

Web 処理を効率化するアクセススケジューリング機構について

加地智彦[†] 最所圭三^{††}

インターネットの普及により、Web サーバの過負荷問題は深刻さを増している。我々は、過負荷時にユーザに対して待ち時間を示すことで不満を軽減させる Web システムの開発を行っている。本稿では提案の Web システムの基本的なモジュールとなる Web アクセススケジューリング機構について報告する。

An Access Scheduling Mechanism to Promote Efficiency of Web Processing

Tomohiko Kaji[†] and Keizo Saisho^{††}

The overload problem of Web servers increases seriousness by the popularization of the Internet. The authors develop a Web system which tells users wait time to reduce their complaint in busy time. In this paper, the access scheduling mechanism which is the fundamental method of the Web system is described.

1. はじめに

インターネットにおいて最も大きな役割を果たす Web サーバはブロードバンドの普及に伴い負荷が増大し続けている。負荷の増大に対応するため、ミラーリングなどのサーバ負荷分散技術が発展してきた。しかしながら、Web サーバへのアクセス要求は急な増減が多く、予測が困難であるため、せっかく投資したハードウェアが無駄になることも多い。また、ハードウェアを増やせば増やしただけ、コストが増加する。このため、コンテンツ提供者は、ハードウェアの追加投資に慎重にならざるを得ない。サーバ提供者の見積もりを越えた負荷がかかれば、利用者はコンテンツを満足に閲覧することができないなどの不都合が生じる。

我々はシステムが過負荷にならないように同時に処理可能なアクセス数を制限するが、これに

よって拒絶されたユーザのアクセスに対していつになればコンテンツを利用できるようになるかを教えることで、不満を軽減させる Web システムの開発を行っている。この Web システムの主な機能として、動的負荷制御とアクセス拒絶を受けたユーザに対しての接続可能な時間帯を保証する機能とがある。これらの機能を実装するために、応答時間の予測を可能にする Web アクセススケジューリング機構を試作した。

本稿では、この Web アクセススケジューリング機構の概要と評価した結果について述べる。

2. 提案 Web システムの概要

Web サーバの過負荷に対してユーザの不満を緩和する Web システム概要を示す。

この Web サーバは以下の機能を持つ。

- i. 動的負荷制御機能
- ii. i の機能によってアクセスを拒絶されたユーザに対して通知した時間にアクセスを行えば、それを受け付けることを保証する機能
- iii. クライアント・サーバ間で協調して効率的な通信を行う

i の機能は、Web サーバの負荷状況に応じて

[†] 香川大学大学院工学研究科

Graduate School of Engineering,
Kagawa University

^{††} 香川大学工学部

Faculty of Engineering, Kagawa University

アクセス制御を行う機能である。この機能によってサーバのダウンを防ぎ、かつ処理中の Web アクセスの応答速度を確保する。

ii の機能は、i の機能によってアクセスが拒絶されたユーザに対して、どのくらい時間が経てばアクセスが可能かを伝え、ユーザが通知した時間にアクセスすれば、それを受け入れることを補償する機能である。

iii の機能は、クライアント・サーバ間で協調することで、例えば、ii の機能でサーバが教えた時間帯よりも早くサーバが過負荷状態より脱した時にクライアントにそれを教え、http コネクションを確保する、といった機能が挙げられる。

これらの機能を持つ Web システムを開発することで、過負荷時のサーバダウンを防ぎ、ユーザの不満を緩和し、効率的な Web アクセスが可能となる。

3. アクセススケジューリング機構の概要

2 節で述べた機能のうち、動的負荷制御機能とアクセスを拒絶されたユーザに対しての救済機能を実装するためには、以下の二つの要件を満たさなければならない。

- i. クライアントへの応答時間を知ること
- ii. 応答時間を予測すること

i については、Web サーバデーモンの応答時間をそのままログに残せば良いように思えるが、Apache では最大同時接続数の設定値よりも多いアクセスがあった場合、TCP コネクションで待機させるが、この間の時間はログに残る応答時間に加味されず、クライアントまでの応答時間を計測することが出来ない。

ii については、OS のスケジューラの影響が大きい。同時に多数の Web アクセスがあった場合、その処理のために同時に多数のプロセスが起動され、各プロセスに割り当てられる CPU 時間は

