# 過負荷時のWebアプリケーションの性能劣化を改善するSession-level Queue Scheduling

松沼正浩　日比野秀章　佐藤芳樹　光来健一　千葉滋

本研究では、アクセスが集中して過負荷状態に陥った Web アプリケーションサーバの性能低下を防ぐための Session-level Queue Scheduling を提案する。Session-level Queue Scheduling は実行中の Web アプリケーションの進捗を監視する事でリソース競合を検出し、競合解消のためにリクエスト処理に動的な制限をかけるアドミッションコントロールを行う。既存の多くのアドミッションコントロールはページまたはサーバ全体を単位として制御を行うが、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、Session-level Queue Scheduling 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組み合わせる。我々はこれらのスケジューラを Tomcat 上に実装し、セッションのタイムアウトやセッション中のユーザのシンクタイム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、Session-level Queue Scheduling を用いた場合には、過負荷時の性能が向上することが確認できた。

## 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 (カート利用)、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。その原因の一つは、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。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、並列化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 などにより実装されるような大量のリソースを消費するページ生成処理を単純に並列処理した場合、リソース競合が発生して処理性能の低下を引き起こす。このようなリソース競合を解消するためには、大量にリソースを消費するページの処理を制御する必要がある。

もう一つの原因は、各ページの処理性能が低下した結果、セッション処理が中断され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。

Masahiro Matsunuma, Hideaki Hibino, Yoshiki Sato, Kenichi Kourai, Shigeru Chiba, 東京工業大学, Tokyo Institute of Technology

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、ユーザが待ちき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、セッションの中断を防ぐ必要がある。

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組み合わせた Session-level Queue Scheduling 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コントロールを行う。ページスケジューラ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、リクエストの最適な並列処理数を動的に決定する。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コントロールを行う。セッションスケジューラはそれぞれのページスケジューラで待たされているリクエスト数を監視することにより、同時に処理されるセッション数を最適に保つ。

我々は Session-level Queue Scheduling 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。我々の実装では、専用のクラスを継承するように既存の Servlet を書き換えるだけで、Session-level Queue Scheduling が適用されるようにすることができる。この実装を用いて既存の Servlet と性能を比較する実験を行い、セッションのタイムアウトやセッション中の思考時間などを考慮した場合でも、Session-level Queue Scheduling を用いることで過負荷時のセッション処理性能が改善することを確認した。

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 章で関連研究を紹介し、6 章で本稿をまとめる。

## 2 典型的なアドミッションコントロール

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成るため、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 [3] [1]。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コントロールが有用であると考えられる。アドミッションコントロールは、特定のリクエストを識別し、サーバの状態や設定に応じてリクエスト処理の制御を行う手法である。例えば、並列処理を許可されるリクエストの数をサーバにあらかじめ設定するアドミッションコントロールが知られている。この章では、このような手法の中でページ単位 [5] [13] とセッション単位 [3] [2] のアドミッションコントロールについて説明し、上記要件を満たす上での問題点を述べる。

**ページ単位アドミッションコントロール**

ページ毎に並列度を制限するアドミッションコントロール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コントロールでは、ページ毎に最適な処理数を決め、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ことができる。たとえば、計算リソースをそれほど必要としない軽いページは、高い並列度で処理することで I/O 処理待ちの CPU を効率よく使用し、処理性能を向上させることができる。逆に、リソースを大量に消費する重いページでは、高い並列度は計算リソースの不足・競合を発生させる [13]。軽いページの処理並列度を上げ、重いページの処理並列度を下げるアドミッションコントロールは、計算リソースを効率よく利用することを可能にする。並列度の決定方法には、アプリケーションプログラマによる予測 [2] や、実行時に消費リソースを測定して行うもの [5] [13] などがある。

しかし、このようなアドミッションコントロール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。たとえば、並列度を低く設定された重いページに対するリクエストは、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は、場合によって Web サーバのタイムアウト †1、ブラウザのタイムアウト †2、さらにユーザが待ちきれずに読み込みを中止する事によって破棄されてしまう。セッション処理が中断されると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検知できず、破棄されたリクエストの処理も行ってしまう。このような無駄な処理の分だけセッション処理性能が低下することになる。

