

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リング法

スキャンヤ スラナワラツ† 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リング法を提案する。

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リング法に関する研究はさかに行われている^{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リング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リングは、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リングを行わなければ、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リングを行えば、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リング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 の考え方をういた研究を文献 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リング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リング法を提案する。

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リングを行って

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、接続待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

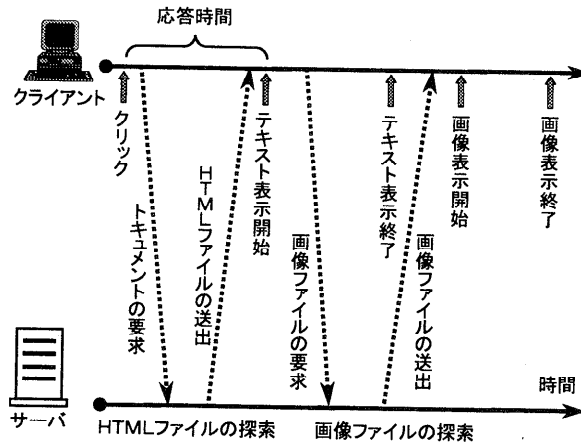


図1 HTTPトランザクションの構造
Fig.1 A structure of HTTP transaction.

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 PFS⁴⁾ (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

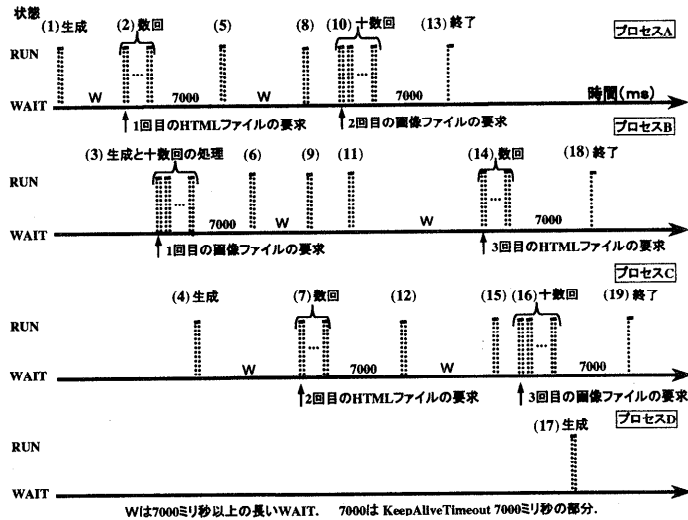


図2 PFSによるサーバの動作内容
Fig. 2 Activation flow of processes according to PFS.

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

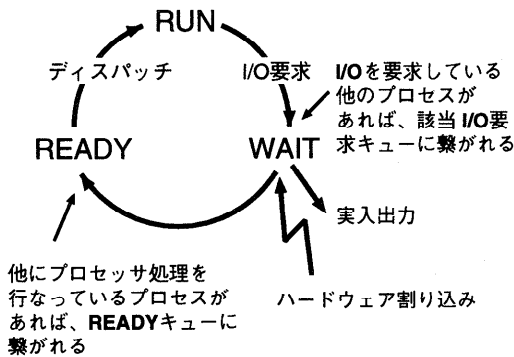


図3 子サーバの状態の変化

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リング法（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 ($\text{wait_cnt} \geq \text{SLP}$)、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 ($\text{wait_cnt} < \text{SLP}$)、

- (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 ($\text{rw_cnt} \leq \text{RW}$)、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 ($\text{rw_cnt} > \text{RW}$)、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

スケジューリング法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、スケジューリング法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、スケジューリング法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

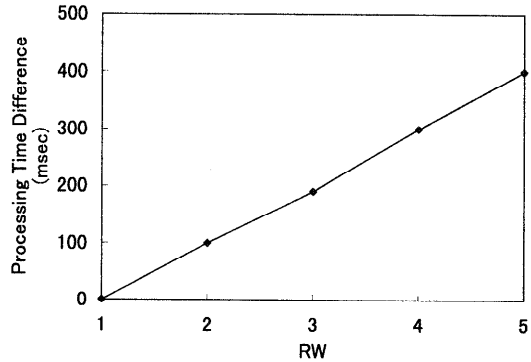


図4 評価プログラムによる結果

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

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。

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。

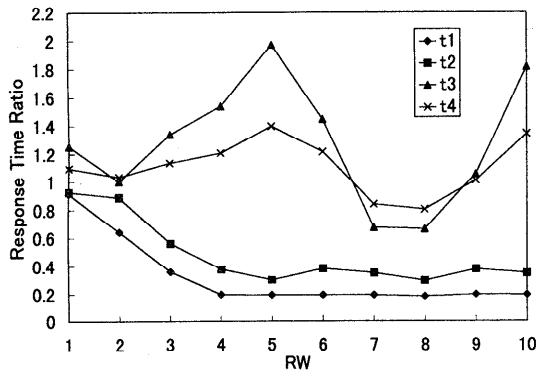


図5 場合1による測定結果 (A)

Fig. 5 The effect of our schedule policy (case 1-(A)).

