推薦論文●ディペンダブルソフトウェア

過負荷時のWebアプリケーションの性能を改善する
Session-aware Queue Scheduling

松沼 正浩　光来 健一　日比野 秀章　佐藤 芳樹　千葉 滋

本稿では過負荷時に Web アプリケーションサーバの性能低下を防ぐ *Session-aware Queue Scheduling*（*SQS*）を提案する．既存の多くのアドミッション制御はサーバ全体またはページを単位として制御を行うため，セッションを利用する現実のワークロードには適していない．実際のワークロードを重視したスケジューリングを行うために，SQS ではページ単位のスケジューラとセッション単位のスケジューラを連携させる．**ページスケジューラ**は Web アプリケーションの進捗を監視することで，最適なページ処理並行度を動的に決定する．**セッションスケジューラ**はページスケジューラの待機リクエストのためのキューを監視することで最適な同時セッション数を動的に決定する．我々はこれらのスケジューラを Tomcat 上に実装し，セッションを考慮したワークロードを用いて実験を行った．その結果，SQS を用いた場合には過負荷時のセッション処理性能を改善できることが確認できた．

This paper proposes *session-aware queue scheduling (SQS)*, which prevents performance degradation of web application servers under overload. Most of existing admission control are not suitable for real workloads using sessions since they limit the number of concurrent processing either to the whole server or each page. To schedule request processing under real workloads, SQS cooperates page-level schedulers and session-level schedulers. A *page scheduler* monitors the progress of a web application and detemines an appropriate concurrency level for processing each page. A *session scheduler* monitors the queue of page requests waiting to be processed in front of each page scheduler and determines the appropriate number of concurrent sessions. We have implemented these schedulers in Tomcat and performed experiments using workloads including sessions. The result shows that SQS improves server throughput for session.

1　はじめに

商用サイトでは複雑な処理を行うために Web アプリケーションサーバが用いられるようになってきている．Web アプリケーションとは動的に Web ページを生成するプログラムであり，Servlet などを用いて作成される．Web アプリケーションサーバでは処理の単位としてセッションという単位が用いられることが多い．セッションとは，ユーザを特定して，複数のページにまたがる処理を一貫して行えるようにするものである．セッションを使えば，例えば，個人認証，商品検索，商品決定，個人情報入力，という処理をそれぞれのページに分けて行うことができる．これらの一連のページはセッションにより特定のユーザに関連づけられる．セッションは個人認証から個人情報入力までの全てのページを完了して初めて利益となるため，Web アプリケーションサーバの性能を向上させるには，ページ単位の処理性能だけでなくセッションを単位とした処理性能も向上させる必要がある．

しかしながら，従来の Web アプリケーションサーバでは大量のクライアントからのアクセスが集中するとセッション処理性能が大幅に低下してしまう．その原因の 1 つは，過負荷時には各ページの生成処理でリソース競合が発生する可能性があることである．

Session-aware Queue Scheduling for Improving Performance of Web Applications under Overload.
Masahiro MATSUNUMA, 東京工業大学（現在 NTT コムウェア）, Tokyo Institute of Technology (currently NTT COMWARE).
Kenichi KOURAI, Hideaki HIBINO, Shigeru CHIBA, 東京工業大学, Tokyo Institute of Technology.
Yoshiki SATO, 東京工業大学（現在 三菱総合研究所）, Tokyo Institute of Technology (currently MRI).
コンピュータソフトウェア, Vol.23, No.2 (2006), pp.1–12.
[論文] 2005 年 5 月 20 日受付.
本論文はディペンダブルソフトウェア研究会から本誌に推薦されたものである．

静的な HTML ページのようにリソースをあまり消費しないページの生成処理は，並行処理することで余剰リソースを効率よく使用し，処理性能を向上させることができる．しかし，Servlet により実装される大量のリソースを消費するページ生成処理を単純に並行処理した場合，リソース競合が発生して処理性能の低下を引き起こす [16]．このようなリソース競合を解消するには，大量にリソースを消費するページの生成処理を制御する必要がある．

もう 1 つの原因は，このような大量にリソースを消費するページの処理性能が低下した結果，セッション処理が途中で失敗してしまう可能性があることである．セッション処理はセッションを構成する複数のページを全て処理してはじめて成功となる．しかし，過負荷時に各ページの処理に時間がかかりすぎると，ブラウザやサーバのタイムアウトが発生したり，ユーザが待ち切れずに読み込みを中止したりして，セッションが途中で中断される場合がある．このようにセッションの途中のページで処理が失敗した場合，セッションの最初から処理をやり直さなければならない場合も多く，それまでに行ってきた処理が無駄になる．このような無駄な処理を減らすためには，セッションの途中失敗を防ぐ必要がある．

そこで我々は，セッション処理性能を向上させるために，ページ単位のスケジューラとセッション単位のスケジューラを連携させる *Session-aware Queue Scheduling*（*SQS*）を提案する．**ページスケジューラ**は個々のページ毎に用意され，リソース競合を起こさずに効率よくページ処理を行えるようにアドミッション制御を行う．このアドミッション制御は，個々のページ生成処理のスループットだけを監視することでリソース競合を検出し，最適なページ処理並行度を動的に決定する．これにより，全体的なサーバのスループットを向上させることができる．一方，**セッションスケジューラ**はセッションの種類毎に用意され，セッション処理の途中失敗を防ぐためのアドミッション制御を行う．このアドミッション制御は，それぞれのページスケジューラでリクエストを待機させているキューの長さを監視することにより，将来の負荷を予測して，同時セッション数を最適に保つ．これにより，過負荷時のセッション処理性能の低下を防ぐことができる．

