2005－OS－99（5）
2005／5／25

性能パラメータの自動調整による Web サーバの性能向上

杉木 章義 † 河野 健二 ‡

† 電気通信大学大学院 電気通信学研究科 情報工学専攻
‡ 慶應義塾大学 理工学部 情報工学科
電子メール: sugiki@zeus.cs.uec.ac.jp, kono@ics.keio.ac.jp

要旨

インターネット・サーバの性能パラメータ調整は，適切な値を求めるのが難しく，多くの時間や経験を要する作業である．Web サーバの主要な性能パラメータのひとつである keep-alive 時間は，適切に調整しない場合，サーバに対する TCP 接続の使用率を低下させ，スループットを低下させることがある．本論文では，Web サーバの keep-alive 時間の自動調整機構を提案する．本機構では，データを頻繁に送受信しているクライアントからの接続は維持し，データを送受信しない接続は切断するよう keep-alive 時間を適切に調整する．本機構を Apache を対象としたライブラリとして実装し，実験を行った．その結果，異なる 2 つの負荷に対して，それぞれ keep-alive 時間を自動的に設定し，サーバの性能を向上できることを確認した．

Improving Web server performance by automatic parameter-tuning

Akiyoshi Sugiki† Kenji Kono‡

† Course in Computer Science and Information Mathematics,
Graduate School of Electro-Communications,
University of Electro-Communications
‡ Dept. of Information and Computer Science, Keio University
e-mail: sugiki@zeus.cs.uec.ac.jp, kono@ics.keio.ac.jp

Abstract

Tuning performance parameters of Internet servers is a hard and time-consuming task for administrators. This paper presents an automatic mechanism of tuning the keep-alive timeout, a performance parameter of Web servers. If it is set improperly, it may cause throughput degradation due to poor usage of connections between clients and a server. To yield higher efficiency of connections, our mechanism tries to keep active connections linked and cut off poorly used connections by adjusting the keep-alive timeout. We have implemented a prototype of this mechanism for the Apache Web server. Experimental results show that our mechanism works well on two different workloads.

1　はじめに

インターネット・サーバは急速に大型化・複雑化しており，適切な設定や性能の維持に多くの作業を必要とする．特に，サーバの性能パラメータ調整は適切な値を求めることが難しく，多くの時間や経験を要する作業である．パラメータの値を変えながら何度もテストする必要があり，適切な値はサーバのハードウェア性能，クライアントのアクセス・パターンなどによって大きく変化する．そのため，サーバ・ソフトウェア自身による自動的な性能調整を可能とする技術が求められている [1, 2, 3, 4].

Web サーバの主要な性能パラメータのひとつに keep-ailve 時間がある．**keep-alive 時間**とは，クライアントとサーバ間の接続のタイムアウト時間を決めるパラメータである．HTTP[5] の規約により，クライアントはサーバとの間で一定期間 TCP 接続を維持することができ，要求の送信やファイルの受信に利用できる．接続の切断は，データを送受信しない期間が keep-alive 時間を越えた場合に行われる．

keep-alive 時間は Web サーバの性能に大きな影響を与え，適切に調整しない場合，サーバのスループットを低下させる．keep-alive 時間を不必要に長く設定した場合，データの送受信を行わない接続が切断されるまでの期間が長くなる．そのため，サーバに対する接続の使用率が低下し，サーバのスループットを低下させる．keep-alive 時間を不必要に短く設定した場合は，頻繁にデータを送受信している接続も切断され，やはりスループットを低下させる．

本論文では Web サーバの keep-alive 時間を自動的に設定し，サーバの性能を向上させる機構を提案する．keep-alive 時間を適切に調整し，データを頻繁に送受信している接続は維持し，使用していない接続は切断する．結果として，サーバに対する接続の使用率を向上させ，サーバ全体のスループットを向上させる．

Linux 上の Apache Web サーバ [6] を対象に，提案機構を外部ライブラリとして実装した．このライブラリは実行時に Apache Web サーバと OS 間に介在する．keep-alive 時間の調整は，サーバがシステムコールを呼び出す時に，引数を書き換えることで行う．そのため，既存のサーバや OS を変更することなく利用可能である．