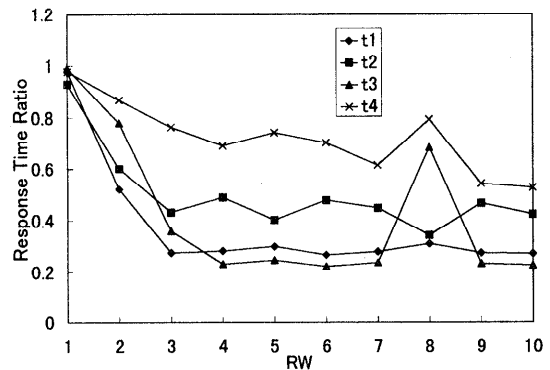


図7 場合1による測定結果 (C)

Fig. 7 The effect of our schedule policy (case 1-(C)).

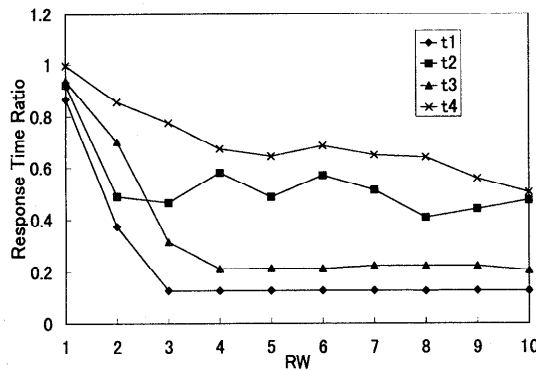


図6 場合1による測定結果 (B)

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、スケジューリング法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 t_1 , t_2 , t_3 , および t_4 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まなう場合 (A) である。図 6 は、子サーバの生成や消滅をまったくとまなわない場合 (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まなう場合 (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ま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 (t_1) は最大 0.13 で

約 9 分の 1、HTML ファイル読み込みの終了 (t_2) 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まなう場合なので、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 (t_1) や HTML ファイル読み込みの終了 (t_2) への効果は同様であるが、画像ファイル読み込みの開始 (t_3) や画像ファイル読み込みの終了 (t_4) 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まなう場合であるが、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 (t_1) や HTML ファイル読み込みの終了 (t_2) 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、

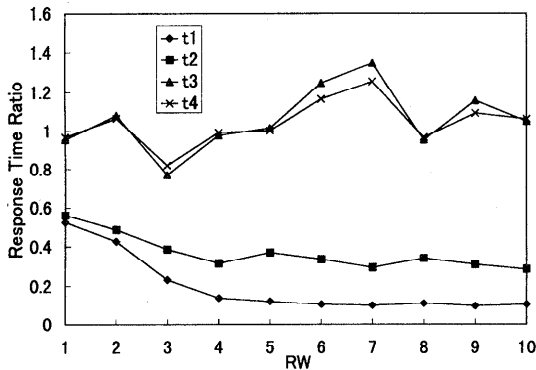


図8 場合2による測定結果

Fig. 8 The effect of our schedule policy (case 2).

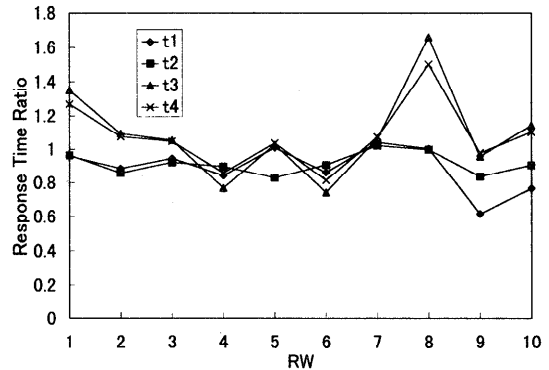


図10 場合4による測定結果

Fig. 10 The effect of our schedule policy (case 4).