我々は SQS の有効性を示すために，これらのスケジューラを Tomcat 上に実装した．そして，セッションを考慮したワークロードを用いて，SQS と既存の制御法および制御を行わない場合の性能を比較する実験を行った．その結果，セッション中のページ遷移に要するユーザの思考時間，ブラウザのタイムアウト，セッションの途中失敗時の再試行を含んだ現実的なワークロードでも，SQS を用いることで過負荷時のセッション処理性能を改善できることが確認できた．

以下，2 章で従来手法とその問題点について述べ，3 章で本手法とそのプロトタイプ実装について述べる．4 章では本手法とその他の制御法を用いて行った実験について述べる．5 章で関連研究について述べ，6 章で本稿をまとめる．

2 典型的なアドミッション制御

商用サイトで用いられる Web アプリケーションサーバには常に高いセッション処理性能が要求される．セッション処理性能は単位時間あたりに処理できるセッション数であり，セッションを処理して初めて利益が得られる商用サイトには特に重要なサーバ性能の指標である．しかしながら，過負荷時においても高いセッション処理性能を保つのは容易ではない．過負荷時に Web アプリケーションサーバが高いセッション処理性能を保てるようにするには，以下に示す要件を満たす必要があると考えられる．

- **各ページ処理の高性能化**
  過負荷時にはリソースを大量に消費するページの生成処理でリソース競合が発生し，処理が滞る場合がある．セッション処理はセッションに含まれる一連のページの生成処理からなるため，過負荷時における個々のページ処理性能を向上させることが，セッション処理性能の向上につながる．
- **セッション処理の保護**
  セッション処理はセッションに含まれる一連のページを処理して初めて完了したことになる．セッション処理の成功率は各ページ処理の成功率の積となるため，セッションの長さに比例して失

敗率も上がる [2] [5]．そのため，セッション処理性能を向上させるには，過負荷時でもセッション処理が途中で失敗しないようにする必要がある．

これらの要件を満たすには，過負荷時のサーバ性能を向上させるために用いられるアドミッション制御が有用である．アドミッション制御は，サーバの状態や設定に応じてリクエスト処理の受け付けを制限する手法である．例えば，同時に処理できる最大クライアント数をサーバにあらかじめ設定するアドミッション制御がよく用いられている．この章では，このような手法の中でページ単位とセッション単位のアドミッション制御について説明し，上記要件を満たす上での問題点を述べる．

2.1 ページ単位アドミッション制御

ページ毎に処理並行度を制限するアドミッション制御でページ処理性能を向上させることによって，ページ処理の集合であるセッション処理に関しても性能向上を見込むことができる．ページ単位のアドミッション制御では，ページ毎に最適な処理並行度を決め，それを越えるリクエスト処理を破棄したり待機させたりする．しかし，現実のワークロードに対して最適な処理並行度を決定するのは容易ではない．例えば，計算リソースをそれほど必要としない軽いページでは，ファイルやネットワークへの I/O 処理が大半を占めるため，処理性能を向上させるには高い並行度で処理して，I/O 処理待ちの CPU を効率よく使用する必要がある．逆に，リソースを大量に消費する重いページでは，高い並行度で処理すると計算リソースの競合を発生させてしまう [16]．同時に様々なページへのアクセスが行われる場合には，全てのページ間の影響を考慮しつつ，計算リソースを効率よく利用できるようにページ毎に最適な処理並行度を決定しなければならない．

従来の処理並行度の決定方法には，あらかじめ測定したリソース消費量から静的に決定する手法 [2] [8] [13] や，実行時に測定した特定のリソースの消費量から動的に決定する手法 [7] [9] [10] などがある．静的な手法では実行時に同時に動作する他の Web アプリケーションの影響を考慮するのが難しい．一方，既存の動的な手法では様々なリソースを使う Web アプリケーションについては監視しなければならないリソースの数が多くなりすぎるため，測定にかかるオーバーヘッドが増加してしまう．

また，このようなアドミッション制御はセッション処理を特別に保護しないため，過負荷時にはセッション処理性能が低下してしまう．例えば，処理並行度が低く設定された重いページに対するリクエストの多くは破棄されたり，すぐに処理されずに一旦待機状態となる．待機させられたリクエストも，ブラウザのタイムアウト †1 や Web アプリケーションサーバのタイムアウト，さらには，ユーザが待ち切れずに読み込みを中止することによって破棄される可能性がある．セッションの途中のページへのリクエストが破棄されると，その時点でセッションが失敗する場合も多く，それまでのページ処理が全て無駄になる．さらに，ユーザやブラウザによってリクエストが破棄された場合には，既存の Web アプリケーションサーバの多くはそれを即座には検知できないため，クライアントがすでに破棄した無駄なリクエスト処理も行ってしまう．

2.2 セッション単位アドミッション制御

同時に処理されるセッション数を制限するアドミッション制御は，セッション処理を保護して，途中で中断されるのを防ぐことができる．セッション単位のアドミッション制御ではセッションの先頭ページへのリクエストのみが制御され，設定された同時セッション数を越える場合には，先頭ページへのリクエストは破棄されるか待機状態となる．待機状態になった後，そのリクエストがブラウザのタイムアウト等で破棄されたとしても，それによる損失は高々，先頭ページの処理だけである．

しかし，最適な同時セッション数を決定するのは容易ではない．第一に，静的に同時セッション数を決定するアドミッション制御は一般にうまく働かない．ユーザがセッションの終了手続き（ログアウト処理）を行わずにセッションを放置する場合があるからである．この場合，サーバ側ではセッションの終了を判断