本機構を用いた実験を行ったところ，Web サーバを対象にした標準的なベンチマークではスループットが 86.8% 向上した．画像の多いサイトを想定した負荷に対しては，スループットを一定に保ったまま，応答時間を向上させた．

以下，2 章では keep-alive 時間の説明とその調整の必要性を示す．3 章では，keep-alive 時間の自動調整機構を示す．4 章で実装について説明し，5 章では実験とその結果を示す．6 章で関連研究について述べ，7 章でまとめと今後の課題を述べる．

2　Web サーバの keep-alive 時間

2.1　keep-alive 時間の調整の必要性

keep-alive 時間は Web サーバの性能に大きな影響を与えるため，適切に調整することが必要である．これを確認するため，Apache Web サーバに対し，標準的なベンチマークを用いて実験を行なった（実験環境の詳細は 5.1 節を参照）．表 1 に，いろいろな keep-alive 時間について，サーバ全体のスループットと平均応答時間の関係を示す．これは，サーバに接続しようとするクライアント数を 1200 とした場合の結果である．

表 1 をみると，keep-alive 時間はサーバの性能に大きな影響を与えていることがわかる．まず，スループットは keep-alive 時間を 400 ミリ秒に設定するのが一番よく，デフォルト値である 15 秒の場合に比べ 150.2% 向上している．一方，平均応答時間も 13.3% 向上している．スループットと応答時間の両方を向上させているため，keep-alive 時間を 400〜600 ミリ秒に設定した方がよい．

2.2　スループット低下の要因

keep-alive 時間が Web サーバの性能に影響を与える要因は，データの送受信を行なわない接続の維持を続けると TCP 接続の使用率が低下することにある．

図 1 に示すように，接続を維持している期間には，データを頻繁に送受信する期間と，データを送受信せず接続を使用しない期間とがある．データを頻繁に送受信する期間は，同一 Web ページを構成する画像や動画などのファイルを続けて取得するために発生する．この期間を**持続期間**と呼ぶ．一般に Web ページは複数のファイルで構成されるため，リクエストの送信が連続する．一方，接続を使用しない期

表 1: keep-alive 時間の違いによる性能の変化

keep-alive 時間	15 秒 (デフォルト)	200 ミリ秒	400 ミリ秒	600 ミリ秒	1 秒	2 秒
スループット [MBytes/sec]	7.51	14.98	18.79	17.29	14.69	11.68
(向上比 [%])	(0)	(99.4)	(150.2)	(130.2)	(95.6)	(55.5)
平均応答時間 [ms]	1016.4	1083.6	880.9	916.4	955.9	882.3
(向上比 [%])	(0)	(−6.6)	(13.3)	(9.8)	(6.0)	(13.2)

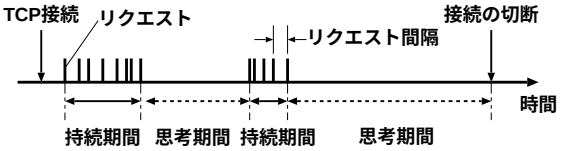

図 1: クライアントからのリクエストの流れ

間は，Web ブラウザを操作する人が次のページへのリンクをクリックするまでの，思考期間のために発生する．これを**思考期間**と呼ぶ．データの送受信を全く行わない思考期間がサーバの性能に影響を与える．

サーバの高負荷時には，思考期間による接続の使用率低下の影響が顕著に現れる．一般的な Web サーバでは，サーバの過負荷状態を防ぐため，同時接続クライアント数を制限している．高負荷時は，思考期間により使用されていない接続があるにもかかわらず，同時接続クライアント数の上限に到達し，新規にクライアントが接続できない．その結果，Web サーバのスループットが低下する．

keep-alive 時間を調整すれば，接続の使用率を向上させることができる．keep-alive 時間は，リクエストを受信する間隔と関係がある．これを**リクエスト間隔**と呼ぶ．keep-alive 接続では，リクエスト間隔が keep-alive 時間を超えると，接続は切断される．そのため，keep-alive 時間が大き過ぎれば，思考期間でも接続が維持されたままとなり，接続の使用率が低下する．逆に，keep-alive 時間が小さ過ぎれば，持続期間中も接続が切断され，スループットが低下する．keep-alive 時間を適切に調整すれば，持続期間中は接続を維持し，思考期間に入ると直ちに接続を切断することができる．その結果，接続の使用率を向上させることができ，サーバの性能が向上すると期待できる．

