WWWサーバにおけるサービスの処理内容を考慮したプロセススケジュール法

スカンヤ スラナワラッ† 谷口秀夫†

既存の多くのオペレーティングシステムが実現しているプロセスのスケジューラは，プロセスの動作内容に関係なく，実時間や時分割の処理といったプロセス実行の以前に得られる利用形態を基に，プロセスの実行を制御している．このため，プロセッサの有効利用を妨げ，プロセスの処理時間を無用に引き延ばしている場合がある．そこで，本論文では，複数のプロセスで構成されるサービスについて，プロセスの動作内容を予測して，その実行を制御する方式について述べる．具体的には，サービスとして，WWWサーバを取り上げ，サーバプロセスの動作内容を予測し，その予測動作に合わせて，クライアント・サーバによるHTTPトランザクションの応答時間を短くするスケジュール法を提案する．

Process Scheduling Policy for a WWW Server Based on Its Contents

SUKANYA SURANAUWARAT† and HIDEO TANIGUCHI†

Process schedulers of many existing operating systems, control process execution based on predetermined policies such as real-time or time sharing system, not based on a process behaviour. Hence, this can hinder an effective use of a processor or, extend the processing time of a process unnecessarily. In this paper, we propose a new process scheduling policy for improving WWW server response time. This policy controls multiple processes of WWW server by adjusting the execution of these processes according to their predicted behaviour.

1. はじめに

既存の多くのオペレーティングシステムが実現しているプロセスのスケジューラは，プロセス実行の以前に得られる利用形態を基に，プロセスの実行を制御している．たとえば，実時間の処理を行う利用形態では，実行優先度により，実時間性の高い処理を優先的に実行している．また，多くの利用者が利用する計算機では，処理の均等化を図るため，タイムスライスによりプロセスの連続実行時間を制限している．処理の均等化を重視したスケジューラは，動画や音声などのデータの時間的な制約を満たさせることができない．このため，処理の時間的な制約を保証することを目的とした実時間処理のスケジュール法に関する研究はさかんに行われている[1)~3)]．

いずれの場合も，「プロセスの動作内容」に合わせて実行制御を行っているわけではない．このため，プロセッサの有効利用を妨げ，また，プロセスの処理時間を無用に引き延ばしている場合がある．具体的な例としては，残りわずかでプロセッサ処理を終えようとするプロセスでも，タイムスライス機能によって，強制的に再スケジュールが行われる場合があげられる．この例では，そのプロセスの処理が次のプロセッサ割当てまで待たされることとなり，プロセスの処理時間の増加をもたらす．また，強制的な再スケジュールは，プロセス切替処理の増加も招く．もし，このとき，強制的な再スケジュールを行わなければ，プロセスの継続実行によりプロセスの処理時間が短縮され，プロセス切替処理も減少させることができる．つまり，「残りわずかでプロセッサ処理を終える」というプロセスの動作を予測し，プロセスのプロセッサ利用に合わせたスケジュールを行えば，上述のような無駄は発生しない．

そこで，プログラム動作内容に合わせてプロセスの実行制御法を変更するという考え方である，プログラム指向スケジュール[4)]（POS：Program Oriented Schedule）を提案した．

POSの基本的な考え方を以下に示す．

（1）プロセスの動作を把握し，プログラム動作内

† 九州大学大学院システム情報科学研究科
Graduate School of Information Science and Electrical Engineering, Kyushu University

容の知識として保存する．

（2）プログラム動作内容の知識を利用して，プロセスの実行制御法を変更する．

POSと似たような考え方として,「アプリケーションの起動を高速化する」というWindows98の機能があげられる．これは，ユーザの使用状況に合わせてディスクを最適化することによって，アプリケーションの起動を高速化する．具体的には，ユーザがどのアプリケーションをどの程度の頻度で利用しているか，各アプリケーションが利用するファイルとその利用頻度をOSが自動的に記録する．それを基にディスクの配置を最適化することで，次にそのアプリケーションを起動する際はより高速に起動できる仕組みである[5]．

これまでに，POSの考え方を用いた研究を文献4)と6)に報告している．文献4)ではプロセスの動作を予測してプロセスの切替時期を変更する方式を提案している．また，文献6)ではプログラムの動作を予測して入力処理を効率化する方式を提案している．前者の方式は，演算スケジュールへ適用したものであり，サービスが1プロセスで構成されている場合に有効である．しかし，複数のプロセスで構成されることが多い今日のサービスへの適用には不十分である．このため，複数のプロセスで構成されるサービスに適用できるスケジュール法を確立する必要がある．

複数のプロセスで構成されるサービスは，大きく2種類に分けられる．1つは，プロセスが互いに関係なく単独で処理を行うサービスであり，もう1つは，プロセスが互いに協調しながら処理を行うサービスである．最近，急増している分散システムでは，大半のサービスがクライアント・サーバ方式で実現されており，多くのサーバのサービスは前者である．サーバでは，同じプログラムによるプロセスが同時に多数，生成と消滅を繰り返して動作を行う．サーバの処理は同じ内容の繰返し，つまり，トランザクション処理の一種である．また，サーバの性能が高いことが要求されている．そこで，本論文では，サーバの処理の効率向上を重点におき，現在よく使われているWWWサーバを対象サービスとして取り上げ，サーバプロセスの動作内容を把握し，その予測動作に合わせて，クライアント・サーバによるHTTPトランザクションの応答時間を短くするスケジュール法を提案する．

関連研究として，文献8)では，複数のプロセスからなるアプリケーションプログラムを1つのプロセスグループにまとめることで，個々のアプリケーションプログラムが1つのまとまった優先度を持ち，アプリケーションプログラムごとのスケジュールを行っている．文献8)では，複数のプロセスからなるアプリケーションプログラムを1つにまとめたプロセスグループをスケジューリング対象とするのに対し，本論文では，サービスを構成する個々のプロセスをスケジューリング対象としている．

2. WWWサーバ

本論文では，無料で入手し利用することが可能であり，インターネット上で約5割のシェアを占めているApacheをWWWサーバとして使用した．

以降，POSの基本制御機構[4]によるApacheサーバの動作内容を理解するのに必要な知識として，Apacheサーバの動作[8),9)]や，クライアント・サーバによるHTTPトランザクション構造[10]について説明する．

2.1 Apacheサーバの動作

Apacheはスタンドアロンまたはinetdモードのどちらかのモードで稼働する．Apacheの機能のいくつかはinetdモードで動作しないので，このモードでのApacheの使用は非常に単純な場合に限られる．したがって，一般に，スタンドアロンモードは通常運用，inetdモードは実験運用で各々利用される．以降，Apacheがスタンドアロンモードで稼働している場合の動作について説明する．

Apacheを稼働させると，1つのプロセスが生成される．このプロセスは，初期化の処理などを行った後，新たなプロセス（以降,「親サーバ」と呼ぶ）を生成して終了する．