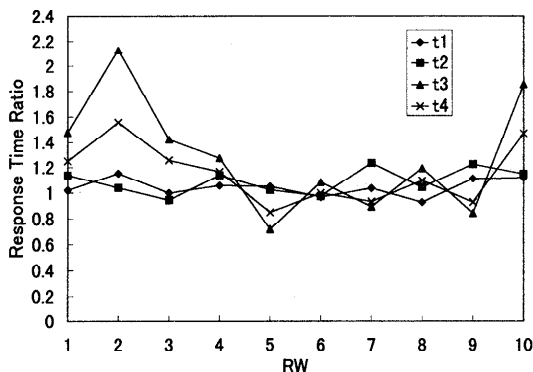


図9 場合3による測定結果

Fig. 9 The effect of our schedule policy (case 3).

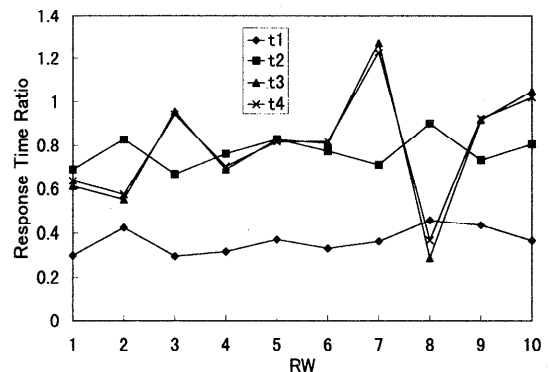


図11 場合5による測定結果

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ので、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ので、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、

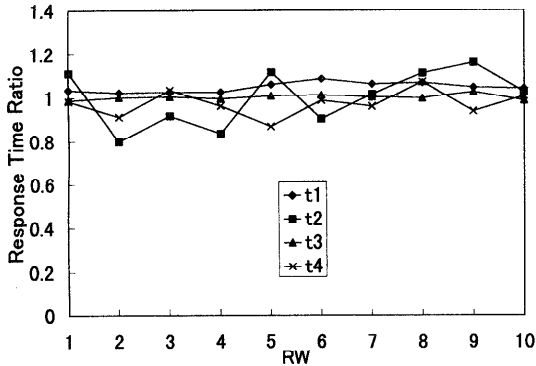


図 12 場合 6 による測定結果

Fig.12 The effect of our schedule policy (case 6).

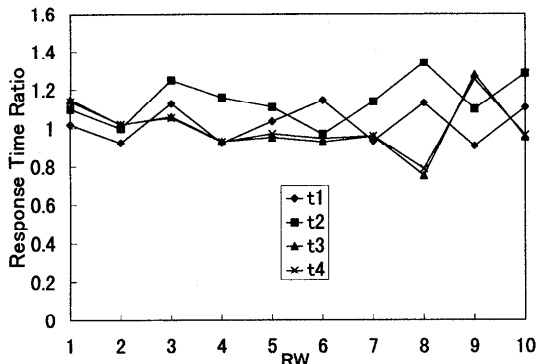


図 13 場合 7 による測定結果

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まなう場合には、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まなわない場合には、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

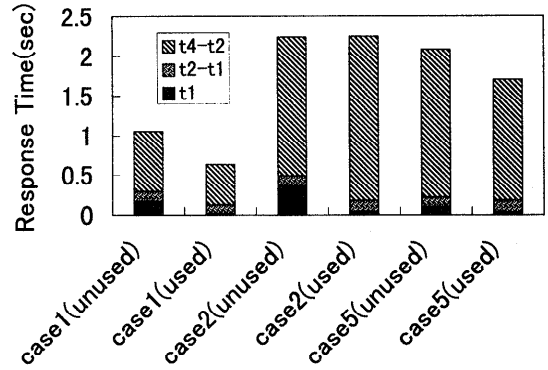


図 14 場合 1 と場合 2 と場合 5 との比較

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まなう場合であり、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ま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まなう場合であり、図 7 と同様な値である。したがって、場合 1 は、SLP を 20 固定としたときに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

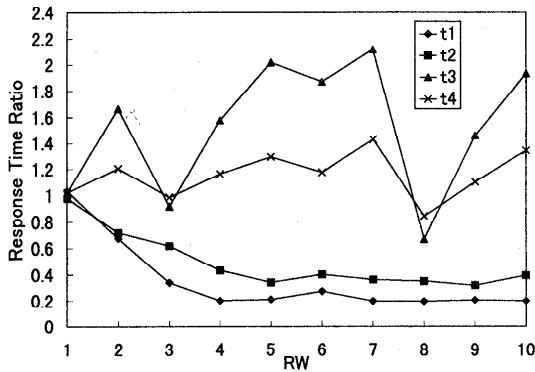


図15 場合1'による測定結果(A)

Fig. 15 The effect of our schedule policy (case 1'-(A)).