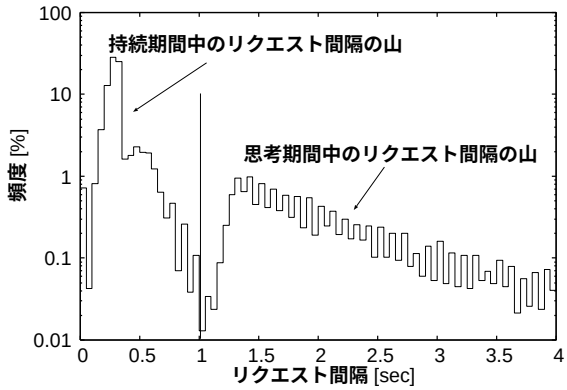

図 2: リクエスト間隔の分布

3 keep-alive 時間の自動調整機構

本章では，リクエスト間隔を用いた keep-alive 時間の自動調整機構を提案する．まず，3.1 節でリクエスト間隔の分析を行う．3.2 節でリクエスト間隔をもとにした提案手法を示す．

3.1 リクエスト間隔の分析

本機構では，持続期間中は接続を維持し，思考期間に入ったら切断するということを目指す．そのため，リクエスト間隔を分析し，リクエスト間隔がどの程度大きくなったら切断するか考える．実際に，リクエスト間隔のヒストグラムを調査した．図 2 は 2.1 節で示したデータを取得した際のリクエスト間隔のヒストグラムである．持続期間中，思考期間中の 2 つのリクエスト間隔の山が得られる．図 2 の場合，1 秒前後を境界として，2 つの山ができている．左の山は持続期間中，右の山は思考期間中である．

本機構では持続期間の山の右端の裾にあたるリクエスト間隔を見つけ，そこを keep-alive 時間として設定する．この時間を**最大持続間隔**と呼ぶ．図 2 では，最大持続間隔は 1 秒である．こうすることで，持続期間内の接続は全て維持され，思考期間に入った場合に接続を切断できる．

```
 1: function control();
 2: // arr は記録用配列 (大きさ N)
 3: // index は arr の添字 (初期値 0)
 4: begin
 5:    // (1) リクエスト間隔の記録
 6:    arr[index] := 前回のリクエストからの経過時間;
 7:    index := index + 1;
 8:    if index < N then return;
 9:    // (2) ヒストグラムの計算
10:    arr からヒストグラム hist を作成;
11:    index := 0;
12:    // (3) 最大持続間隔の計算 （3.2.1 節）
13:    for t := 0 to T − 1 do begin
14:       ratio := hist[t]/∑_{j≤t} hist[j];
15:       // 最大持続間隔 >keep-alive 時間（3.2.2 節）
16:       if keep-alive 時間に到達:
17:          err := (ratio − target)/target
18:          err に応じて keep-alive 時間を大きく.
19:          return ;
20:       if ratio < target then begin
21:          t_max := t; // 最大持続間隔の更新
22:          // (4)keep-alive 時間の更新 （3.2.3 節）
23:          t'_ka = α · t_ka + (1 − α) · t_max // 式 (1)
24:          return
25:       end;
26:    end;
27: end.
```

図 3: 本調整機構の擬似コード

3.2 調整方式

提案する keep-alive 時間調整機構は，3.1 節の手法を実現する．図 3 に調整方式全体の疑似コードを示す．関数 control() は，リクエスト間隔を測定する度に呼び出す．control() では，まず (1) 全てのリクエスト間隔を記録用配列 *arr* に記録し（5–8 行），(2) リクエスト間隔のヒストグラム *hist* を作成する（9–11 行）．*hist* は配列であり，リクエスト間隔 t を添字に与えると，そのリクエスト間隔の区間での頻度を返す．(3) 作成したヒストグラムをもとに，最大持続間隔 t_{max} を見つけ（12–26 行），(4)keep-alive 時間 t_{ka} を更新する（22–23 行）．以上により，keep-alive 時間の調整を実現する．