親サーバはソケット処理などを行い，Configファイルに指定された起動時のサーバプロセスの数（以降,「StartServers」と略す）で新たなプロセス（以降,「子サーバ」と呼ぶ）を生成し，これらの子サーバの状態を監視する．具体的には，接続持ち状態の子サーバの数がConfigファイルに指定された接続待ち状態の子サーバの最小数（以降,「MinSpareServers」と略す）よりも少なくなった場合は，親サーバが新しい子サーバを1秒に1つ新たに生成する．

実際に，クライアントからの接続要求の処理を行うのは，子サーバである．子サーバは，Configファイルに指定された接続待ち状態の子サーバの最大数（以降,「MaxSpareServers」と略す）や，各子サーバが処理できる要求の最大数（以降,「MaxRequestsPerChild」と略す）に達すると終了する．

2.2 HTTPトランザクションの構造

HTTPトランザクションの構造を**図1**に示し，ブラウザを用いてWWWサーバにアクセスし，ドキュメントを要求する場合を例にその働きを説明する．な

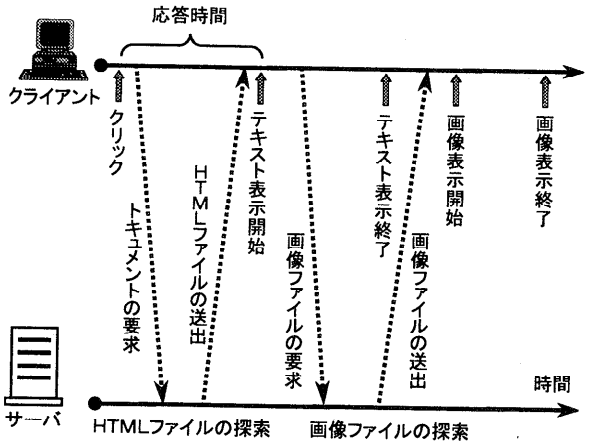

図1 HTTPトランザクションの構造
Fig. 1 A structure of HTTP transaction.

お，サーバが提供しているドキュメントには，1つの画像が含まれるとする．

まず，ブラウザはユーザから入力されたURLを読み込んで解釈し，該当するサーバマシンへ接続してドキュメントを要求する．次に，サーバマシン上でクライアントからの接続を待機している，複数の子サーバの中の1つがこの要求を受け付け，URLに指定されたドキュメントパスに対応するHTMLファイルを探して，要求元のブラウザに返す．最後に，ブラウザは，サーバから取得したHTMLファイルを解釈しながら必要な整形を行い，画面に表示する．また，ブラウザがこのHTMLファイルを解釈し，このドキュメントの表示には画像ファイルが必要なことが判明したら，ブラウザはWWWサーバに画像ファイルの要求を送信する．その後，HTMLファイルの要求の場合と同様な処理を行う．

2.3 サーバの動作内容

本節では，Apacheサーバの動作内容を把握し，分析した結果を報告する．

サーバの動作内容を把握するために，各子サーバのプロセスの状態とその時刻を記録した．そして，それぞれの子サーバの識別子とその状態の列からなる情報であるPFS[4]（Program Flow Sequence）を作成した．以降，PFSを基にして分析したサーバの動作内容について説明する．

図2には，ユーザが同一ブラウザに同一URLを3回入力したときの動作内容を示す．サーバは，Apache Ver.1.2.5を，ブラウザは，Netscape Navigator Ver.3.04（以降，「Netscape」と略す）を使用した．サーバのConfigファイルには，StartServersを1，MinSpareServersを1，MaxSpareServersを3，MaxRequestsPerChildを3と設定した．また，サーバが提供しているドキュメントには，1つの画像が含まれている．

図2において，親サーバは，StartServersの値と同じ数の子サーバ（プロセスA）を生成してから，次のプロセスの状態を監視するまでの間（1秒）sleepする．生成されたプロセスAは，初期化などの処理を行った後，ブラウザからの接続を待機してWAIT状態となる．一方，ブラウザはユーザに入力された1回目のURLを読み込んで解釈し，該当するマシンへ接続して要求を送信する．その後，接続を待機していたプロセスAはRUN状態となり，URLに指定されたドキュメントパスに対応するHTMLファイルを探して，要求元のブラウザに返す．そして，ブラウザからの再接続時の遅延を最小限とするための最大要求待ち時間（以降，「KeepAliveTimeout」と略す）で，接続を保持した状態でWAIT状態となる．ただし，KeepAliveTimeoutは7秒と設定した．また，このとき，親サーバは，再び子サーバの状態を調べ，MinSpareServersの値より子サーバの数が少なくなったことが分かり，新たな子サーバ（プロセスB）を生成して，再びsleep（1秒）する．

一方，ブラウザは，サーバからのHTMLファイルを解釈し，このドキュメントの表示には画像ファイルが必要であると判明し，2つ目の要求を送信する．そして，生成されたプロセスBは，プロセスAと同様の処理を行い，WAIT状態になる直前に，2つ目の要求が来たので，画像ファイルを探し，要求元のブラウザに返す．そして，プロセスAと同様にKeepAliveTimeout

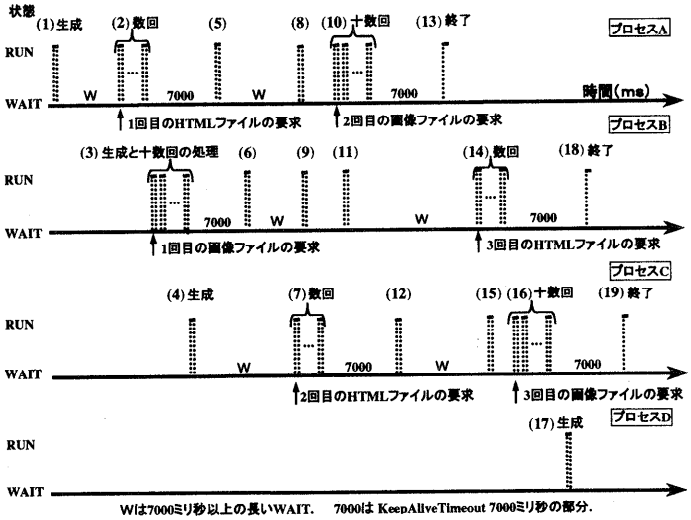

図2 PFSによるサーバの動作内容
Fig. 2 Activation flow of processes according to PFS.