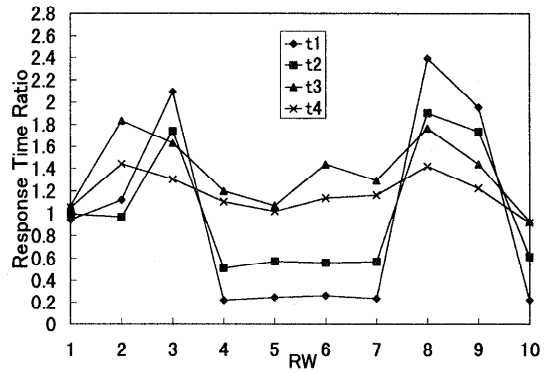


図18 場合2'による測定結果

Fig. 18 The effect of our schedule policy (case 2').

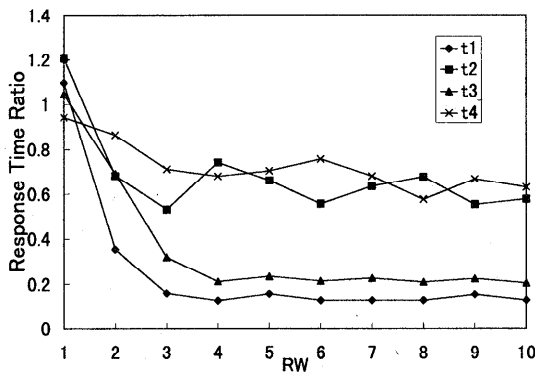


図16 場合1'による測定結果(B)

Fig. 16 The effect of our schedule policy (case 1'-(B)).

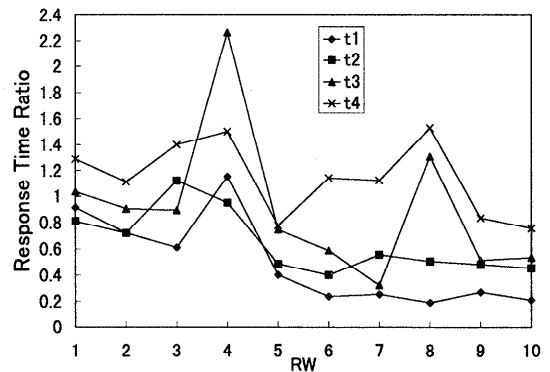


図19 場合3'による測定結果

Fig. 19 The effect of our schedule policy (case 3').

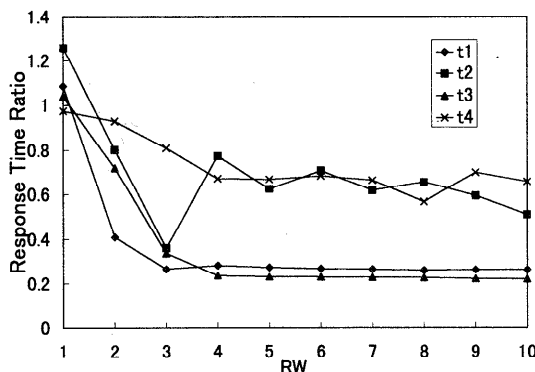


図17 場合1'による測定結果(C)

Fig. 17 The effect of our schedule policy (case 1'-(C)).

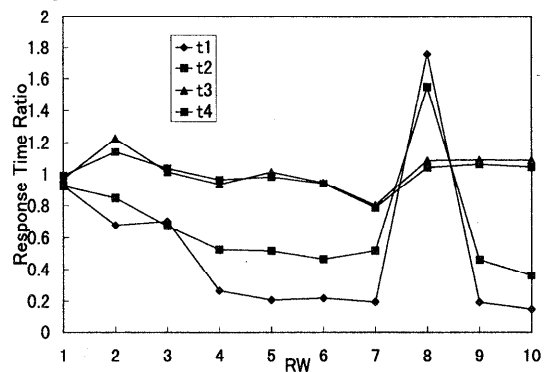


図20 場合4'による測定結果

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リング法の効果が、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リング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リング法により、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リング法を提案し、実現時の課題と対処を示した。スケジューリング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リング法の効果は、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まなう場合には、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リング法の効果は小さい。サーバプロセスの生成や消滅をとまわらない場合には、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リング法の効果は RW が小さくとも大きい。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リング法の効果は、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リング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リング法により、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リング法により、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應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 Lehoczkzy, J.P.: Timing Analysis for Fixed-Priority 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リングの提案, 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 User Guide*, Prentice Hall. 三輪幸男（訳）：Apache 活用ガイド，プレントイスホール (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)



興味を持つ。

スキャン 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



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。