なお，keep-alive 時間の計算は定期的に行う．これは，Web サーバに対する負荷の変化に応じて，適切な keep-alive 時間も変化するからである．

以下，疑似コードの詳細な説明を行う．簡単のため，最大持続期間が現在の keep-alive 時間以下の場合（3.2.1 節），最大持続間隔が現在の keep-alive 時間を超える場合（3.2.2 節）に分けて説明する．最後に 3.2.3 節で，keep-alive 時間の更新方法を説明する．

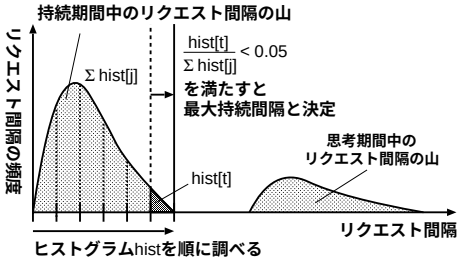

図 4: 最大持続間隔の計算

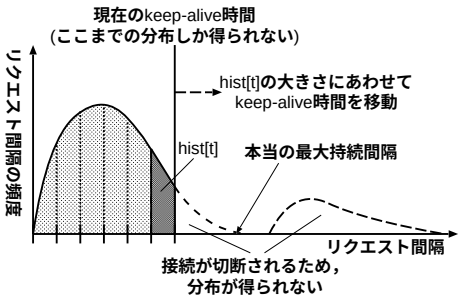

図 5: 最大持続間隔の計算 (keep-alive 時間より大きい場合)

3.2.1 最大持続間隔の計算 (最大持続間隔 ≤keep-alive 時間)

最大持続間隔を求める過程を図 4 を用いて説明する．最大持続間隔となるリクエスト間隔を求めるため，リクエスト間隔 0 から順に大きい方へ，ヒストグラム *hist* を調べていく．そして，リクエスト間隔の頻度が十分小さくなった地点を最大持続期間とする．その判定は新しく調べるリクエスト間隔 t の頻度 $hist[t]$ が，これまでの累積の頻度 $\sum_{j\leq t} hist[j]$ に占める割合を用いて行う（14，20 行目）．これにより，リクエスト間隔の山の立ち上がりに最大持続間隔が誤って設定されることを避けることができる．判定のための閾値 *target* は，現在は 0.05 としている．

3.2.2 最大持続間隔の計算 (最大持続間隔 >keep-alive 時間)

最大持続間隔が現在の keep-alive 時間より大きい場合，図 5 に示す通り，リクエスト間隔が keep-alive 時間を超える場合の接続は切断され，keep-alive 時間を超えるリクエスト間隔の分布が得られない．

本機構では，keep-alive 時間の直前の区間 t の頻度 $hist[t]$ の割合が，最大持続期間を判定する閾値

target からの誤差の大きさに応じて，最大持続間隔を現在の keep-alive 時間より大きくする．これにより，keep-alive 時間を最大持続間隔に近づけることができる．なお，keep-alive 時間を大きく調整し過ぎた場合でも，前述の 3.2.1 節の調整方式により修正されるため問題はない．