非常に短くなる。Web アクセスのタイミングによってはなかなか CPU 時間が割り当てられず、急に応答時間の伸びるものが発生する[2]。そのような Web アクセスが現れた場合、応答時間の予測は難しくなる。

以上の点を解決するための機構を試作した。この機構は、Web アクセスレベルでのキューイングによるスケジューリングを行うことで、クライアントまでの応答時間をサーバ側で把握できるようにする。更に、同時アクセス数と処理を待機しているアクセス数を制御することで、応答時間の予測を行うことができる。

このスケジューリングの概念を図 1 に示す。Apache により accept されたアクセスを三つのグループに分ける。Run_Ready アクセスグループは、そのまま通常の処理を行うアクセスグループである。このグループに入るアクセスの数は、同時実行可能な数で決定される。Wait アクセスグループは、アクセスの処理直前で待機するアクセスグループであり、到着時間順にキューイングされる。キューの大きさはメモリの大きさや各アクセスの処理時間などの条件により決定される。最後の Next_Wait アクセスグループは、アクセスが拒否されたが、次回接続可能な時間を保証されたアクセスである。ディスク上にアクセス内容などを記録しておく。

4. アクセススケジューリング機構の設計

3 節で述べたアクセススケジューリング機構における処理の流れについて述べる。現在、Run_Ready アクセスグループと Wait アクセスグループの処理までを実装した。Next_Wait アクセスグループに入るアクセスは現在では単にエラーを返すだけとしている。

試作したスケジューリング機構の動作を述べる。この機構は Apache 処理サイクル（図 2）中のアクセス制御ステップとログ出力ステップに組み込まれる。新しいアクセスがあると、accept され、Apache の処理が始まり、いくつかの処理ステップを経た後、アクセス制御ステップに移る。

アクセス制御ステップに組み込んだモジュールは、各アクセスを次のように処理する。

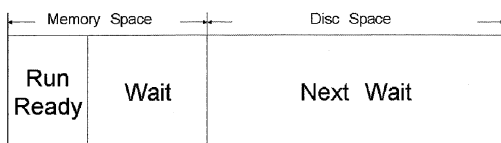


図 1 : アクセススケジューリングの概念

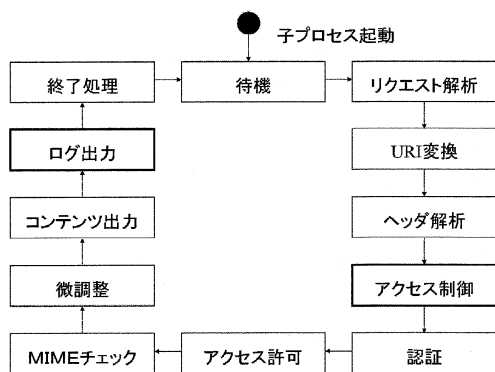


図 2：Apache の処理サイクル[3]

- Run_Ready アクセスグループが一杯でなければ、Run_Ready アクセスグループに入れ、次の処理ステップに移る。
- Run_Ready アクセスグループが一杯であり、Wait アクセスグループが一杯でなければ Wait アクセスグループに入れる。このとき、シグナル待ちにすることにより次の処理ステップに移らないようにしている。シグナルを受信すれば、Wait アクセスグループから Run_Ready グループに移動し、次の処理ステップに入る。
- Wait アクセスグループも一杯ならば、Next_Wait グループに入り、アクセスを拒絶される。現状では、HTTP エラー503のステータスを与えられ、ログ出力ステップに移る。
(将来は、アクセス情報と次回接続可能な時間帯を教えるからアクセス拒絶を行う予定である)

ログ出力ステップに組み込んだモジュールは、以下の処理を行う。

- もし、Wait アクセスがキューの中に入っていたら、その先頭のアクセスに対してシグナルを送信する。
- もし、ステータスコードが Next_Wait アクセスグループに属するアクセスならば、しばらくの間処理をスリープさせる。これは、クライアントが拒絶後直ちにリトライすることにより負荷が軽減しないことを防ぐため

である。

5. アクセススケジューリング機構の評価

本節では、試作したアクセススケジューリング機構を以下の点について評価した。

- クライアントにレスポンスが返るまでの応答時間を知ること
- 将来の応答時間を予測すること

実験に使用した計算機環境を表 1 に示す。ネットワークは 1000baseT の LAN である。Web サーバデーモンとして Apache2.20 を使用する。また、実験用の Web アプリケーションとして、KENT 氏作成の「ASKA BBS」[4]を用いる。