**セッション単位アドミッションコントロール**

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コントロールは、セッション処理が途中で中断されないように保護することができる。例えば、セッション処理のボトルネックになっている最も重いページにあわせて、処理するセッションの数を制限すれば、セッションを構成するどのページでも並列処理による競合は起きなくなる。このような方法では、リクエストはセッションの先頭ページで制御され、決められた並列度を越えるリクエストは先頭ページで破棄されるか待機状態となる。待機状態になった後、リクエストがタイムアウト等で破棄されたとしても、それによる損失はたかだか先頭ページでリクエストが破棄された後でも無駄に行なわれてしまう処理だけである。

しかし、保護されたセッション処理が特定のクライアントによる計算リソースの占有を引き起こすため、サーバ全体の性能が悪化するという問題がある。セッション処理を保護するためには、保護されたクライアントが次のページにリクエストを出すまでの思考時間 (シンクタイム) でさえ他のクライアントのセッション処理を受け付けずにリソースを占有し続ける必要がある。もしシンクタイム中に他のクライアントの処理を開始してしまうと、保護されたクライアントを優先的に処理するのが難しくなる。また、クライアントがセッションの終了手続き (ログアウト処理) を行なわずにセッションを放置することも多い。そのため、単純なセッション単位のアドミッションコントロールでは、リソースが長時間開放されないことになる。なぜなら、サーバ側ではセッションの終了をログアウトなしで判断するのは難しく、セッションのタイムアウト時間 †3が経過するまで、セッション処理を保護し続けなければならないからである。

## 3 Session-level 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コントロールを組み合わせた Session-level Queue Scheduling を提案する。本手法は、動的にページ処理を制御するページスケジューラと、ページスケジューラを監視しながらセッション処理を保護するセッションス

†1 Apache 1.3 の初期値は 5 分
†2 Safari 1.2.3 以前の初期値は 60 秒
†3 Tomcat の初期値は 30 分

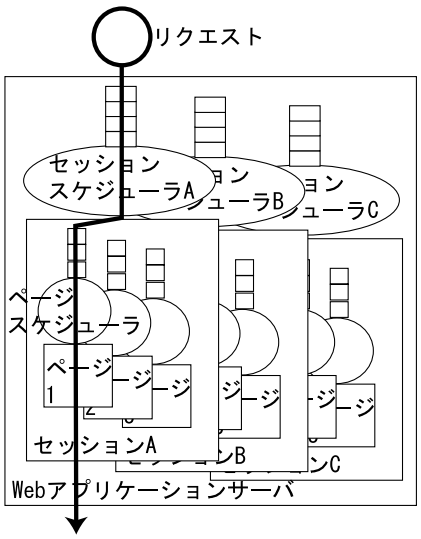


**図 1 Session-level Queue Scheduling の概要**

ケジューラによって実現される。図 1 に示すように、Web サーバに到達したリクエストは、最初にセッションスケジューラによるアドミッションコントロールを受ける。セッションスケジューラが許可したリクエストは、続いて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コントロールを受ける。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。セッション処理を一度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。

### 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キューに格納される (図 2)。それぞれの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列度で処理が行なわれる。ページ毎の並列処理数は、ページ生成処理のスループットから 、Progress-based Regulation [4] の手法に基づい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するスケジューリング手法である。ページ毎のスループットは、他のアプリケーションによって使用されていないサーバ計算機の有効リソース量に依存して変動する。ページスケジューラは、ページ生成処理のスループットを測定し、進捗が悪化した時には並列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、並列度を下げる調整を行う。逆に、進捗が悪化しなければ、さらに並列度を上げる調整を行う。

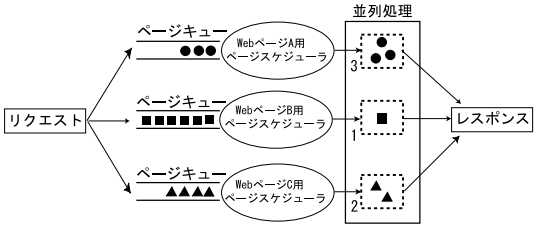


**図 2 ページスケジューラによるリクエスト処理の制御**

また、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並列度を決定する。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、並列度の決定には単純なスループットの測定・比較ではなく過去数回に渡っての測定データを利用する。

### 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のキューを監視することで、セッションの並列度を決定する。例えば、ページスケジューラにおける待機リクエスト数が増加してページキューが長くなると、セッション途中において失敗がおきやすくなるので同時セッション数を減らす。逆にページキューが短い場合は、最適な並列度に達し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、セッション処理の開始間隔を調節する。セッションスケジューラに到着したリクエストはまず