3.2.3 keep-alive 時間の更新

最後に，3.2.2 節までの調整方式により計算した最大持続間隔をもとに，keep-alive 時間を更新する．

本機構では，リクエストのバーストを考慮した keep-alive 時間の更新を行う．インターネット・サーバでは，頻繁にリクエストのバーストが観測される [7, 8, 9]．一時的なバーストによって，リクエスト間隔が大きく変化し，最大持続間隔が誤った値に設定される可能性がある．そのため本機構では，3.2.2 節までの方式で求めた最大持続間隔をただちに keep-alive 時間とするのではなく，過去の keep-alive 時間を考慮して keep-alive 時間を更新する．過去の keep-alive 時間を含めることで，一時的なバーストにより最大持続間隔が大きく変化しても，その影響を抑えることができる．実現には指数移動平均法による図 3 中の式 (1)（23 行目）を用いる．t_{max} は図 3 の擬似コードにより計算した最大持続間隔である．t_{ka} は式 (1) で計算した一つ前の時刻の keep-alive 時間であり，これを用いて新しい keep-alive 時間 t'_{ka} を計算する．α は重みであり，現在は $\alpha = 0.9$ としている．

Web サーバの起動時には，keep-alive 時間は任意の値 (例えばデフォルト値) でよい．その後，式 (1) により徐々に適切な keep-alive 時間へと近付けていく．

4 実装

3 章で述べた手法を，Linux 上の Apache 2.0.49 に実装を行った．

Apache ではクライアントからの接続ごとにプロセスを割り当て，各プロセスは I/O 入出力を監視するシステムコール `poll()` により，クライアントからの要求の送信を監視している．クライアントから要求が送信されると，その処理を行い，再び `poll()` により次の要求を待つ．keep-alive 時間は `poll()` の引数であるタイムアウト時間で指定されており，`poll()` がタイムアウトすると，接続を切断する．

本機構に必要な keep-alive 時間の調整は，この `poll()` の引数を書き換えることで実現できる．タイムアウト時間を 3 章で述べた手法で計算した値に設定することで，keep-alive 時間を調整する．その計算に必要な，リクエスト間隔の測定は `poll()` の前後の時間差を測定することで実現する．

本機構は環境変数 `LD_PRELOAD` の機能を利用し，実行時に OS とサーバ間に挿入するライブラリとして実装した．ユーザが作成したライブラリを環境変数 `LD_PRELOAD` に指定しておくと，OS やアプリケーションを改変することなしに，標準ライブラリ関数（とそれに対応するシステムコール）への呼出しをフックし，置き換えることができる．本機構では `poll()` の呼出しをフックし，タイムアウト時間の書き換えを行った．そのため，本実装は Apache と Linux を変更することなしに，利用可能である．

5 実験

提案する keep-alive 時間の自動調整機構を実験により検証した．異なる 2 つの負荷に対して，本機構によりそれぞれ適切に keep-alive 時間を設定し，Web サーバの性能が向上することを示す．keep-alive 時間を手動でさまざまな値に設定した場合と，本機構で自動的に調整した場合で比較し，本機構の有効性を検証する．

5.1 実験環境

サーバ計算機は，CPU が Pentium4 2.8GHz，主記憶 512MB，SCSI 接続，7200 回転の HDD，33MHz，32 ビット PCI バスの PC を用いた．クライアント計算機は，サーバ計算機と同じ構成のものを 16 台用いた．サーバ計算機とクライアント計算機は，1000 Base-T で 1 台のスイッチに接続されている．

Web サーバは Apache 2.0.49 を用いた．最大同時接続クライアント数を決める `MaxClients` パラメータの値は 512 とした．他の全てのパラメータは Apache の設定ファイルのデフォルト値を用いた．

表 2: 実験で用いる負荷

負荷	説明	ファイル・サイズ分布 ≤1KB	≤10KB	≤100KB	≤1MB	総ファイル・サイズ
(a) SPECWeb 標準	SPECWeb99 標準の負荷	35%	50%	14%	1%	3.6GB
(b) ファイル大	SPECWeb99 標準を平均ファイル・サイズが大きくなるように修正したもの	1%	14%	50%	35%	3.6GB

5.2 実験に用いる負荷

負荷生成は SPECWeb99[10] に思考時間を導入したものを用いた．SPECWeb99 は Web サーバを対象とした標準的なベンチマークである．しかし，サーバの性能測定を目的とするため，思考期間などのクライアントの振舞いは考慮されていない．そのため，思考期間中のリクエスト間隔を模倣するため，Pareto 分布にしたがってリクエストを生成するようにした．一般に，思考期間は Pareto 関数で近似するのがよいことが知られている [11]．Pareto 関数の係数は，文献 [11] の値を用いた．今回，動的ページに対する要求は行わず，静的ページに対する要求のみとする．