により WAIT 状態となる．また，このとき，親サーバは，再び子サーバの状態を調べ，先ほどと同様に新たな子サーバ（プロセス C）を生成し，再び sleep（1 秒）する．

プロセス A，プロセス B は，KeepAliveTimeout を経過したら，順次 WAIT 状態から RUN 状態へ移行し，2.1 節で述べた終了条件である MaxSpareServers と MaxRequestsPerChild を調べる．しかし，このときには，終了条件を満足していないので，再び接続を待機して WAIT 状態となる．

次に，2 回目の URL が入力されると，ブラウザが同様な処理を行った後，先に接続を待機して WAIT 状態となったプロセス C が RUN 状態となり，HTML ファイルを探して要求元のブラウザに返す．そして，KeepAliveTimeout で WAIT 状態となる．その後，プロセス C の次に接続を待機していたプロセス A は，2 回目 URL 入力についての 2 つ目の要求に対して，画像ファイルを探し要求元のブラウザに返す．そして，KeepAliveTimeout で WAIT 状態となる．その後，プロセス C もプロセス A も，順に KeepAliveTimeout による WAIT 状態から RUN 状態へ移行して終了条件を調べる．このとき，プロセス C は，終了条件を満足していないので，再び接続を待機して WAIT 状態となる．一方，プロセス A は，終了条件である MaxRequestPerChild に達したので，終了する．

最後に，3 回目の URL が入力されると，上述の処理と同様に，プロセス B は HTML ファイルを，プロセス C は画像ファイルを探して，要求元のブラウザに返す．そして，KeepAliveTimeout で WAIT 状態となる．このとき，親サーバは，再び子サーバの状態を調べ，MinSpareServers の値より子サーバの数が少なくなったことが分かり，新たな子サーバ（プロセス D）を生成し，再び sleep（1 秒）する．生成されたプロセス D は，初期化などの処理を行った後，ブラウザからの接続を待機して WAIT 状態となる．

3. スケジュール法

本章では，クライアント・サーバによるトランザクションの応答時間をどのように短縮することができるのかを示し，新しいスケジュール法を提案する．

3.1 応答時間を短縮する考え方

図 1 において，ドキュメントの要求に関するトランザクションは，HTML ファイルの要求と画像ファイルの要求に関する 2 つのトランザクションからなる．ユーザにとっては，クリックしてからテキストを表示するまでの時間，すなわち，HTML ファイルの要求に関するトランザクションの応答時間が短いことが望ましいと考えられる．この応答時間を短くするには，HTML ファイルの要求を処理する子サーバの処理時間を短縮することによって実現できると考えられる．以降，HTML ファイルの要求を処理する子サーバの状態の変化を図 **3** に示し，いかに処理時間を短縮するかを示す．図 3 に示したように，子サーバは，RUN 状態，WAIT 状態および READY 状態を繰り返しながら 1 つのトランザクションを処理する．この一連の処理を行ううえで，基本的に削減できない時間は，プロ

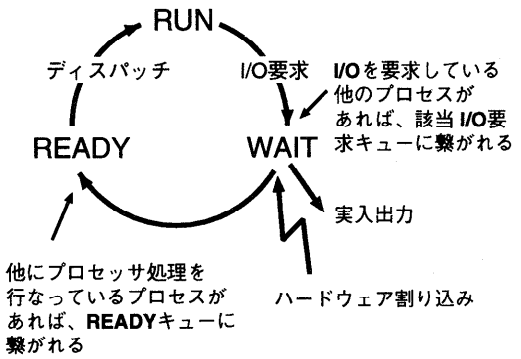

図3 子サーバの状態の変化
Fig. 3 The state transition of a server's process.

セッサ処理時間（プロセス状態はRUN状態）とDISK入出力やデータ通信のための実入出力を行う処理時間（プロセス状態はWAIT状態）である．しかし，削減できる時間として，OSのプロセス制御により発生する待ち時間（ハードウェアを占有できれば発生しない時間）がある．この時間は大きく3つある．READYキューでの待ち時間，DISK入出力待ちキューでの待ち時間，およびデータ通信待ちキューでの待ち時間である．したがって，HTMLファイルの要求を処理する子サーバがそれぞれのキューに置かれているとき，キューに繋がれている時間を短くすることにより，処理時間を短縮することができる．

3.2 スケジュール法

3.1節の考え方を基にして，以下の3つのスケジュール法を示す．

(1) READYキューでの優先制御
プロセッサがボトルネックになった場合には，READYキューに繋がっているプロセスの中で，HTMLファイルの要求を処理するプロセスを優先してキューを繋ぎ換える．

(2) DISK入出力待ちキューでの優先制御
ディスクがボトルネックになった場合には，DISK入出力待ちキューに繋がれているプロセスの中で，HTMLファイルの要求を処理するプロセスを優先してキューを繋ぎ換える．

(3) データ通信待ちキューでの優先制御
データ通信がボトルネックになった場合には，データ通信待ちキューに繋がれているプロセスの中で，HTMLファイルの要求を処理するプロセスを優先してキューを繋ぎ換える．

これらのスケジュール法により，テキスト表示の開始および終了を速くできる．

3.3 実現時の課題と対処

本章では，READYキューでの優先制御のスケジュール法（以降，「スケジュール法1」と略す）について，実現時の課題と対処を述べる．課題として，
〈課題1〉HTMLファイルの要求を処理するプロセスの検出方式
〈課題2〉プロセスを優先制御するためのREADYキューでの処理方式
があり，対処が必要である．
〈課題1への対処〉図2により，HTMLファイルの要求を処理するプロセスは以下の特徴を持つ．
（特徴1）長いWAIT状態の後に走行する．
（特徴2）RUN状態とWAIT状態を数回繰り返す．

そこで，プロセスがこれらの特徴を持つかどうかを判断するために，以下の判断基準を導入する．
（1）長いWAIT状態を判断するためのしきい値SLP
（2）RUN状態とWAIT状態が数回であることを判断するためのしきい値RW

RUN状態の直前のWAIT状態の時間がSLPより長いとき，そのプロセスは長いWAIT状態の後に走行（特徴1）したと判断する．次に，(特徴1）を有するプロセスについて，RUN状態とWAIT状態の繰返し回数がRWより小さいとき，そのプロセスはRUN状態とWAIT状態を数回繰り返した（特徴2）と判断する．これにより，HTMLファイルの要求を処理するプロセスを検出する．
〈課題2への対処〉プロセスを優先制御するためのREADYキューでの処理方式として，HTMLファイルの要求を処理するプロセスをREADYキューの先頭に繋ぐことにする．具体的には，(特徴1）を有するプロセスはREADYキューの先頭に繋ぎ，(特徴2）を失った時点でREADYキューの末尾に繋ぎ換える．

なお，SLPとRWは，2.3節で述べたPFSを基に，SLPとRWを予測して設定する．SLPとRWの予測手法を以降に説明する．
〈SLPの予測について〉
HTMLファイルの要求を処理するプロセスは，クライアントからの接続を待機しているプロセスである．接続を待機する時間が比較的長い．したがって，単位時間においてPFSを基に各子サーバごとに最も長いWAIT状態の時間を求め，次に，すべての子サーバの最長WAIT状態時間のうちの最小時間をSLPと設定する．
〈RWの予測について〉
HTMLファイルや画像ファイルの要求を処理するプロセスのファイルの入力処理によるRUN状態とWAIT

表1 対象プロセスの管理表
Table 1 A management table of processes with regard to our schedule policy.

pid	status	flag	rw_cnt	wait_cnt
⋮				
⋮				

状態の繰返し回数は，ファイルの大きさに比例する．通常，画像ファイルがHTMLファイルより大きいので，図2に示したように，画像ファイルの要求を処理するプロセスのRUN状態とWAIT状態の繰返し回数は，十数回である．一方，HTMLファイルの要求を処理するプロセスのRUN状態とWAIT状態の繰返し回数は数回である．このことを利用して，単位時間においてPFSを基に各子サーバごとに最小のRUN状態とWAIT状態の繰返し回数を求め，次に，すべての子サーバの最小繰返し回数のうちの最大回数をRWと設定する．