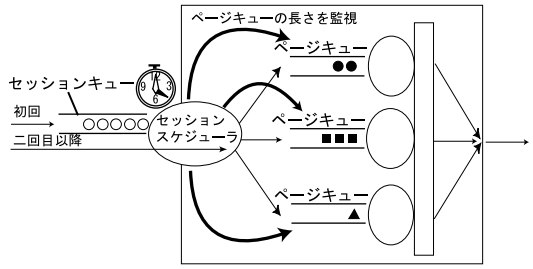


**図 3 セッションスケジューラによるリクエスト処理の制御**

セッションキューに入れられる。そして、セッション処理の許可に適度な遅延を持たせ、その遅延時間を調節することで、同時セッション数をコントロールする。遅延時間を大きくしていくと、同時セッション数を減少させられ、その結果、ページキューを短くできる。逆に遅延時間を小さくすると、同時セッション数を増加させられ、その結果、ページキューを長くできる。

また、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てしまうかもしれない。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すぐに破棄[†4]することで、リクエストの過度な蓄積を回避する。

### 3.3 プロトタイプ実装

Session-level Queue Scheduling の有効性を実証するために Tomcat [6] 上にこれらのスケジューラの実装を行った。ページスケジューラとセッションスケジューラが実装された ControlServlet クラスを提供しており、各 Servlet プログラムは ControlServlet クラスを継承することで実行を制御される。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、はじめに ControlServlet が呼ばれ、その内部でアプリケーション Servlet が呼び出される。ControlServlet はページの処理並列度 (maxThread) と、セッション処理の開始間隔 (sessionInterval) を保持し、その値は実行時に統計処理で決定される。

#### 3.3.1 maxThread の決定

maxThread は、ページ処理並列度を変更した時のスループットの変動を計測して決定される。ある時点において maxThread の値を増加・減少させた時にその効果がスループットの計測により確認できる場合、その変更を行い続けるという戦略をとる。ここでは、ある時点 $T$ で実行中の maxThread の値を MaxThread($T$) とし、それよりも一つ前に実行した時の maxThread の値を MaxThread($T$-1) とし、それぞれで計測したスループットの値 (処理リクエスト数/総処理時間) を Throughput($T$)、Throughput($T$-1) とする。今回計測したスループットとその時の maxThread が

$$\frac{\texttt{Throughput}(T) - \texttt{Throughput}(T-1)}{\texttt{MaxThread}(T) - \texttt{MaxThread}(T-1)} \geq 0$$

の関係を満たす時、つまり maxThread が大きい方がスループットが良いと判断された時、

$$\texttt{MaxThread}(T+1) = \texttt{MaxThread}(T) + \Delta k$$

となる。一方、

$$\frac{\texttt{Throughput}(T) - \texttt{Throughput}(T-1)}{\texttt{MaxThread}(T) - \texttt{MaxThread}(T-1)} < 0$$

の関係を満たす時、つまり maxThread の小さい方がスループットが良いと判断された時、

$$\texttt{MaxThread}(T+1) = \texttt{MaxThread}(T) - \Delta k$$

となる。なお $\Delta k$ の初期値は 1 に設定しているが、アプリケーション Servlet から任意に設定が可能である。

#### 3.3.2 sessionInterval の決定

sessionInterval は、ページ毎に用意したキューの長さを測定して決定している。次に具体的な sessionInterval の決定アルゴリズムを示す。ある時点におけるページキューの長さの平均が一定値 (maxQueue) 以上である場合には、sessionInterval の値を増加させる。逆にページキューの長さの平均が一定値 (minQueue)

†4 エラーページを返信するなど

以下である場合には sessionInterval の値を減少させる。増分、maxQueue、minQueue の値はセッションの種類毎に任意に設定可能である。

## 4 実験

Session-level Queue Scheduling 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
  (約 650KB の XML ファイルをパースして探索)
- 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)

1 リクエストに対する処理時間は 5 ページ目のみ 400ms で、それ以外は 30ms 程度である。セッション処理性能を比較するために、

(a) 制御無し
(b) ページ単位アドミッションコントロール [13]
(c) セッション単位アドミッションコントロール (セッション並列度:30)
(d) 重いページにあわせたセッション単位アドミッションコントロール (セッション並列度:1)
(e) Session-level Queue Scheduling