表 1：実験環境

	クライアント	サーバ
CPU	Athron64 x2 4400+	Xeon 2.4GHZ *2
メモリ	1GB	1GB
OS	WindowsXP SP2	FedoraCore 3

実験 1：

まず、i に関する実験を行った。実験用のテストツールとしてクライアントでの応答時間を計測できる Jmeter2.2[5]を使用した。150 個のスレッドを生成し、10KB の文字データを POST するアクセス要求を 60 回繰り返す負荷テストを行った。サーバ側の設定として、アクセススケジューリング機構を用いない場合は MaxClients を 10、ListenBackLog の大きさを 5,000、TimeOut 時間を 300 秒とした。アクセススケジューリング機構を用いる場合は、MaxClients を 1,000 とし、その他の設定値は機構を用いない場合と同じにした。スケジューリング機構用の設定として、Run_Ready グループの上限を 10、Wait グループの上限を 200 とした。

実験結果を図 3、4 に示す。図 3 がモジュール適用前、図 4 がモジュール適用後のクライアント及びサーバでの応答時間のグラフである。モジュール適用前後でクライアントとサーバで計測した応答時間の相関に大きな差があることが分かる。相関係数を計算してみると、モジュール適用前は 0.62、適用後は 0.99 であり、モジュールを

適用することによって、i の点を満たしていることがわかる。

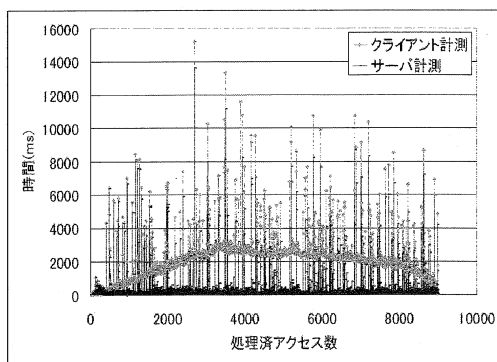


図 3：モジュール適用前の負荷実験

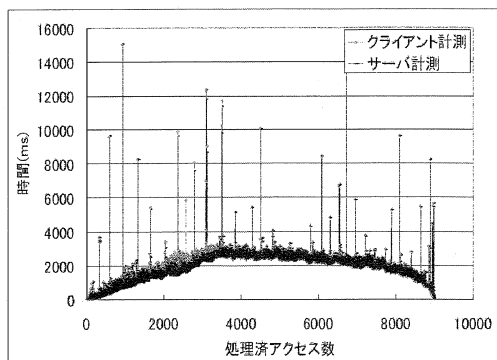


図 4：モジュール適用後の負荷実験

実験 2：

次に、ii 関する実験を行った。実験用のテストツールとして、ここでは Jmeter より負荷量を把握しやすい Web Application Stress Tool[6]を用いる。サーバ側の設定は MaxClients を 1,000、ListenBackLog を 5,000、Timeout を 300 秒とする。

まず、同時アクセスの数を変えて実験を行った。Run_Ready グループの数は 10 個、Wait キューの大きさは 200 に固定した。モジュール適用後の Web サーバに対して 1KB の文字データを書き込むアクセスを 50 個、100 個、150 個と変え、それぞれ 300 秒間継続して行うテストを行った。その結果を図 5 と表 2 に示す。図 5 は Wait キューに入っているアクセスの数と応答時間の関係

を示したグラフである。表 2 は応答時間及びスループットを示す。

このモジュールを適用することによって、ある程度のばらつきはあるものの、Wait キューの中のアクセスの数と応答時間がほぼ正比例していることが、図 5 のグラフより分かる。このモジュールを使用することで、将来の応答時間の予測が可能である。