4. 実現方式

子サーバを制御するため，**表1**に示す管理表を用意した．スケジュール法1を実現するアルゴリズムを以下に説明する．

(1) プロセスが生成されたときに，スケジュール法1の対象プロセスであるために，そのプロセスを表1に示す管理表に登録する．表1における1要素は，プロセスの識別子（pid），プロセスの状態（status），READYキューの先頭または末尾に繋ぐかを示すフラグ（flag），RUN状態とWAIT状態の繰返し回数を記録するカウンタ（rw_cnt），およびWAIT状態の連続時間を記録するカウンタ（wait_cnt）の5項目からなる．なお，プロセスの状態とは，そのプロセスが動作中か消滅したかの情報である．

(2) 対象プロセスがWAIT状態になると，wait_cntを1秒ごとにカウントアップする．

(3) 対象プロセスがWAIT状態からREADY状態へ移行する際，

(A) その対象プロセスがWAIT状態にいる時間が長い場合（wait_cnt >= SLP），そのプロセスはHTMLファイルの要求を処理するプロセスと判断し，READYキューの先頭に繋ぎ，flagをONにし，rw_cntを1とカウントアップする．そして，wait_cntをクリアする．

(B) その対象プロセスがWAIT状態にいる時間が短い場合（wait_cnt < SLP），

(a) そのプロセスが前回READYキューの先頭に繋がれたら，すなわち，flagがONであれば，wait_cntをクリアする．そして

(i) RUN状態とWAIT状態の繰返し回数が小さい場合（rw_cnt <= RW），今回も，そのプロセスをREADYキューの先頭に繋ぎ，rw_cntをカウントアップする．

(ii) RUN状態とWAIT状態の繰返し回数が大きい場合（rw_cnt > RW），今回は，そのプロセスをREADYキューの末尾に繋ぎ，flagをOFFにし，rw_cntをクリアする．

(b) そのプロセスが前回READYキューの末尾に繋がれたら，すなわち，flagがOFFであれば，wait_cntをクリアし，プロセスをREADYキューの末尾に繋ぐ．

3.3節に基づき，SLPとRWの設定アルゴリズムを以下に示す．

(1) 子サーバが生成されたときに，子サーバのPFSを作成するようにPFSを管理する表に登録する．

(2) 登録したすべての子サーバのPFSの作成の時間間隔を指定する．この時間間隔は，3.3節で述べたSLPやRWの予測手法における単位時間である．

(3) 指定した時間間隔で作成した各子サーバごとのPFSから最も長いWAIT状態の時間と最も短いWAIT状態の時間を求める．次に，すべての子サーバの最長WAIT状態時間のうちの最小時間をSLP，またすべての子サーバの最小WAIT状態時間のうちの最大時間をshortSLPと設定する．ただし，shortSLPは短いWAIT状態を判断するためのしきい値である．そして，各子サーバごとのPFSから最小のRUN状態とshortSLP以下のWAIT状態の繰返し回数を求め，すべての子サーバの最小繰返し回数のうちの最大回数をRWと設定する．

(4) 子サーバが終了するときに，子サーバのPFSの作成を止める．

5. 評　　価

スケジュール法1を評価するために，4章で述べたアルゴリズムBSD/UNIXに実現した．測定は，シングルユーザ環境で，OSが持つ入出力バッファのキャッシュ効果の影響を受けないように，入出力バッファ・キャッシュがヒットしない状態で行った．

評価は，測定用に作成した評価プログラムとNetscapeについて行った．それぞれについて以下に説明する．

5.1 評価プログラムの場合

評価プログラムの処理は，5回のreadシステムコールを繰り返し発行した後，SLP以上の時間でWAIT状態になる．その後，再び5回のreadシステムコールを繰り返し発行し終了する．使用した計算機は，プロセッサがAMD-K6 233 MHzである．無限ループのプロセッサ処理を行うプロセスと共存させ，処理時間を測定した．測定の結果を**図4**に示す．図4は，RWを1から5まで変化させて，スケジュール法1を使わないときの処理時間との差を示す．結果から，RWの増加とともに処理時間が短くなった．これは，次の要因による．評価プログラムを実行したプロセスが，SLP以上の時間でWAIT状態になってから，スケジュール法1の対象プロセスとなる．以降，readシステムコールによる入力処理が終了すると，プロセスが即座にRUN状態へディスパッチされる．このとき，タイムスライス間隔が100ミリ秒なので，処理時間は約100ミリ秒ずつ短くなったのである．

5.2 Netscapeの場合

実測に用いたサーバプログラムは，Apache Ver.1.2.5であり，2.3節と同じConfigファイルで設定した．使用した計算機は，Apacheがプロセッサ AMD-K6 233 MHz の計算機，Netscapeがプロセッサ Pentium Pro 180 MHz の計算機である．それぞれはBSD/UNIXを使用し，10 Mbpsのイーサネット型通信路で結ばれている．

処理の内容は，**表2**に示すパラメータの組合せで，Netscapeを用いて，サーバマシンへ接続しドキュメントを要求する．ドキュメントは，1つのHTMLファイル（1772バイト）と1つの画像（43770バイト）からなる．また，Netscapeが持つバッファ・キャッシュの効果の影響を受けないように，バッファ・キャッシュを無効にした状態で測定を行った．表2の各パラメータについて説明する．

(1) 共存プロセス
サーバプロセスと無限ループのプロセッサ処理を行うプロセスの共存の有無を示す．

(2) アクセス頻度
サーバマシンへ接続し，ドキュメントを要求する間隔が，10秒または30秒固定か，ランダムであるかを示す．

(3) アクセス数
サーバマシンへ接続し，1つまたは同時に3つのドキュメントを要求するかを示す．

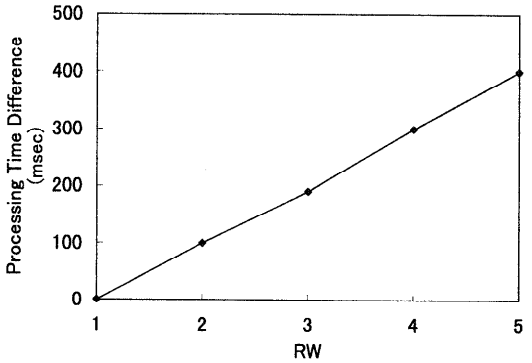

図4 評価プログラムによる結果
Fig. 4 The effect of our schedule policy (test program).

表2 測定パラメータの組合せ
Table 2 The combination of parameters for measure.