の 5 つの手法を用いた。

実験には、Intel Xeon 2.40GHz × 2 の CPU、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 2.4.2、Web アプリケーションサーバには Tomcat 3.3.1 [6] を動作させた。また、Intel Xeon 3.06GHz × 2 の CPU、2GB のメモリ、ギガビットイーサネットを持つクライアント計算機を用い、OS には Linux 2.6.8 を動作させた。

### 4.1 ページスケジューラによる性能改善

はじめに、ページ処理並列度の増加が引き起こすリソース競合とページスケジューラによるその競合解消を確認するための実験を行なった。実験は、同時にアクセスするクライアント数を 1 から 400 まで変えて同時にリクエストを送りはじめ、全クライアントのセッションが完了するまでの総処理時間を測定した。クライアントがセッション内のページを遷移する間に要するシンクタイムは 3 秒とした。また、ページスケジューラの効果だけを確認するために、クライアントは無制限にレスポンスを待つものとした。それによりセッションは失敗せずに必ず保護される。

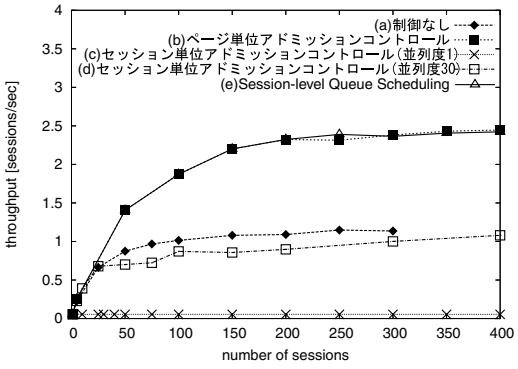


**図 4 セッション処理性能 (タイムアウト無し、シンクタイム 3 秒)**

図 4 に総処理時間から算出した、同時セッション数に対するセッション処理性能を示す †5。(a) の制御無しと (d) のセッション単位アドミッションコントロール (並列度 30) を用いた場合、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。一方、(b) のページ単位アドミッションコントロールと (e) の Session-level Queue Scheduling は、各ページの並列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。セッションが失敗しない場合、その二つの手法は同じ挙動を示すため同程度の性能を得た。この二つからセッションスケジューラのオーバーヘッドは十分小さいことが分かる。なお、(c) のセッション単位アドミッションコントロール (並列度 1) で著しく処理性能が低下したのは、シンクタイムを含んだセッション処理が逐次的に行なわれたためである。

†5 途中でデータが途切れているものは、それ以上のセッション数では 500 秒以上にわたりレスポンスが得られなかったことを示している。以下の実験でも同様。

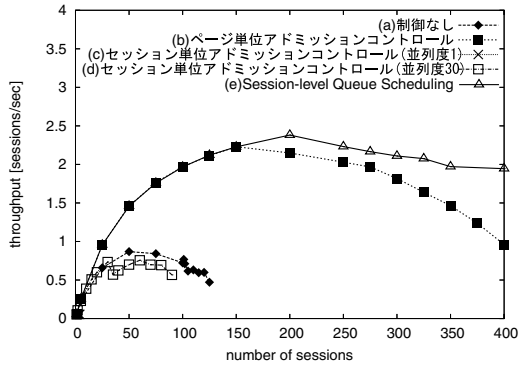


**図 5 セッション処理性能 (タイムアウト 60 秒)**

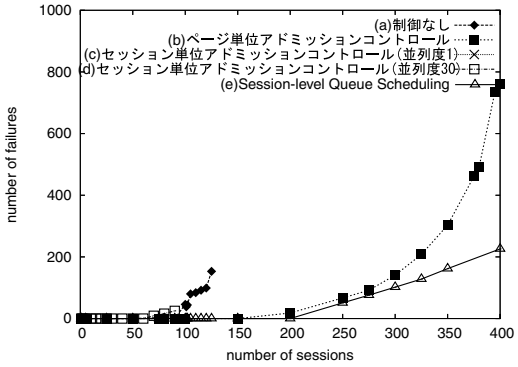


**図 6 セッション失敗数 (タイムアウト 60 秒)**

### 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なった。タイムアウトの時間は 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、セッションの先頭ページから再試行するものとした。