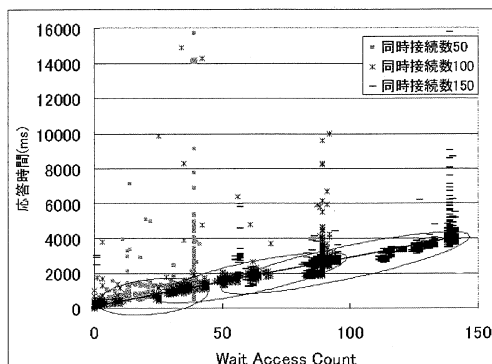


図 5：Wait キューの大きさと応答時間の関係

表 2：応答時間及びスループット

同時アクセス数	50	100	150
全アクセス数	11,080	11,003	10,950
スループット (Access/Sec)	37	36.6	36.5
平均応答時間 (ms)	1,297	2,466	3,898
最大応答時間 (ms)	15,728	14,882	15,807
標準偏差	352	665	467

実験 3：

次に、Wait キューの大きさを超えるアクセスをサーバに対して行う実験を行った。Run_Ready グループの数は 10 個、Wait キューの大きさは 150、同時アクセスは 150, 180 の場合に分け、300 秒間の負荷テストを行った。

実験結果を図 6、7、8 及び表 3 に示す。図 6 は 150 個同時アクセスをかけた場合、図 7 は 180 個同時アクセスをかけた場合のグラフである。図 8 は、図 7 から 503 エラーのアクセスを取り除いたグラフである。表 3 は応答時間及びスループットの表である。

503 エラーアクセスを返したとき、3 秒間のスリープ時間を与えているため、図 7 では変動の大きいグラフになっているが、503 エラーのアクセスを除くと、図 8 に示すようにばらつきは小さくなっている。図 6 と図 8 と比較すると図 8 は多少ばらつきが大きくなっているが、ある程度一定の応答時間を維持しているため、たとえ同時アクセス数が Wait キューの大きさを超えたとしても、応答時間を予測することは可能である。

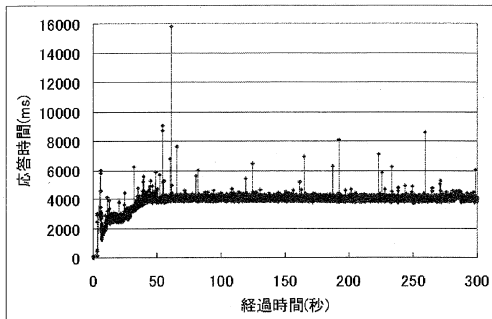


図 6：150 個同時アクセスをかけた場合

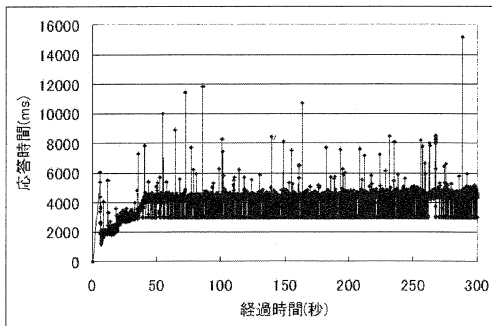


図 7：180 個同時アクセスをかけた場合
(503 エラーアクセス含む)

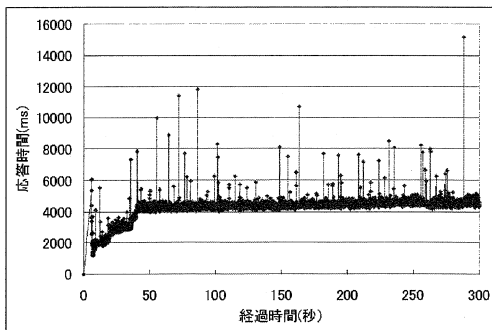


図 8：180 個同時アクセスをかけた場合
(503 エラーアクセス含まず)

表 3：応答時間及びスループット

同時アクセス数	150	180	180 (除 503)
拒否率	0%	57%	0%
全アクセス数	10,950	24,762	10,590
スループット(Access/Sec)	36.5	35.3	35.3
平均応答時間(ms)	3,898	3,528	4,210
最大応答時間(ms)	15,807	15,165	15,165
標準偏差	467	754	671

実験 4：

最後に、Run_Ready グループの数を変えた実験を行った。Wait アクセスのキューの大きさは 150、同時アクセスは 150 で、Run_Ready グループの数を 10、40、70 と変え、300 秒間の負荷テストを行った。結果を図 9、10、11 及び表 4 に示す。図 9 が Run_Ready の数が 10、図 10 が 40、図 11 が 70 のときの応答時間を示すグラフである。表 4 は応答時間及びスループットを示す。