場合	共存プロセス	アクセス頻度	アクセス数
1	あり	30秒固定	1
2		30秒固定	3
3		10秒固定	3
4		ランダム	3
5	なし	30秒固定	3
6		ランダム	1
7		ランダム	3

測定は，SLPを固定したときと自動的に設定したときについて，RWを1から10まで変化させて，以下の時間を測った．

(1) URLの入力からHTMLファイル読み込み開始までの時間（T1）
(2) URLの入力からHTMLファイル読み込み終了までの時間（T2）
(3) URLの入力から画像ファイル読み込み開始までの時間（T3）
(4) URLの入力から画像ファイル読み込み終了までの時間（T4）

ここで，HTMLファイル読み込み開始とは，NetscapeがHTMLファイルの先頭データを受信したときを意味し，HTMLファイル読み込み終了とは，NetscapeがHTMLファイルの最後のデータを受信したときを意味する．なお，画像ファイルについても同様する．

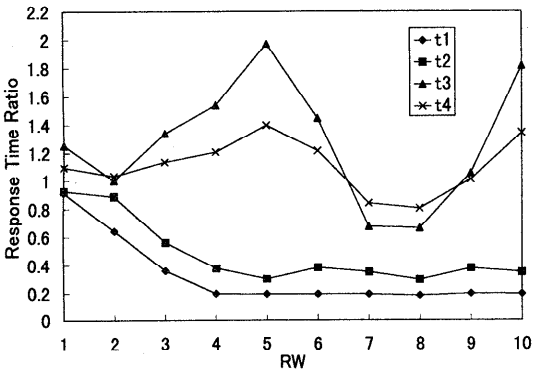

図5 場合1による測定結果（A）
Fig. 5 The effect of our schedule policy (case 1-(A)).

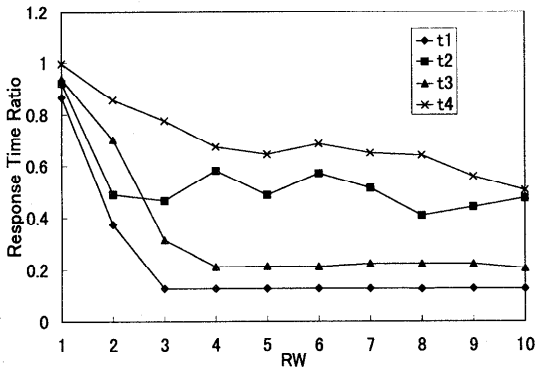

図6 場合1による測定結果（B）
Fig. 6 The effect of our schedule policy (case 1-(B)).

以降の測定した結果の各図では，スケジュール法1を利用したときの時間を利用しないときの時間で割算した値をt1，t2，t3，およびt4で示す．

5.2.1 SLPを固定

SLPを20秒に固定したとき，表2の場合1から場合7についての実測結果を以下に示す．

表2の場合1についての実測結果を，**図5**，**図6** および**図7** に示す．各図は，子サーバの動きの違いに対応した測定結果である．図5は，ドキュメント要求時に子サーバの生成処理をともなう場合（A）である．図6は，子サーバの生成や消滅をまったくともなわない場合（B）である．図7は，ドキュメント要求処理後に子サーバの消滅処理をともなう場合（C）である．場合1では，ドキュメント要求が30秒間隔であり，SLPの値（20秒）より大きいので，READYキュー操作の効果が現れ，各図とも応答時間の比は1以下になっている．特に，図6は子サーバの生成や消滅をまったくともなわない場合なので，その効果は非常に大きい．この効果は，RWが大きいほど大きく，HTMLファイル読み込みの開始（t1）は最大0.13で

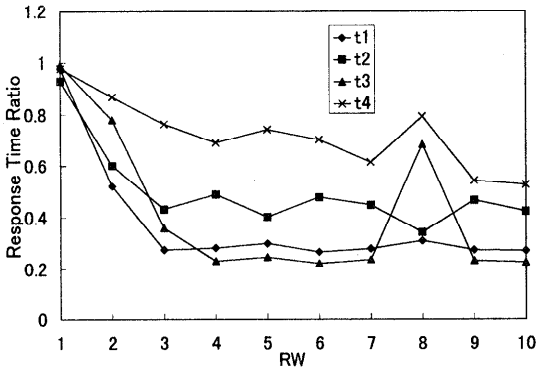

図7 場合1による測定結果（C）
Fig. 7 The effect of our schedule policy (case 1-(C)).

約9分の1，HTMLファイル読み込みの終了（t2）は最大0.53で約2分の1になる．ただし，RWを3～4以上にすることで，最大に近い効果を得ることができる．これは，測定に利用したHTMLファイルの大きさに関係している．HTMLファイルの大きさにより実DK入出力の回数が決定し，これがRW回数に関係している．図5はドキュメント要求時に子サーバの生成処理をともなう場合なので，このプロセス生成処理が影響し，図6より効果は小さい．図6に比べ，HTMLファイル読み込みの開始（t1）やHTMLファイル読み込みの終了（t2）への効果は同様であるが，画像ファイル読み込みの開始（t3）や画像ファイル読み込みの終了（t4）への効果はない．これは，画像ファイルの大きさが大きく，その実DISK入出力の回数が大きいため，プロセス生成処理の影響を受けることによる．図7はドキュメント要求処理後に子サーバの消滅処理をともなう場合であるが，子サーバの消滅処理はドキュメント要求処理後であるため，その影響は小さく，図6と同様の効果を示している．

今回の測定条件では，場合1と場合6を除き，必ず子サーバの生成処理や消滅処理が発生する．しかし，場合6は場合1と同様に子サーバの動きによって3つに分類できるので，READYキュー操作の効果を明確にするため，図12は子サーバの生成処理や消滅処理の影響を受けない場合の結果を示す．

表2の場合2についての実測結果を**図8** に示す．図6と図8の測定条件の違いは，同時にアクセスする数の違い（1または3）である．図8の場合，HTMLファイル読み込みの開始（t1）やHTMLファイル読み込みの終了（t2）への効果は，図6と同様である．ただし，RWが小さいときでも効果が大きい．これは，図8の場合は図6の場合に比べアクセス数が大きいため，サーバ内の同時走行プロセス数が大きくなり，

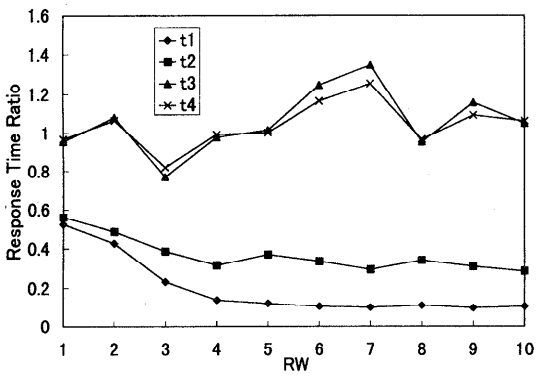

図8 場合2による測定結果
Fig. 8 The effect of our schedule policy (case 2).

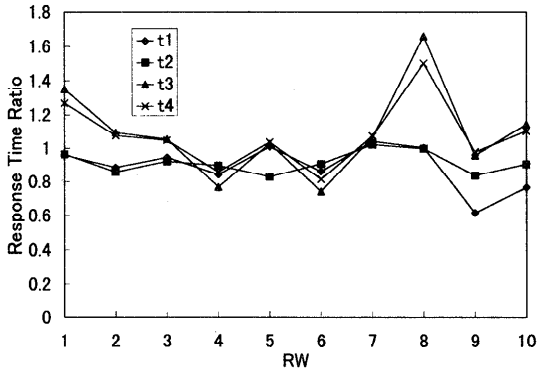

図10 場合4による測定結果
Fig. 10 The effect of our schedule policy (case 4).

