleep システムコールの発行回数と精度に起因する．



(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 6　走行モードの違いによる影響

以上のことから，両調整はユーザ調整のみやカーネル調整のみを行う場合ではうまく調整できないプログラムに対しても精度のよい調整を行えるといえる．特に，CPU 処理の割合が高いプログラムに対する調整の精度がよい．

4. 関連研究

従来，プロセス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．これらに対し，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，実行速度を調整する．

5. おわりに

ライブラリによるプログラムの走行モードを考慮した実行速度調整の実現方式ついて述べた．具体的には，プログラムを実行する場合，そのプログラムを実行するプロセスは，2つのモードで走行することに着目した．ユーザモード走行時の調整は，システムコール処理終了から次のシステムコール発行までの時間と利用者の指定する要求性能から算出した制御量を基に，システムコール発行直前にプロセスを停止させる．一方，スーパバイザモード走行時の調整は，システムコール処理開始からシステムコール処理終了までの時間と利用者の指定する要求性能から算出した制御量を基に，システムコール処理終了直後にプロセスを停止させる．

要求性能の指定法として，利用者が要求性能を任意に決定できるように共有メモリを利用した．また，nanosleepシステムコールの精度による影響を避けるため，文献3)の方式(閾値(L)を設け，合計停止時間がLより大きい場合のみプロセスを停止する)を採用しカーネル調整にも適用した．

さらに，ライブラリとして実装し，評価した．実行速度を調整する際，プログラムを10ミリ秒以上停止させるため，既存のライブラリに追加した処理の影響は小さい．また，カーネル調整を行う場合，閾値(L)が50ミリ秒の場合に調整の精度がもっともよいことを示した．さらに，提案した制御法は，プログラムの走行モードの走行割合に関わらず，うまく調整できることを示した．特に，CPU処理の割合が高い場合は，CPU処理の割合が低い場合に比べて調整の精度が高い．これは，nanosleepシステムコールの発行回数と精度に起因する．

残された課題として，カーネル調整を行う場合の精度の向上がある．

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) 谷口秀夫，“入出力時間の制御によりプログラム実行速度を調整する制御法，”電子情報通信学会論文誌(D)，vol.J83-D-I，no.5，pp.469-477，(2000.04)．
2) 田端利宏，谷口秀夫，“複数サービスの処理時間を調整するプロセススケジュール法，”電子情報通信学会論文誌(D)，vol.J86-D-I，no.7，pp.458-468，(2003.09)．
3) 境　講一，田端利宏，谷口秀夫，箱守　聰，“システムコール発行間隔の制御によるプロセッサ使用量の調整，”情報処理学会研究報告，2008-OS-109，vol.2008，no.77，pp.101-108，(2008.08)
4) Anna Povzner，Tim Kaldewey，Scott Brandt，Richard Golding，Theodore M. Wong，Carlos Maltzahn，“Efficient Guaranteed Disk Request Scheduling with Fahrrad，”EuroSys'08，pp.13-25，(2008.04)
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，(2001.06)．