図 5 より、ブラウザのタイムアウトを考慮しても、 (e) の Session-level Queue Scheduling だけが高いセッション処理性能を維持できた。(a) の制御無しの場合、約 50 クライアントで処理性能が頭打ちになる。また、(b) のページ単位アドミッションコントロールでは、同時クライアント数が 150 を越えたぐらいからセッション処理性能を維持できなくなった。一方、(c) のセッション単位アドミッションコントロール (並列度 1)、(d) のセッション単位アドミッションコントロール (並列度 30) 共に、セッションの途中失敗数がそれぞれの最大クライアント数に達した時点で、その失敗したセッションをセッションタイムアウトまで保護するため、他のリクエストが全く処理されなくなる。

#### 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 6 に示す。この結果より、途中失敗数の増加がセッション処理性能の低下を引き起こしている事が分かり、(e) の Session-level Queue Scheduling が最も途中失敗数を抑えることに成功しているため、過負荷時においても高いセッション処理性能を維持できていると考えられる。

さらに、(a) の制御無し、(b) のページ単位アドミッションコントロール、(e) の Session-level Queue Scheduling の三手法における、セッションの途中失敗数の内訳を表 1 に示す。内訳は、(a) の制御無しで測定できた最大クライアント数 (125)、および全体を通しての最大クライアント数 (400) を抽出した。(e) の Session-level Queue Scheduling 以外では、リソースを多く消費するページ (5 ページ目) でコネクションタイムアウトによる途中失敗が多く発生していた。一方、本手法では、途中失敗が低く抑えられると共に、ほとんどのコネクションタイムアウトが先頭ページで起こっているため、セッションを途中まで処理してから失敗することがほとんど無い。つまり、セッションスケジューラによるセッション処理の保護が高いセッション処理性能の維持につながっていることが分かる。

#### 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、sessionInterval の調節とそれにより変動するページキューの平均長の推移を図 7 に示し、各ページ maxThread の調節を図 8 に示す (ページ 2〜4 とページ 6、7 はページ 1 の結果とほぼ同様の結果であり、図を見やすくするために省略)。測定データはクライアント数が 400 の時のものである。グラフ

**表 1 セッション内の各ページでの失敗数の内訳 (125 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 |
|---|---|---|---|---|---|---|---|
| (a) 制御無し | 0/- | 0/- | 0/- | 3/- | 150/- | 0/- | 0/- |
| (b) ページ単位アドミッションコントロール | 0/0 | 0/0 | 0/0 | 0/0 | 0/735 | 0/0 | 0/0 |
| (e) Session-level Queue Scheduling | 0/226 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 | 0/0 |