†1 Safari 1.2.3 以前のデフォルト値は 60 秒，Mozilla のデフォルト値は 120 秒．

するのは難しいため，サーバにおけるセッションのタイムアウト時間 †2が経過するまで，利用可能な同時セッション数が減ってしまう．

第二に，動的に同時セッション数を決定するアドミッション制御も，Web アプリケーションに対してはうまく働かない場合がある．SBAC [5] は CPU 利用率を監視し，利用率がしきい値以下の時にのみ，余裕があると判断して新しいセッションを開始する．SBAC が対象としているのは静的なファイルへのアクセスなので，同時セッション数を変えない限り将来の負荷もほぼ一定であるという仮定が成り立つが，Web アプリケーションのように必要なリソース量がそれぞれ大きく異なる場合にはこの仮定は成り立たない．例えば，セッション中の軽いページに合わせて同時セッション数を増やしてしまうと，重いページでリソース競合が起こってセッション処理が途中で失敗する可能性がある．セッション単位アドミッション制御では，1 つのセッションを許可するとセッション中の一連のリクエスト処理を許可することになるため，同時セッション数が大きくなりすぎないように注意深く制御しなければならない．

3 Session-aware Queue Scheduling

過負荷時における Web アプリケーションサーバのセッション処理性能の低下を改善するために，我々はページ単位とセッション単位のアドミッション制御を連携させる *Session-aware Queue Scheduling* (*SQS*) を提案する．SQS は図 1 のように，動的にページ処理を制御する**ページスケジューラ**と，動的に同時セッション数を制御する**セッションスケジューラ**によって実現される．ページスケジューラはセッション内にリソースを大量に消費するページがある場合でも個々のページ処理性能を最大化する．セッションスケジューラはページスケジューラの情報を利用することで効率のよいセッション保護を実現する．

Web アプリケーションサーバに到達したリクエストは，まずセッションスケジューラによるアドミッション制御を受ける．セッションスケジューラが許可したリクエストは，次に対象ページのページスケジューラによるアドミッション制御を受ける．いずれの場合もリクエストの処理が許可されない場合は，そのリクエストは各スケジューラの保持するキューで待機状態となる．一旦セッション処理を許可されたクライアントからのリクエストは，同一セッションへのリクエストに限り，セッションスケジューラで待機状態となることはない．

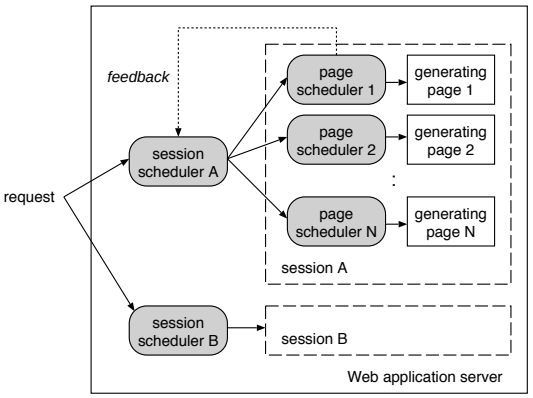

図 1　Session-aware Queue Scheduling の構成

3.1 ページスケジューラ

ページスケジューラは，計算リソースの競合を解消しページ処理性能を向上させる．ページスケジューラに到着したリクエストは，まず，ページ毎に用意されたページキューに格納される（図 2）．ページキューに格納されたリクエストは，ページ毎に設定された並行度以下で処理される．ページ毎の処理並行度は Progress-based Regulation [6] の手法を応用し

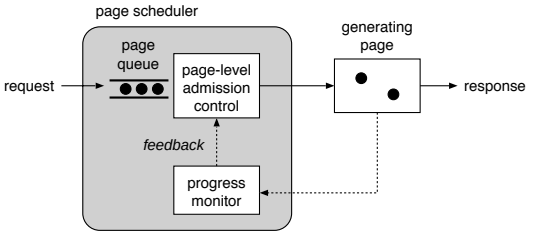

図 2　ページスケジューラによるリクエスト処理の制御

†2　Tomcat のデフォルト値は 30 分．

て動的に決定される．Progress-based Regulation は処理の進捗状況に応じてリソース割り当てを動的に変更し，協調動作させるスケジューリング手法である．ページスケジューラはページ毎の生成処理のスループットを進捗として用いる．この手法は特定のリソースを監視するわけではないため，Web アプリケーションが使うリソースには依存しない．ページスケジューラが処理並行度を上げた時にスループットが悪化すれば，並行度が高すぎるために何らかのリソース競合が発生していると判断して並行度を下げ，スループットが改善すればさらに並行度を上げる．逆に，並行度を下げた時にスループットが悪化すれば並行度を上げ，改善すればさらに並行度を下げる．この制御法は OS やサーバの用いるスケジューリングポリシーには依存せず，実行時にポリシーに合わせた最適な処理並行度を決定できる．

安定した制御を行えるようにするために，ページスケジューラは過去のスループットの変遷履歴を統計処理してページ処理並行度を決定する．Web アプリケーションのスループットは，リクエスト時に与えられるパラメータによる処理内容の違いやサーバ計算機上で動作する他のプロセスの影響によって変動することがある．この影響にページスケジューラが即座に反応してしまわないように，ページ処理並行度の決定には直前のスループットだけではなく過去の測定データも利用する．

3.2 セッションスケジューラ

セッションスケジューラは同時に実行されるセッション数を最大化する一方で，セッション処理の途中失敗を防ぐ．セッションスケジューラはセッションに含まれる全てのページスケジューラのページキューを監視することにより，セッション全体を考慮しながら動的に同時セッション数を決定する．ページスケジューラにおける待機リクエストは将来必ず処理されるので，ページキューの長さは将来の負荷を表わす．そのため，すべてのページキューの平均長を見ることで比較的正確に将来の負荷を予測することができると考えられる．例えば，ページキューの平均長が長くなるとセッションの途中失敗が発生しやすくなるので，