実験では，表 2 の 2 つの負荷を用いる．最初の **(a) SPECWeb 標準**は，SPECWeb99 の規約で定められている標準設定である．もう一つの **(b) ファイル大**は，画像を多く含むサイトなどを想定した平均ファイル・サイズの大きい負荷である．

5.3 実験結果: (a) SPECWeb 標準

keep-alive 時間を変化させたときの，スループットと平均応答時間の結果をそれぞれ図 6，図 7 に示す．手動で調整した場合には 400 ミリ秒が一番よい．スループットが最大であり，平均応答時間は他と変わらない値を示しているからである．本機構では，keep-alive 時間を 400 ミリ秒に近い 450 ミリ秒に設定しており，適切な調整が行われている．サーバのデフォルトの値である 15 秒と比較すると，平均応答時間は変らず，スループットは 86.8% 向上している．SPECWeb 標準では，keep-alive 時間調整の効果がスループットに大きく現れる．平均応答時間があまり変らないのは，各クライアントのリクエスト処理にかかる時間が短く，接続にかかる時間も短くなるためである．

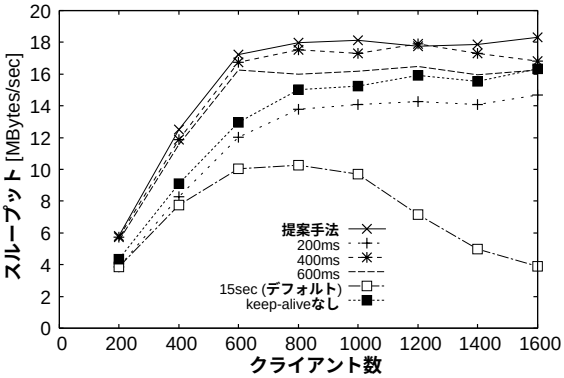

図 6: スループット (SPECWeb99)

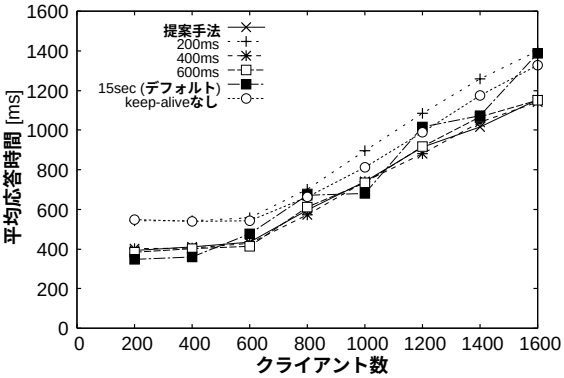

図 7: 平均応答時間 (SPECWeb99)

5.4 実験結果: (b) ファイル大

keep-alive 時間を変化させた場合のスループットを図 8，平均応答時間を図 9 に示す．スループットは，keep-alive 時間を変化させてもあまり変化しない．ファイル転送に多くの時間を要し，keep-alive 時間調整の効果が打ち消されているためである．測定のばらつきが大きいのは，ファイル・サイズの増加に伴って，ディスク性能の影響を受けているためである．一方，平均応答時間は keep-alive 時間調整の効果が現れている．手動の場合，800 ミリ秒が一番平均応答時間がよいが，本機構によりそれに近い 850 ミリ秒に調整している．ファイル・サイズの増加にともなって，クライアントあたりの接続に要す

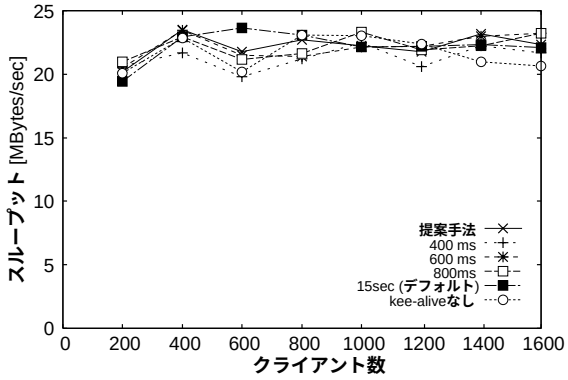

図 8: スループット (ファイル大)