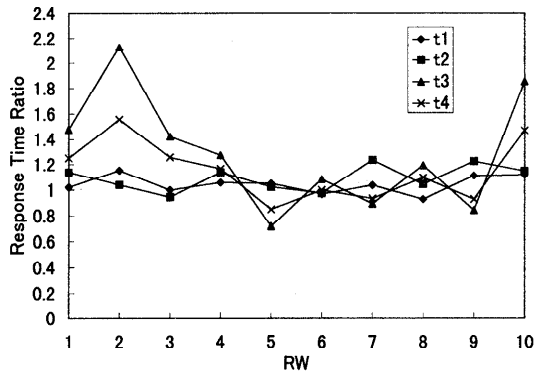

図9 場合3による測定結果
Fig. 9 The effect of our schedule policy (case 3).

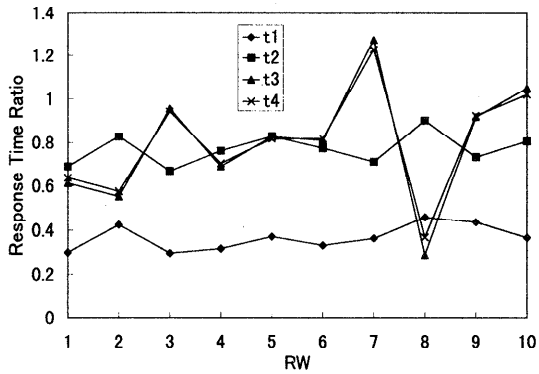

図11 場合5による測定結果
Fig. 11 The effect of our schedule policy (case 5).

READYキュー操作の効果が現れやすいためである．一方，画像ファイルはその大きさが大きいため，画像ファイル読み込みの開始（t3）や画像ファイル読み込みの終了（t4）への効果はほとんどない．

表2の場合3についての実測結果を図**9**に示す．図8と図9の測定条件の違いは，ドキュメント要求の間隔である．図8では，ドキュメント要求の間隔が30秒であり，SLPの値（20秒）より大きいので，READYキュー操作の効果を観測できた．一方，図9では，ドキュメント要求の間隔が10秒であり，SLPの値（20秒）より小さいので，READYキュー操作が行われない．このため，応答時間の比は1に近い．

表2の場合4についての実測結果を図**10**に示す．図10と図8や図9の測定条件の違いは，ドキュメント要求の間隔である．図10は，ドキュメント要求の間隔がランダム（最大30秒，最小10秒）である．このため，ドキュメント要求ごとにREADYキュー操作が行われる場合と行われない場合があり，平均値の応答時間の比は1に近い．ここで，RW＝10のとき，図8や図9の分散はそれぞれ0.00020と0.00021であるのに対し，図10の分散は0.00464であり，このことを裏付けている．

表2の場合5についての実測結果を図**11**に示す．図6と図11の違いは，共存プロセスがあるか否かである．図11では，共存プロセスがないため，図6と同様なREADYキュー操作の効果を観測することはできなかった．しかし，HTMLファイル読み込みの開始（t1）への効果は，最大0.30で約3分の1もある．これは，アクセス数が大きいため，サーバ内の同時に走行プロセス数が大きくなり，READYキュー操作の効果を観測できた．したがって，プロセッサがボトルネックの状態でない場合でも，READYキュー操作の効果があるといえる．

表2の場合6と場合7についての実測結果を図**12**と図**13**に示す．この2つの場合は場合4と同様にドキュメント要求の間隔がランダムであるため，ドキュメント要求ごとにREADYキュー操作が行われる場合と行われない場合があり，平均値の応答時間の比は1に近い．

最後に，応答時間の内訳を比較する．表2の場合1，

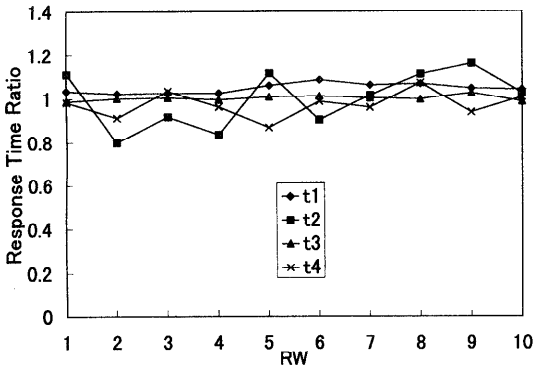

図12 場合6による測定結果
Fig. 12 The effect of our schedule policy (case 6).

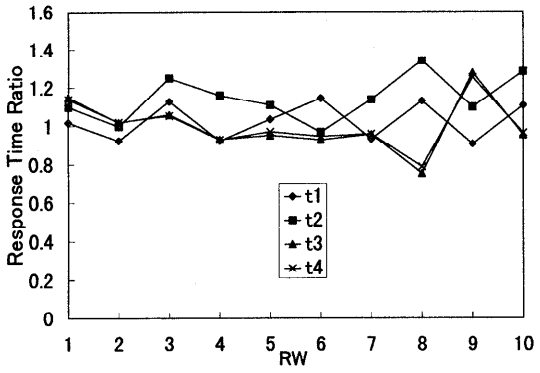

図13 場合7による測定結果
Fig. 13 The effect of our schedule policy (case 7).

場合2および場合5について，スケジュール法1を利用したときと利用しないときについて，RW = 5のときの応答時間の内訳を**図14**に示す．図14から分かるように，HTMLファイル読み込み開始までの時間（t1）への効果が非常に大きい．画像読み込み時間は，アクセス数が少ない場合は短くなっているが，アクセス数が多い場合はむしろ長くなっている．また，共存プロセスがない場合も短くなっている．

SLPの設定値の影響は大きく，提案したプロセススケジュール法の効果を大きく左右する．また，サーバプロセスの生成や消滅をともなう場合には，計算機内でのプロセス動作がPFSで予測される動作とは異なる動作になるため，提案したプロセススケジュール法の効果は小さい．サーバプロセスの生成や消滅をともなわない場合には，提案手法によりHTMLファイル読み込みの開始が約9分の1に短縮でき，HTMLファイル読み込みの終了が約2分の1に短縮できる．さらに，Netscapeからのアクセス数が多くなると，サーバの競合が発生し，提案したプロセススケジュール法の効果はRWが小さくても大きい．ドキュメント要求

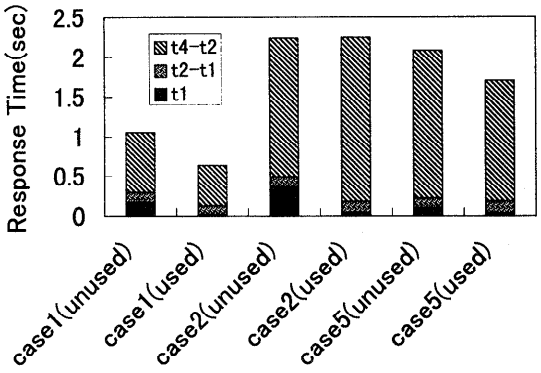

図14 場合1と場合2と場合5との比較
Fig. 14 The comparison among case 1, case 2 and case 5.