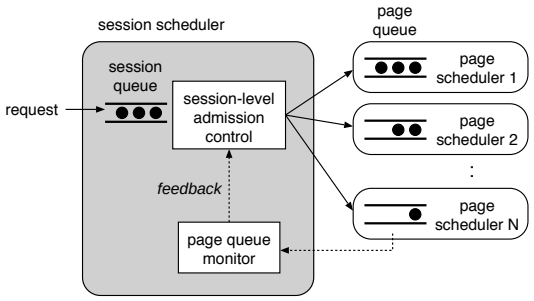

図 3 セッションスケジューラによるセッション処理の制御

同時セッション数を減らすことでページキューを短くする．逆に，ページキューの平均長が短い場合は，リソースが有効活用できていない状態を意味するため，同時セッション数を増やす．

セッションスケジューラをページスケジューラと組み合わせるもう 1 つの利点は，より柔軟にアドミッション制御を行うことができることである．セッション単位アドミッション制御のみを用いた場合と違い，同時セッション数を増やしすぎたとしても，セッションが途中失敗しない範囲ならば，ページスケジューラによって影響が吸収される．各ページ処理でリソース競合が起きるようならば，ページスケジューラがリクエストを待機させ，処理並行度を最適に保つからである．そのため，セッションスケジューラは同時セッション数を増やす方向の制御を行いやすくなる．

セッションスケジューラは同時セッション数を変更するために，セッション処理の投入間隔を調節する．セッションスケジューラに到着したセッションの先頭ページへのリクエストはまず，セッションキューに入れられる（図 3）．そして，セッション処理の投入に適度な遅延を持たせ，投入間隔を調節することで同時セッション数を制御する．投入間隔を大きくしていくと，次第に同時セッション数を減らすことができる．逆に，投入間隔を小さくすると，次第に同時セッション数を増やすことができる．投入間隔の調整で同時セッション数を変更するのは，セッションが同時に投入されるのを防ぎ，サーバへの負荷の均一化が期待できるからである．

さらに，セッションキューには受け付けられるリクエスト数の上限を設定できる．もし，セッションスケ

ジューラがリクエストを無制限に受け付け，それを待機させると，実際の処理が開始される前にクライアントのタイムアウト等によりリクエストが破棄される可能性が高まる．すると，サーバがそれを検知できずに既に破棄されたリクエストがセッションキューに残ったままになってしまう．一定時間以内に処理が行われないと判断できるリクエストを待機させず，エラーページを返信するなどして即座に破棄することで，リクエストの過度な蓄積を回避できる．

3.3 プロトタイプ実装

SQS の有効性を実証するために Tomcat [1] 上にこれらのスケジューラの実装を行った．我々は `ControlServlet` というクラスを作成してページスケジューラとセッションスケジューラを実装した．この `ControlServlet` を拡張して Servlet を実装することにより実行が制御される．`ControlServlet` はクライアントからのリクエストをインターセプトしており，リクエストが Web アプリケーションサーバに到達すると，まず `ControlServlet` が呼ばれ，その内部で実際の処理を行う Servlet のメソッドが呼び出される．`ControlServlet` はページ毎の処理並行度（$maxRequest$）とセッション処理の投入間隔（$sessionInterval$）を保持し，その値は実行時に以下のように決定される．

- **$maxRequest$ の決定**
  $maxRequest$ を増減させた時にある区間におけるスループットが継続的に向上するようならば $maxRequest$ の値を同じ方向にさらに増減させ，低下するようならば逆の方向に増減させる．増分は経験的に最も安定して制御ができた 1 に設定している．
- **$sessionInterval$ の決定**
  ページキューの平均長がしきい値以上の時には $sessionInterval$ の値を増加させ，しきい値以下の時には $sessionInterval$ の値を減少させる．しきい値や増分はセッションの構成やサーバ性能に合わせてカスタマイズ可能である．現在のところ，過去の履歴は使用していない．

4 実験

SQS による性能改善を示すためにいくつかの現実的なシナリオに基づいた実験を行った．実験に使用した Web アプリケーションは，商用サイトの商品購入アプリケーションに見立てたもので以下のような Servlet 群から構成される．

- 1 ページ目: ログイン（セッションの開始）
- 2〜4 ページ目: 商品の閲覧
- 5 ページ目: 商品検索・表示
- 6 ページ目: 商品を購入
- 7 ページ目: ログアウト（セッションの終了）

5 ページ目は商品情報を記述した XML ファイル（約 650KB）を検索する処理を含む重いページで，処理時間は 400ms 程度である．この処理は CPU 以外にメモリも大量に使用し，Java VM の GC を引き起こす．それ以外のページは軽いページであり，処理時間は 30ms 程度である．

セッション処理性能を比較するために，以下の 4 つの制御法に対して実験を行った．

- SQS
- ページ単位アドミッション制御
- セッション単位アドミッション制御
- 制御なし

ページ単位アドミッション制御は SQS のページスケジューラのみを取り出したものを使用した．セッション単位アドミッション制御では与えたワークロードでセッション処理性能が最大になる同時セッション数を静的に設定し，同時セッション数を越えるリクエストを破棄するようにした．これはページ毎にリソース消費量が大きく異なるワークロードでうまく働く動的なセッション単位アドミッション制御が知られていないためである．

実験には，Intel Xeon 2.40GHz の CPU を 2 つ，2GB のメモリ，ギガビットイーサネットを持つサーバ計算機を用い，OS には Linux 2.4.20，Web アプリケーションサーバには Tomcat 3.3.1 [1] を用いた．また，クライアントには Intel Xeon 3.06GHzCPU を 2 つ，2GB のメモリ，ギガビットイーサネットを持つ計算機を用い，OS には Linux 2.6.8 を用いた．