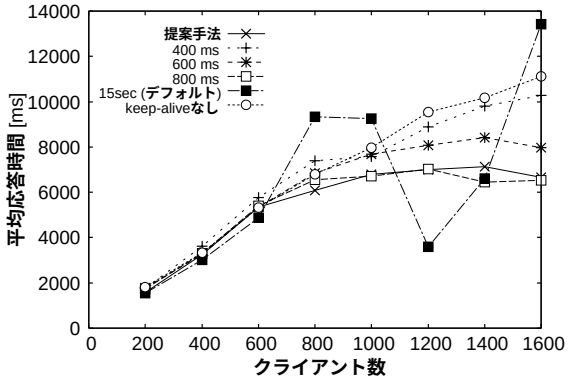

図 9: 応答時間 (ファイル大)

る時間が増加し，平均応答時間に効果が現れている．

6 関連研究

6.1 Web サーバの性能向上

文献 [12] では，keep-alive 接続を維持するためにサーバ側の資源を占有することを指摘し，サーバのメモリが少ない場合に keep-alive 接続がスループットを低下させることを実験により示している．その対策として，同一 Web ページを構成する画像や動画などをすべて取得する度に接続を切断することを推奨しており，文献中ではこれを *early close* と呼んでいる．しかし，*early close* をどう実現するかについては論じられていない．本論文で提案する手法は，この *early close* を自動的に行う機構と見ることもできる．

この他にも Web サーバの性能を向上させる研究は，サーバの実装法から既存のサーバの改良まで数多く行われている [13, 14, 15, 16, 17].

文献 [17] では，インターネット・サーバの高負荷時の振る舞いの改善を目的に，従来のプロセスやスレッドを用いたサーバ構成ではなく，ステージを用いたサーバ構成を提案している．文献 [16] は，インターネット・サーバ向けのスレッド・ライブラリを提案している．文献 [15] では，応答時間に基づく要求の受付・拒否の制御を提案している．keep-alive 時間の調整は，これらの新しい構成法のサーバでも必要であり，本研究と組み合わせることが可能である．

既存の Web サーバを対象とした研究として，文献 [13] では，Apache の発行するシステム・コール列を解析し，時間のかかるシステムコールの結果をキャッシュするなどの最適化をしている．文献 [14] は Apache を対象とし，ネットワーク I/O 関数の変更による高速化，送信するバイト列のチェックサムの計算結果のキャッシュ，TCP 階層での最適化を行っている．これらは Web サーバの低レベルな最適化を対象としており，keep-alive 時間の自動調整を行う本研究とは異なる．

6.2 モデルによるパラメータ調整

インターネット・サーバに対して，数学的なモデルや制御理論によるパラメータ調整を行っているものに文献 [18, 19] がある．文献 [18] は，Apache を対象としており，同時接続クライアント数と keep-alive 時間の 2 つのパラメータを，CPU 使用量，メモリ使用量というパラメータで調整可能とする．ある数学的なモデルを仮定し，相互の変換を行っている．文献 [19] はメモリとストレージの 2 つの資源をモデル化し，共有ホスティングを行なう Web サーバを対象に実験を行なっている．

インターネット・サーバは高負荷時に，さまざまな要素が絡み，複雑な挙動を示す．これらの研究は，CPU とメモリのみをモデル化の対象としているなど，単純なモデルを仮定している．現在の Web サーバは複雑な構成をしており，仮定しているモデルがどの程度正しいかを見極める必要がある．

keep-alive 時間などのインターネット・サーバのパラメータはサーバの負荷が低いときはあまり問題とならず，高負荷時に大きな性能差として現れる．多くのモデルを用いたパラメータ調整は，CPU 負荷が 58 % など，システムが安定して動作する負荷が低い場合を想定している．そのため Web サーバ