の間隔がランダムな場合は，効果の有無に差が生じ，平均的には効果が見られない．したがって，SLPを固定とした場合にはより現実的な負荷への効果は，あまりないと推察する．

5.2.2 SLPを自動化

PFSを基にSLPを単位時間30ミリ秒ごとに自動的に更新し設定したときの表2の場合1から場合4を場合1′から場合4′とし，測定した結果を図15～図20に示す．

場合1′についての実測結果を，**図15**，**図16**および**図17**に示す．各図は，表2の場合1の図5，図6および図7と同様に子サーバの動きの違いに対応した測定結果である．図15は，ドキュメント要求時に子サーバの生成処理をともなう場合であり，図5と同様な値である．図16は，子サーバの生成や消滅をまったくともなわない場合であり，図6と同様な値である．図17は，ドキュメント要求処理後に子サーバの消滅処理をともなう場合であり，図7と同様な値である．したがって，場合1は，SLPを20固定としたときが提案手法が効果的な場合であるから，SLPを自動化する設定アルゴリズムは有効であるといえる．

場合2′についての実測結果を**図18**に示す．また，場合3′についての実測結果を**図19**に示す．図18では，ドキュメント要求の間隔が固定したSLPの値より大きい場合の図8と同様なREADYキューでの操作の効果を観測することができなかった．これは，SLPの予測が外れたことが原因だと考えられる．一方，ドキュメント要求の間隔が固定したSLPの値より小さい場合の図9が，READYキューでの操作の効果を観測できなかったことに対し，図19では，READYキューでの操作の効果を観測でき，HTMLファイル読み込みの開始（t1）が最大0.19で約5分の1，HTMLファイル読み込みの終了（t2）が最大0.40で約5分の

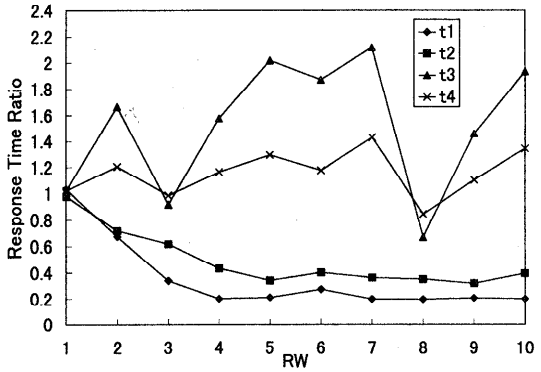

図15 場合1′による測定結果（A）
Fig. 15 The effect of our schedule policy (case 1′-(A)).

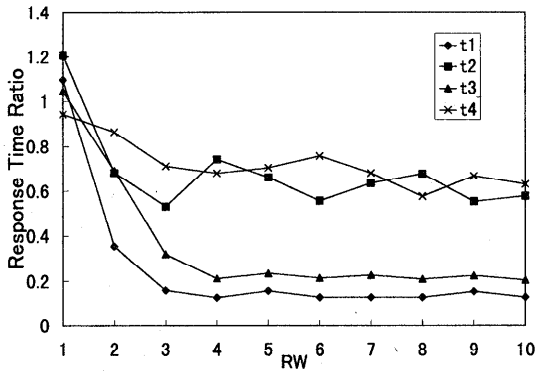

図16 場合1′による測定結果（B）
Fig. 16 The effect of our schedule policy (case 1′-(B)).

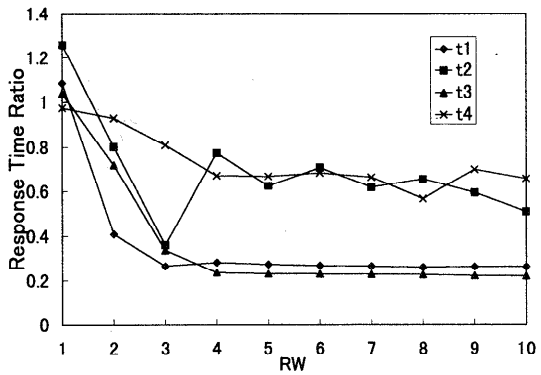

図17 場合1′による測定結果（C）
Fig. 17 The effect of our schedule policy (case 1′-(C)).

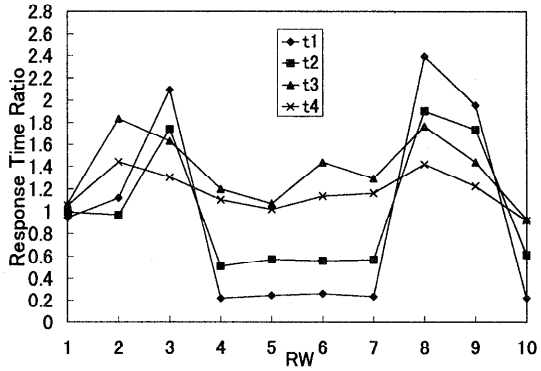

図18 場合2′による測定結果
Fig. 18 The effect of our schedule policy (case 2′).

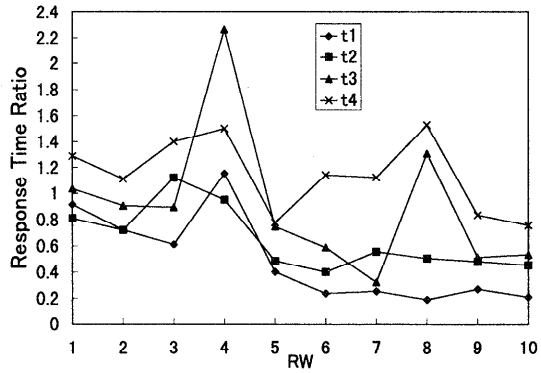

図19 場合3′による測定結果
Fig. 19 The effect of our schedule policy (case 3′).

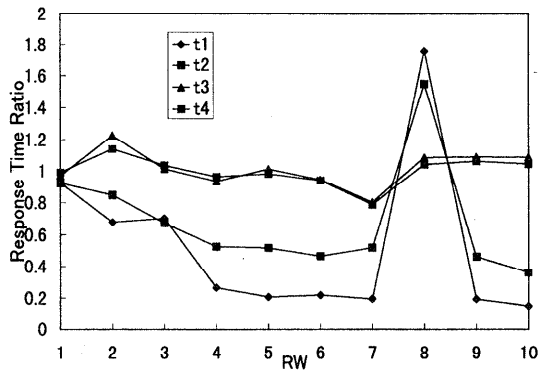

図20 場合4′による測定結果
Fig. 20 The effect of our schedule policy (case 4′).