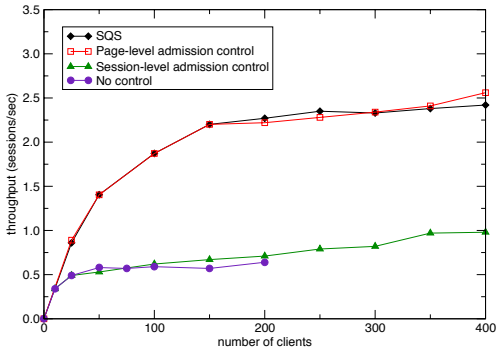

図 4 セッション処理性能
（タイムアウトなし，思考時間 3 秒）

4.1 ページスケジューラによる性能改善

まず，ページスケジューラによるリソース競合の解消を確認するための実験を行った．この実験では，クライアント数を 1 から 400 まで変えて，1 クライアントにつき 1 セッションを実行させた．そして，セッションが開始されてから全てのセッションが完了するまでに要する総処理時間を測定した．ページスケジューラの効果だけを確認するために，ブラウザのタイムアウトは考慮せず，セッションが途中で失敗しないようにした．一方，ユーザがセッション内のページを遷移する間に要する思考時間は 3 秒とした．セッション単位アドミッション制御では性能が最大となった同時セッション数（75）を用いた．

図 4 に実行された総セッション数を総処理時間で割って算出したセッション処理性能を示す †3．SQS とページ単位アドミッション制御は各ページの処理並行度を制御してリソース競合を解消するため，最もセッション処理性能が高くなった．SQS はセッションスケジューラの処理も行っているが，ページスケジューラの処理のみを行うページ単位アドミッション制御とほぼ同等の性能であった．このことから，セッションスケジューラのオーバヘッドは十分小さいことが分かる．一方，セッション単位アドミッション制御と制御なしの場合，リソース競合が発生してセッション処理性能が低下した．

†3 途中でデータが途切れているものは，それ以上のクライアント数では 500 秒以上にわたって応答がなくなり，全てのセッションが完了できなかったことを示している．以下の実験でも同様．

4.2 セッションスケジューラによる性能改善

セッションスケジューラによる性能改善を示すために，4.1 節と同様の実験をブラウザのタイムアウトを発生させて行った．タイムアウトは 60 秒とし，タイムアウトしたクライアントはそのページへのリクエストを破棄し，セッションの第 2 ページから再試行するものとした．セッション単位アドミッション制御では性能が最大となった同時セッション数（35）を用いた．

図 5 より，ブラウザのタイムアウトを考慮した場合でも SQS だけが高いセッション処理性能を維持できていることが分かる．ページ単位アドミッション制御では，クライアント数が 150 を越えたあたりからセッション処理性能を維持できなくなった．セッション単位アドミッション制御でもセッションを保護することによりセッション処理性能を維持できてはいるが，SQS と比べると全体的に性能が低く抑えられていた．また，制御なしの場合，約 25 クライアントで処理性能が頭打ちになった．

次に，制御なしの場合に測定できた最大のクライアント数（100）および 400 クライアントの場合に，セッションの先頭ページへのリクエストを送ってからそのセッション処理を完了するまでの時間の分布を調べた．図 6 に経過時間と完了した総セッション数の関係を表わす．このグラフより，SQS とページ単位アドミッション制御は他の制御法よりセッション完了までの時間が短いことが分かる．一方，図 7 より，負荷が高まった時でも SQS はセッション完了までの時間を短く保つことができていることが分かる．

セッション処理性能差の原因

セッション処理性能の差異を生み出す原因を分析するために，それぞれの手法における途中失敗数を図 8 に示す．ただし，先頭ページで待機させられることなく即座に破棄されたリクエストは含めていない．セッション単位アドミッション制御では途中失敗は全く起こっておらず，SQS でも途中失敗を抑えることができている．図 8 と図 5 より，途中失敗を抑えることができれば，過負荷時においてもセッション処理性能

表 1　セッション内の各ページでの失敗数の内訳 （100 クライアント/400 クライアント）

ページ番号	1	2	3	4	5	6	7
SQS	0 / 91	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0	0 / 0
ページ単位アドミッション制御	0 / 0	0 / 0	0 / 0	0 / 0	0 / 1179	0 / 0	0 / 0
制御なし	0 / -	0 / -	0 / -	0 / -	317 / -	0 / -	0 / -