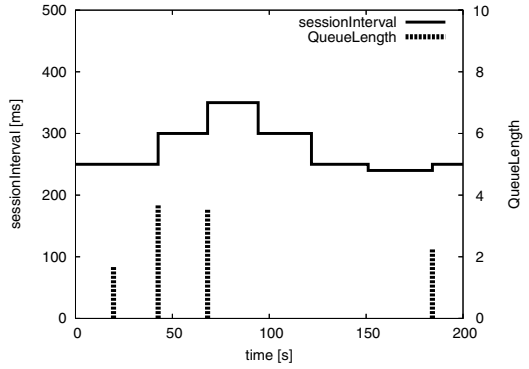


**図 7 sessionInterval とページキューの変動**

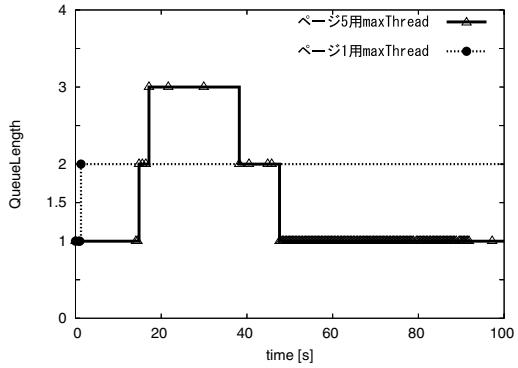


**図 8 各ページの maxThread の変動**

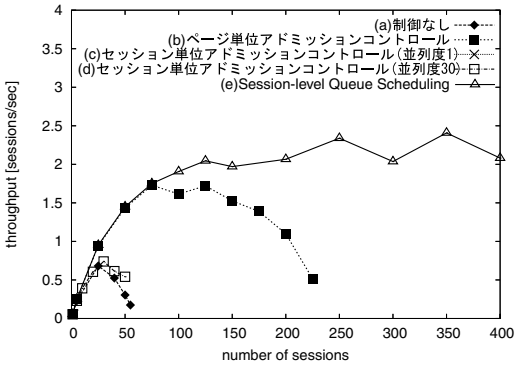


**図 9 セッション処理性能 (タイムアウト 30 秒)**

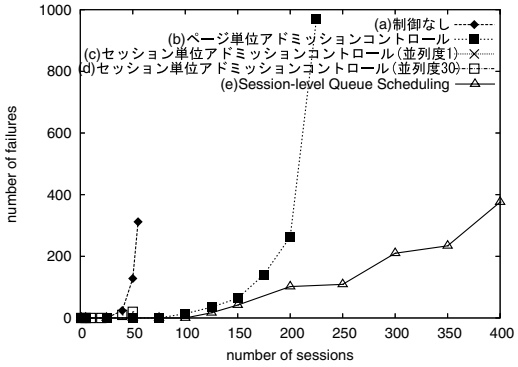


**図 10 途中失敗数 (タイムアウト 30 秒)**

から、ページキューが長すぎて待機リクエスト数が多いときは、セッションスケジューラが sessionInterval を増加させることが分かる。逆に、ページキューが短すぎる場合には、sessionInterval が減少していく。また、各ページの maxThread は、セッション中の最も重いページにあわせて変動し、一定の時間を経過して安定している。

### 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、タイムアウトとシンクタイムをそれぞれ変更して同様の実験を行った。

**タイムアウト時間の影響**

タイムアウト時間がセッション処理性能へ与える影響を調べるためにシンクタイムは 3 秒のままで、タイムアウト時間を 30 秒へ変更して同様の実験を行なった。図 9 より、(e) の Session-level Queue Scheduling

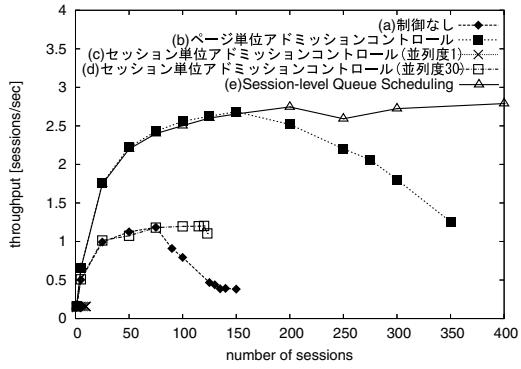


**図 11 セッション処理性能 (シンクタイム 1 秒)**

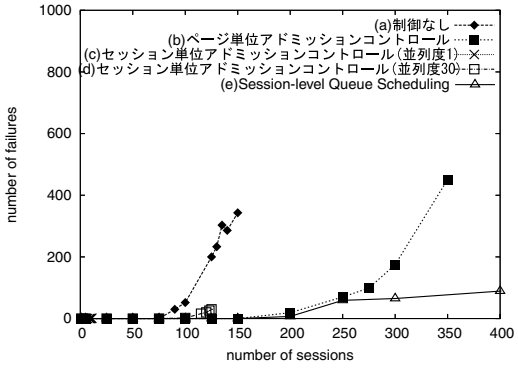


**図 12 途中失敗数 (シンクタイム 1 秒)**

のセッション処理性能は、タイムアウト時間が 60 秒の時の結果 (図 5) とさほど変わらない。これは、セッションスケジューラによりタイムアウトが発生しないようにセッションが保護されているためである。一方、その他の手法はタイムアウト時間を短くすることで、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、セッションタイムアウトに達するまでの同時セッション数が少なくなった (図 10) からである。

### シンクタイム時間の影響

シンクタイム時間がセッション処理性能へ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、シンクタイム時間を 3 秒から 1 秒へ変更して同様の実験を行なった。セッション処理性能は、シンクタイム 3 秒の場合 (図 5) とほとんど変わらなかった (図 11)。シンクタイムの長短は次にリクエストを投げるまでの時間が変わるだけで、タイムアウトによる失敗にはそれほど影響しないため (図 12)、セッション処理性能自体にはほとんど差が生じなかった。