2になった．場合2（場合2′）と場合3（場合3′）ではドキュメント要求の間隔が異なる（30秒または10秒）．したがって，SLPを決定する単位時間の値とドキュメント要求の間隔の関係が深いと推察できる．

場合4′についての実測結果を**図20**に示す．ドキュメント要求の間隔がランダムでSLPを固定した場合の図10が平均値の応答時間の比は1に近いのに対し，図20は，RWが8であるとき以外にREADYキューでの操作の効果を観測でき，HTMLファイル読み込みの開始（t1）が最大0.15で約6分の1，HTMLファイル読み込みの終了（t2）が最大0.36で約3分の1になる．RWが8であるとき，HTMLファイル読み込みの開始（t1）および終了（t2）が悪くなったことは，SLPの予測が外れたことが原因だと考えられる．

SLPを自動化としたときの提案したプロセススケジュール法の効果が，SLPを固定としたときの効果的な場合と同様な効果が見られ，SLPを自動化する設定アルゴリズムは有効であるといえる．しかし，Netscapeからのアクセス数が多くなると，ドキュメント要求の間隔の影響は大きく，提案したプロセススケジュール法の効果を大きく左右し，SLPを決定する単位時間の値とドキュメント要求の間隔の関係が深いと推察する．ドキュメント要求の間隔がランダムな場合，SLPを固定としたときには効果が見られない．しかし，SLPを自動化としたときの提案ケジュール法により，HTMLファイル読み込みの開始が約6分の1に短縮でき，HTMLファイル読み込みの終了が約3分の1に短縮できる．したがって，SLPを自動化とした場合には，より現実的な負荷への効果は大きいと推察する．

6. おわりに

READYキューでの優先制御のスケジュール法を提案し，実現時の課題と対処を示した．スケジュール法の基本的な考え方は，プロセッサがボトルネックになった場合，READYキューに繋がっているプロセスの中で，HTMLファイルの要求を処理するプロセスを優先してキューを繋ぎ換えることである．このため，HTMLファイルの要求を処理するプロセスの検出手法が必要となり，READYキューに繋がっているプロセスがHTMLファイルの要求を処理するプロセスであるか否かをプロセスの特徴を基に判断するためのしきい値SLPとRWを導入した．さらに，Netscapeを用いてSLPを固定したときとSLPを自動的に設定したときについて，RWを1から10まで変化させた場合の評価を行った．SLPを固定したときの提案したプロセススケジュール法の効果は，SLP設定値の影響により大きく左右する．また，サーバプロセスの生成や消滅をともなう場合には，計算機内でのプロセス動作がPFSで予測される動作とは異なる動作になるため，提案したプロセススケジュール法の効果は小さい．サーバプロセスの生成や消滅をともなわない場合には，提案手法によりHTMLファイル読み込みの開始が約9分の1に短縮でき，HTMLファイル読み込みの終了が約2分の1に短縮できる．さらに，Netscapeからのアクセス数が多くなると，サーバの競合が発生し，提案したプロセススケジュール法の効果はRWが小さくても大きい．ドキュメント要求の間隔がランダムな場合は，効果の有無に差が生じ，平均的には効果が見られない．したがって，SLPを固定とした場合にはより現実的な負荷への効果は，あまりないと推察する．

SLPを自動化としたときの提案したプロセススケジュール法の効果は，SLPを固定としたときの効果的な場合と同様な効果が見られ，SLPを自動化する設定アルゴリズムは有効であるといえる．しかし，Netscapeからのアクセス数が多くなると，ドキュメント要求の間隔の影響は大きく，提案したプロセススケジュール法の効果を大きく左右し，SLPを決定する単位時間の値とドキュメント要求の間隔の関係が深いと推察する．ドキュメント要求の間隔がランダムな場合，SLPを固定としたときには効果が見られない．しかし，SLPを自動化としたときの提案スケジュール法により，HTMLファイル読み込みの開始が約6分の1に短縮でき，HTMLファイル読み込みの終了が約3分の1に短縮できる．したがって，SLPを自動化とした場合には，より現実的な負荷への効果は大きいと推察する．

上述により，提案したプロセススケジュール法により，テキスト表示の開始および終了が速くなるという期待した結果が得られた．さらに，プロセッサがボトルネックの状態でない場合でも，READYキュー操作の効果があることを確認した．

残された課題として，OSが持つ入出力バッファのキャッシュがヒットする状態での評価や，RWの値の自動設定に適応するアルゴリズムに対する評価や，SLPの値とRWの値の両方の自動設定に対する評価を行う必要がある．また，SLPを決定する単位時間の値とドキュメント要求の間隔の関係を明らかにする必要がある．

参考文献

1) Sha, L. and Goodenough, J.B.: Real-Time Scheduling Theory and Ada, *IEEE Computer*, Vol.23, pp.53–62 (Apr. 1990).
2) Stankovic, J.A.: Scheduling Algorithms for Hard Real-Time Systems – A Brief Survey, *Hard Real-Time Systems*, Stankovic, J.A. and Ramamritham, K. (Eds), pp.150–173, IEEE Computer Society (1991).
3) Harbour, M.G., Klein, M.H. and Lehoczky, J.P.: Timing Analysis for Fixed-Prirority Scheduling of Hard Real-Time Systems, *IEEE Trans. Softw. Eng.*, Vol.20, No.1, pp.13–28 (1994).
4) 谷口秀夫：POS：プログラム指向スケジュールの提案，情報処理学会シンポジウム論文集，Vol.96, No.7, pp.123–130 (1996).
5) http://biztech.nikkeibp.co.jp/image/cgi/

bizfrm.cgi/general/bp980203307.html
6) 谷口秀夫：入出力要求流れの学習による入出力処理の効率化，情報処理学会シンポジウム論文集，Vol.97, No.8, pp.141–148 (1997).
7) 藤枝隆行，新城　靖，板野肯三：プロセスのグループ化によるスケジューリングとファイルのアクセス制御方式，情報処理学会オペレーティングシステム研究会，97-OS-74, pp.201–206 (1997).
8) Rootbox: *Apache UserGuide*, Prentice Hall. 三輪幸男（訳）：Apache活用ガイド，プレンティスホール (1997).
9) Laurie, B. and Laurie, P.: *Apache The Definite Guide*, O'Reilly. 田辺茂也（監訳），三代川信義（訳）：Apacheハンドブック，オライリー・ジャパン (1997).
10) Wong, C.: *Web Client Programming with Perl*, O'Reilly. 法林浩之（監訳），須田隆久（訳）：Webクライアントプログラミング，オライリー・ジャパン (1997).

(平成10年12月1日受付)
(平成11年 4 月1日採録)

スカンヤ　スラナワラッ

平成8年九州大学工学部情報工学科卒業．平成11年同大大学院修士課程修了．現在，同大大学院システム情報科学研究科博士後期課程に在学中．オペレーティングシステムに興味を持つ．

谷口　秀夫（正会員）

昭和53年九州大学工学部電子工学科卒業．昭和55年同大大学院修士課程修了．同年日本電信電話公社電気通信研究所入所．昭和63年NTTデータ通信（株）開発本部移籍．平成5年九州大学工学部助教授．平成8年九州大学システム情報科学研究科助教授．博士（工学）．オペレーティングシステム，分散処理に興味を持つ．著書「オペレーティングシステム」（昭晃堂）．電子情報通信学会，日本ソフトウェア科学会，ACM各会員．