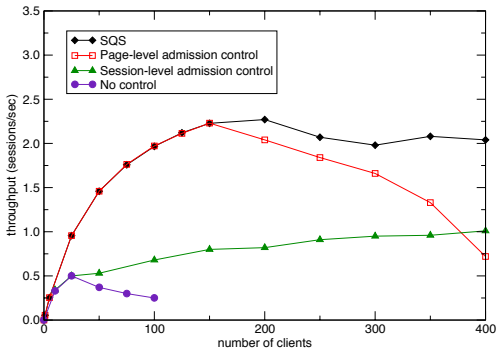

図 5　セッション処理性能
（タイムアウト 60 秒，思考時間 3 秒）

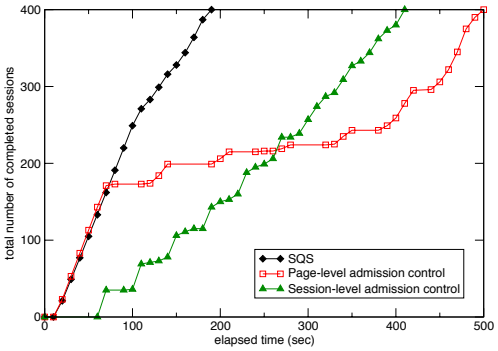

図 7　セッション完了時間の分布（400 クライアント）

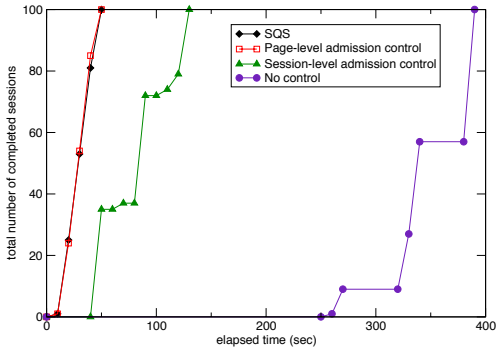

図 6　セッション完了時間の分布（100 クライアント）

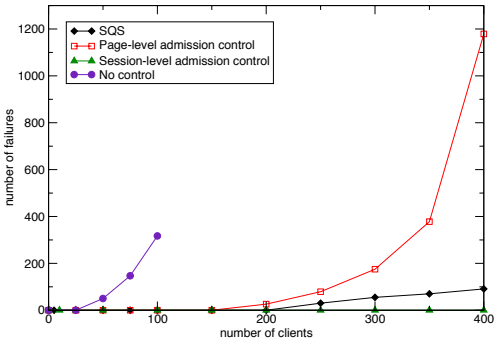

図 8　セッション途中失敗数
（タイムアウト 60 秒，思考時間 3 秒）

を維持できることが分かる．

さらに，SQS，ページ単位アドミッション制御，制御なしの 3 手法において，セッション内の各ページでの失敗数の内訳を表 1 に示す．この内訳には，制御なしの場合に測定できた最大のクライアント数（100）および 400 クライアントの場合を抽出した．SQS 以外では，リソースを多く消費するページ（5 ページ目）でタイムアウトによる失敗が多く発生していた．一方，SQS ではページ単位アドミッション制御と比べて途中失敗が低く抑えられていたことに加え，全てのタイムアウトが先頭ページで起こっていたため，セッションを途中まで処理してから失敗することがなかった．

スケジューラの挙動

セッションスケジューラとページスケジューラの挙動を示すために，ページキューの平均長に応じたセッション処理の投入間隔（$sessionInterval$）の変動を図 9 に示し，ページ毎の処理並行度（$maxRequest$）の変動を図 10 に示す（ページ 2〜4，6，7 はページ 1 の結果とほぼ同様の結果であったため，図を見やす

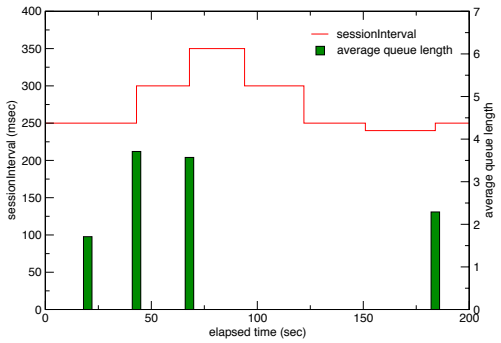

図 9 ページキューに応じた *sessionInterval* の変動

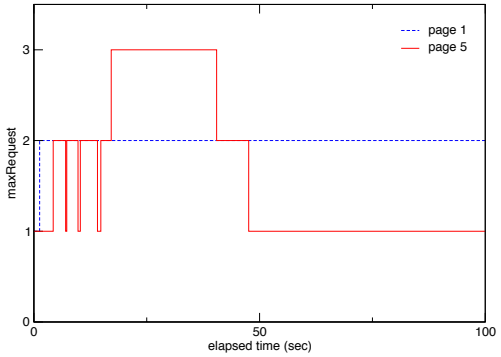

図 10 各ページの *maxRequest* の変動

くするために省略）．測定データは 400 クライアントの時のものである．グラフから，ページキューが長くなり待機リクエスト数が増えた時は，セッションスケジューラが *sessionInterval* を増加させて同時セッション数を減らしていることが分かる．逆に，ページキューが短くなると *sessionInterval* は減少させられている．一方，ページスケジューラはセッション中の最も重いページの *maxRequest* を頻繁に変動させて最適なページ処理並行度を探し，一定時間後に安定していることが分かる．

4.3 クライアントの挙動による影響

クライアントの挙動がセッション処理性能に与える影響を調べるために，タイムアウトと思考時間をそれぞれ変更して同様の実験を行った．

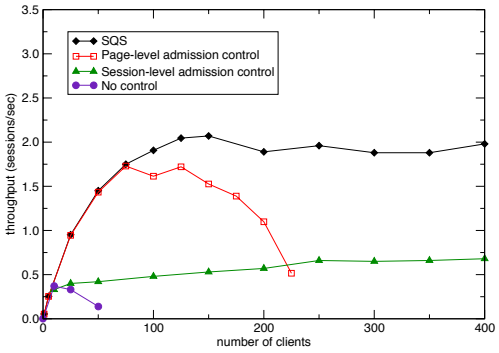

図 11 セッション処理性能
（タイムアウト 30 秒，思考時間 3 秒）

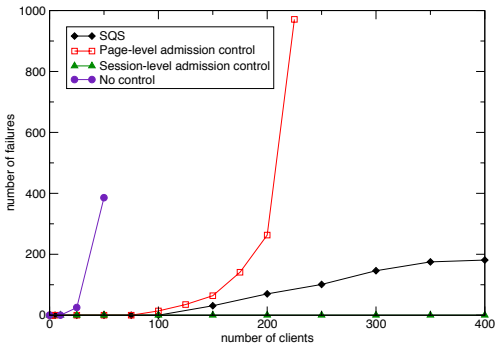

図 12 セッション途中失敗数
（タイムアウト 30 秒，思考時間 3 秒）

タイムアウトの影響

タイムアウトの長さがセッション処理性能に与える影響を調べるために，思考時間は 3 秒のままで，タイムアウトを 60 秒から 30 秒に変更して同様の実験を行った．セッション単位アドミッション制御では性能が最大となった同時セッション数（15）を用いた．図 11 より，SQS のセッション処理性能は，タイムアウトが 60 秒の時の結果（図 5）と比べて少し低下しているが，過負荷時にも性能を保てていた．これはタイムアウトが発生しないようにセッションスケジューラがセッションを保護しているためである．一方，ページ単位アドミッション制御や制御なしの場合は，タイムアウトが短くなったことにより，より少ないクライアント数でもセッション処理性能が低下するようになった．これは，図 12 に示されるように，タイムアウトが 60 秒の時（図 8）と比べてセッション