表 3 から、Run_Ready の数が大きくなればなるほど平均応答時間は短くなっていくが、応答時間のばらつき及び最大応答時間が大きくなった。その様子は図 9～11 のグラフからも明らかである。

最大応答時間が大きくなった原因については、Run_Ready アクセスグループの数が増えることによって、タイミングによって処理が伸びるアクセスが多くなると共に最大応答時間も伸びてしまったためと推測できる。平均応答時間が短くなるのは Run_Ready の数が大きくなったことにより Wait キューで待つアクセスの数が減ったため(図 9：140、図 10：110、図 11：80)、Wait キューでの待ち時間が短くなったことが原因と考えられる。

以上の実験結果より、アクセススケジューリング機構を導入することによって、クライアントで計測される応答時間とほぼ等しい時間をサーバ側で計測でき、かつ Run_Ready アクセスグループと Wait アクセスキューの大きさを適切に設定することで、応答時間の予測が可能になることを確認できた。

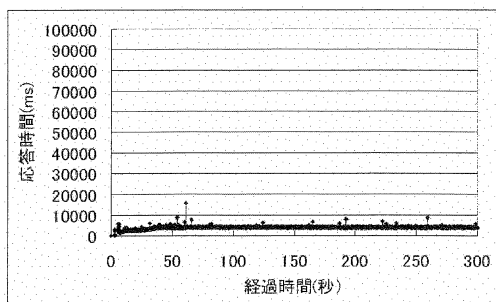


図 9 : Run_Ready の数が 10

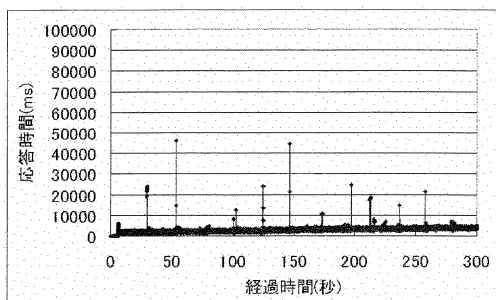


図 10 : Run_Ready の数が 40

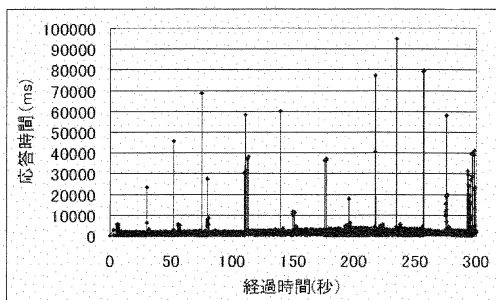


図 11 : Run_Ready の数が 70

表 4 : スループット及び応答時間

Run_Ready 閾値	10	40	70
全アクセス	10,950	11,077	11,064
スループット(Access/Sec)	36.5	36.5	36.5
平均応答時間(ms)	3,898	2,825	1,591
最大応答時間(ms)	15,807	46,282	94,922
標準偏差	467	1,252	3,054

6. まとめと今後の課題

Web サーバの過負荷問題を解決するために、アクセス制御を行うが、アクセスを拒絶されたユーザに対して次回接続可能な時間帯を教えることで不満を緩和する Web システムを提案した。この Web システムの基本機能の一つである Web アクセスレベルでのスケジューリングを行う機構を試作した。この機構を用いることにより、クライアントから見えるサーバの応答時間をサーバ側から計測することが可能になった。そして、応答時間の予測が可能になった。

今後の課題として、

- ・ 様々な Web アプリケーションを動かした場合の応答時間予測の方法の確立
- ・ 応答時間予測手法の確立
- ・ 適切な閾値の動的制御方法の確立

などが挙げられる。

参考文献

- [1]Apache、
<http://httpd.apache.org/>
- [2]加地智彦、最所圭三：Web サーバの動的負荷制御のための負荷解析、DICOMO2006 シンポジウム、pp.757-760、2006
- [3]小山浩之：Web エンジニアのための Apache モジュールプログラミングガイド、技術評論社、2003
- [4]ASKA BBS、
<http://www.kent-web.com/>
- [5]Jmeter、
<http://jakarta.apache.org/jmeter/index.html>
- [6]Web Application Stress Tool、
<http://www.microsoft.com/japan/technet/archive/itsolutions/intranet/downloads/webstres.mspx?mfr=true>