## 5 関連研究

ページ単位アドミッションコントロールは、ページ毎に処理並列度を設定することでリソース競合を解消し、サーバ全体の処理性能を向上させることができる。ページ単位の処理並列度の決定方法には、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 [1] [7] [10] 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 [5] [8] [9] などがある。しかし、静的な決定方法では、実行時に同時に動作する他の Web アプリケーションによる影響を考慮することができない。一方、特定のリソースを監視する手法では、Web アプリケーションが消費する可能性がある全てのリソースを考慮しようとすると、監視する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する。それに対して我々の手法では、リソース消費量の総合的な指標である進捗を用いることによりこれらの問題を解決している。

セッション単位アドミッションコントロールに関する研究はあまりないが、SBAC [3] ではセッションの保護を行うために特定のリソースを監視する。SBAC は新しいセッションを処理するために必要なリソースに余剰が生じた場合にのみ新しいセッションを開始する。しかし、実際のクライアントの挙動を考慮した場合に、シンクタイム中にサーバのリソースが使われていないにも関わらず、処理が行なわれなくなるといった問題点がある。また、本手法と同様にセッション単位とページ単位のアドミッションコントロールを併用している研究もある [2]。しかし、そこで提案されている手法では、セッション単位アドミッションコントロールとページ単位アドミッションコントロールを連携させる機構がないため、実際のワークロードでは SBAC と同様の問題が発生すると考えられる。これら手法の有効性はシミュレーションでのみ検証されており、実際のサーバには適用されていない。

セッション処理性能の向上を妨げる原因の一つであ

るコネクションタイムアウトを発生させにくくするために、ハートビートと呼ばれる機構がサーバに実装されている場合がある。この手法は、タイムアウトする可能性のあるページへのリクエストに対して簡単なページを返しておき、そのページに埋め込まれたメタタグにより一定時間毎にブラウザにページをリロードさせる。これにより、ブラウザのタイムアウトを防止することができる。しかし、ハートビートがサーバに実装されていたとしても、サーバのタイムアウトやクライアントによる読み込みの中止によりセッションが途中終了する場合があるため、本手法も必要になる。

過負荷時の性能を改善するために、SEDA アーキテクチャ [12] を用いたアドミッションコントロールの研究も行われている [11]。SEDA はサーバの処理性能を向上させるための手法で、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なう。各ステージでアドミッションコントロールを行うことで、過負荷時にも各ステージの性能を保つことができる。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。

## 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用いる Session-level Queue Scheduling 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列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ワークロードを用いて本手法の有効性を確かめた。今後の課題としては、より現実的な 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が挙げられる。

## 参考文献


[ 1 ] N. Bhatti and R. Friedrich. Web server support for tiered services. In *IEEE Network, 13(5):64–71*, 1999.

[ 2 ] J. Carlstrom and R. Rom. Application-aware admission control and scheduling in web servers. In *Proceedings of IEEE Infocom*, 2002.

[ 3 ] L. Cherkasova and P. Phaal. Session based admission control: a mechanism for improving the performance of an overloaded web server. In *Proceedings of the International Workshop on Quality of Service*, 1998.

[ 4 ] J. Douceur and W. Bolosky. Progress-based regulation of low-importance processes. In *Proceedings of Symposium on Operating Systems Principles*, pages 247–260, 1999.

[ 5 ] S. Elnikety, E. Nahum, J. Tracey, and W. Zwaenepoel. 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*, pages 276–286, 2004.

[ 6 ] Jakarta. Tomcat. http://jakarta.apahe.org.

[ 7 ] K. Li and S. Jamin. A measurement-based admission-controlled web server. In *Proceedings of INFOCOM*, pages 651–659, 2000.

[ 8 ] D. McWherter, B. Schroeder, A. Ailamaki, and M. Harchol-Balter. Priority mechanisms for OLTP and transactional web applications. In *Proceedings of 20th International Conference on Data Engineering*, 2004.

[ 9 ] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated resource management for cluster-based internet services. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation*, 2002.

[10] T. Voigt, R. Tewari, D. Freimuth, and A. Mehra. Kernel mechanisms for service differentiation in overloaded web servers. In *Proceedings of 2001 USENIX Annual Technical Conference*, pages 189–202, 2001.

[11] M. Welsh and D. Culler. Adaptive overload control for busy internet servers. In *Proceedings of the 4th USENIX Conference on Internet Technologies and Systems*, 2003.

[12] M. Welsh, D. Culler, and E. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *Proceedings of Symposium on Operating Systems Principles*, pages 230–243, 2001.

[13] 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. In **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** *(SPA 2004)*.