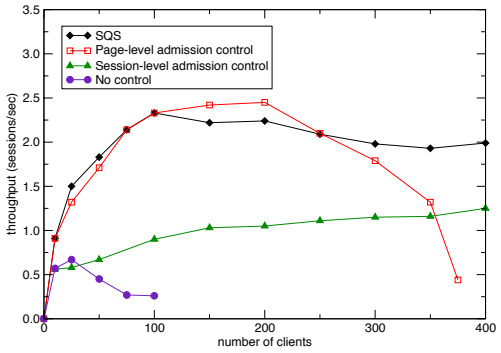

図 13　セッション処理性能
（タイムアウト 60 秒，思考時間 1 秒）

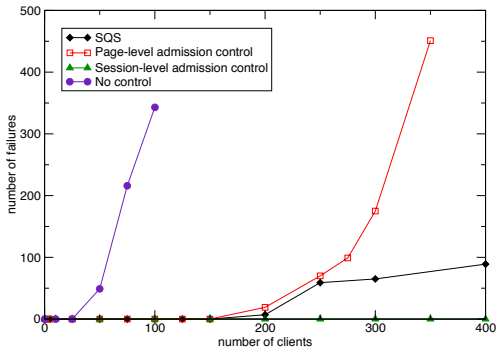

図 14　セッション途中失敗数
（タイムアウト 60 秒，思考時間 1 秒）

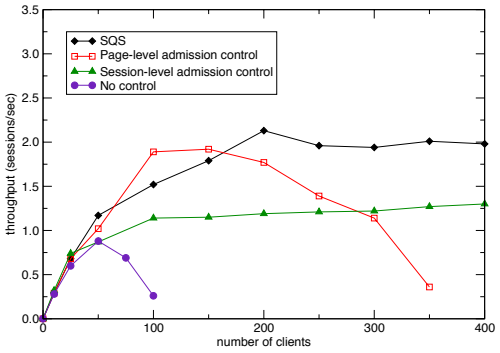

図 15　セッション処理性能
（タイムアウト可変，平均思考時間 3 秒）

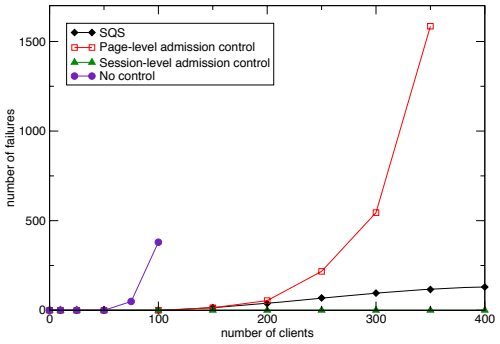

図 16　セッション途中失敗数
（タイムアウト可変，平均思考時間 3 秒）

が途中で失敗しやすくなったためである．セッション単位アドミッション制御に関しては，タイムアウトしないように同時セッション数を減らしたため，性能が低下している．

思考時間の影響

ユーザの思考時間がセッション処理性能に与える影響を測定するために，タイムアウトは 60 秒のままで，思考時間を 3 秒から 1 秒へ変更して同様の実験を行った．セッション単位アドミッション制御では性能が最大となった同時セッション数（35）を用いた．図 13 に示されるように，SQS のセッション処理性能は思考時間が 3 秒の場合（図 5）と比べて，あまり変わらなかった．また，セッションの途中失敗数に関しても，図 14 に示されるように，制御なしの場合以外は，思考時間が 3 秒の場合（図 8）とあまり変わらなかった．これらより，思考時間は過負荷時のセッション処理性能には大きな影響を与えないと考えられる．

現実的なクライアントの挙動の影響

より現実的なクライアントの挙動を用いた時のセッション処理性能を測定するために，セッション毎にタイムアウトを 50%の確率で 30 秒または 60 秒に設定した．また，ユーザの思考時間を平均 3 秒の指数分布に従うように設定した．セッション単位アドミッション制御では性能が最大となった同時セッション数（35）を用いた．結果は図 15，図 16 に示されるようになり，思考時間を 1 秒にした時の結果（図 13，図 14）とよく似ていた．この結果より，タイムアウトおよび思考時間がばらつく場合でも，SQS はほとんどの場合に最も高いセッション処理性能を達成できることが分かる．

4.4 考察

我々の実験結果より，ページ単位とセッション単位のアドミッション制御を組み合わせることはセッション処理性能を向上させるのに有用であることが分かる．ページ単位アドミッション制御だけでも総セッション数がある程度までなら性能を向上させることができている．しかし，サーバが過負荷状態になると，急激に性能が低下してしまっている．一方，セッション単位アドミッション制御だけを用いるとサーバが過負荷になってもほぼ一定の性能を保つことができている．その反面，セッションを保護するためにサーバリソースを有効に活用できておらず，全体の性能は高くない．それに対して，SQS では過負荷時でもセッションスケジューラによって同時セッション数を抑えることによってセッションの失敗を防ぎ，ページスケジューラによって各ページの性能を向上させることができている．

5 関連研究

SQS と同様に，ページ単位とセッション単位のアドミッション制御を併用する手法が提案されている [3]．この手法では，セッション単位アドミッション制御とページ単位アドミッション制御の 2 段階の制御を行っているが，それらを連携させる機構はない．そのため，セッション単位アドミッション制御のみを用いる場合と同様に性能が低く抑えられてしまう．加えて，この手法の有効性はシミュレーションで検証されているだけである．