の性能を向上させる場合など，高い負荷が予想される場合には用いることはできない．

7 まとめ

本論文では，Web サーバを対象とした keep-alive 時間の自動調整機構を提案した．本機構は持続期間を keep-alive 接続とし，思考期間に入ると接続を切断するように keep-alive 時間を自動的に調整する．2 つの負荷に対して実験を行なったところ，それぞれ keep-alive 時間を適切に設定し，Web サーバの性能を向上できた．

今後は，インターネット環境で検証することが必要である．また，Web サーバの最大プロセス数などの別のパラメータについても自動制御機能の実装を行う予定である．

謝辞

本研究の一部は，科学技術振興機構 CREST「ディペンダブル情報処理基盤」による支援を受けている．

参考文献

[1] IBM: An architectural blueprint for autonomic computing. `http://www.ibm.com/autonomic/`.

[2] NEC: VALUMO. `http://www.sw.nec.co.jp/valumo/`.

[3] Fujitsu: TRIOLE. `http://triole.fujitsu.com/`.

[4] HITACHI: Harmonious Computing. `http://www.hitachi.co.jp/Prod/it/harmonious/`.

[5] Berners-Lee, T., Fielding, R. and Frystyk, H.: Hypertext Transfer Protocol – HTTP/1.0, *RFC 1495* (1999).

[6] The Apache Software Foundation: Apache HTTP Server. `http://www.apache.org/`.

[7] Jung, J., Krishnamurthy, B. and Rabinovich, M.: Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites, *Proc. WWW2002* (2002).

[8] Fox, A., Gribble, S., Chawathe, Y., Brewer, E. and Gauthier, P.: Cluster-Based Scalable Network Services, *Proc. 16th ACM Symp. on Operating System Principles*, pp. 78–91 (1997).

[9] Welsh, M. and Culler, D.: Adaptive Overload Control for Busy Intenet Servers, *Proc. 4th USENIX Symp. on Internet Technologies and Systems* (2003).

[10] Standard Performance Evaluation Corporation: The SPECweb99 benchmark. `http://www.spec.org/osg/web99/`.

[11] Barford, P. and Crovella, M.: Generating Representative Web Workloads for Network and Server Performance Evaluation, *Proc. ACM SIGMETRICS'98*, pp. 151–160 (1998).

[12] Barford, P. and Crovella, M.: A Performance Evaluation of Hyper Text Transfer Protocols, *Proc. ACM SIGMETRICS'99* (1999).

[13] Hu, Y., Nanda, A. and Yang, Q.: Measurement, Analysis and Performance Improvement of the Apache Web Server, *Proc. 18th IEEE Int'l Performance, Computing and Communications Conference* (1999).

[14] Nahum, E., Barzilai, T. and Kandlur, D. D.: Performance Issues in WWW Servers, *IEEE/ACM Transations on Networking*, Vol. 10, No. 1, pp. 2–11 (2002).

[15] Pai, V. S., Druschel, P. and Zwaenepoel, W.: Flash: An Efficient and Portable Web Server, *Proc. 1999 USENIX Annual Technical Conference* (1999).

[16] von Behren, R., Condit, J., Zhou, F., Necula, G. C. and Brewer, E.: Capriccio: Scalable Threads for Internet Services, *Proc. 19th ACM Symp. on Operating Systems Principles* (2003).

[17] Welsh, M., Culler, D. and Brewer, E.: SEDA: An Architecture for Well-Conditioned, Scalable Internet Services, *Proc. 18th ACM Symp. on Operating Systems Principles*, pp. 230–243 (2001).

[18] Diao, Y., Gandhi, N., Hellerstein, J., Parekh, S. and Tilbury, D.: Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics With Application to the Apache Web Server, *Proc. 7th IEEE/IFIP Symp. on Integrated Network Management* (2001).

[19] Doyle, R. P., Chase, J. S., Asad, O. M., Jin, W. and Vahdat, A. M.: Model-Based Resource Provisioning in a Web Service Utility, *Proc. 4th USENIX Symp. on Internet Technologies and Systems* (2003).