SBAC [5] は CPU 利用率に基づいて将来の負荷を予測し，セッション単位アドミッション制御を行う．それに対して，SQS では将来の負荷を表わしているページキュー内の待機リクエストの数を基に予測するため，より正確に予測を行えると考えられる．また，SBAC でもクライアントのタイムアウトやユーザの思考時間が考慮されているが，これらの時間を変えることによる性能の変化については調べられていない．この手法に関しても，有効性はシミュレーションでしか検証されていない．

セッションの途中失敗を減らすために DWFS というスケジューリングが提案されている [4]．DWFS は各ページ処理に対して動的に優先度をつけ，セッションをより多く完了できそうなページ処理を優先する．この手法はブラウザのタイムアウトではなくサーバでのセッションタイムアウトを対象としているため，我々の手法と組み合わせることにより，さらにセッション処理性能を向上させられる可能性がある．

ページ単位でもセッション単位でもない制御として，SEDA アーキテクチャを用いたアドミッション制御の研究も行われている [14]．SEDA [15] は 1 つのリクエスト処理を細かいステージに分割してリソース管理をより細かく行う．各ステージでアドミッション制御を行うことで，過負荷時にも各ステージの性能を維持することができる．SEDA アーキテクチャを用いるためには Web アプリケーションの大幅な変更が必要となるが，我々の手法と組み合わせることも可能であると考えられる．

待ち行列理論を用いてウェブサーバの性能モデルを構築する研究も行われている [11] [12]．性能モデルが構築できれば，よりよいアドミッション制御が可能になる．しかし，我々の用いたワークロードではメモリを大量に使用することによって Java VM で GC が引き起こされ，Web アプリケーション本体以外の要因も考慮する必要がある．そのため，現状では正確な性能モデルを構築するのは難しいと考えられる．

6 まとめ

本稿では，過負荷時における Web アプリケーションの性能を改善するためにページスケジューラとセッションスケジューラを連携させる Session-aware Queue Scheduling（SQS）を提案した．ページスケジューラは各ページの生成処理における計算リソースの競合を回避して性能を改善するために，ページ毎の最適な処理並行度を動的に決定する．セッションスケジューラは各ページスケジューラの持つページキューを監視し，最適な同時セッション数を決定する．我々は Tomcat 上にこれらのスケジューラを実装し，セッションを考慮した現実的なワークロードを用いて SQS の有効性を確かめた．その結果，ページスケジューラによって全体的なセッション処理性能が向上し，セッションスケジューラによってセッションの途

中失敗による性能低下を抑えられることが分かった．

今後の課題としては，さらなる性能向上を目指して各スケジューラのアルゴリズムを洗練させることや，既に提案されている他の手法と組み合わせてみることが挙げられる．また，データベースを用いたより大規模な Web アプリケーションに対して本手法を適用し，その有効性を確かめることも挙げられる．

参 考 文 献

[1] Apache Software Foundation: Apache Tomcat, `http://tomcat.apache.org/`.

[2] Bhatti, N. and Friedrich, R.: Web Server Support for Tiered Services, *IEEE Network*, Vol. 13,No. 5(1999), pp. 64–71.

[3] Carlstrom, J. and Rom, R.: Application-aware Admission Control and Scheduling in Web Servers, in *Proceedings of the Infocom 2002*, 2002, pp. 506–514.

[4] Chen, H. and Mohapatra, P.: Session-based Overload Control in QoS-aware Web Servers, in *Proceedings of the Infocom 2002*, 2002, pp. 516–524.

[5] Cherkasova, L. and Phaal, P.: Session Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites, *IEEE Transactions on Computers*, Vol. 51,No. 6(2002), pp. 669–685.

[6] Douceur, J. and Bolosky, W.: Progress-based Regulation of Low-importance Processes, in *Proceedings of the 17th Symposium on Operating Systems Principles*, 1999, pp. 247–260.

[7] Elnikety, S., Nahum, E., Tracey, J. and Zwaenepoel, W.: A Method for Transparent Admission Control and Request Scheduling in E-commerce Web Sites, in *Proceedings of the 13th International Conference on World Wide Web*, 2004, pp. 276–286.

[8] Li, K. and Jamin, S.: A Measurement-based Admission-controlled Web Server, in *Proceedings of the Infocom 2000*, 2000, pp. 651–659.

[9] McWherter, D., Schroeder, B., Ailamaki, A. and Harchol-Balter, M.: Priority Mechanisms for OLTP and Transactional Web Applications, in *Proceedings of the 20th International Conference on Data Engineering*, 2004, pp. 535–546.

[10] Shen, K., Tang, H., Yang, T. and Chu, L.: Integrated Resource Management for Cluster-based Internet Services, in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002, pp. 225–238.

[11] Slothouber, L.: A Model of Web Server Performance, in *Proceedings of the 5th International Conference on World Wide Web*, 1996.

[12] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M. and Tantawi, A.: An Analytical Model for Multi-tier Internet Services and Its Applications, in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, 2005, pp. 291–302.

[13] Voigt, T., Tewari, R., Freimuth, D. and Mehra, A.: Kernel Mechanisms for Service Differentiation in Overloaded Web Servers, in *Proceedings of the Annual Technical Conference*, 2001, pp. 189–202.

[14] Welsh, M. and Culler, D.: Adaptive Overload Control for Busy Internet Servers, in *Proceedings of the 4th Conference on Internet Technologies and Systems*, 2003.

[15] Welsh, M., Culler, D. and Brewer, E.: SEDA: An Architecture for Well-conditioned, Scalable Internet Services, in *Proceedings of the 18th Symposium on Operating Systems Principles*, 2001, pp. 230–243.

[16] 松沼正浩, 千葉滋, 佐藤芳樹, 光来健一: 過負荷時における Web アプリケーションの性能劣化を改善する Page-level Queue Scheduling, 第 7 回プログラミングおよび応用のシステムに関するワークショップ オンライン論文集, 2004.