ディペンダブル基盤ソフトウェア

千葉　滋
東京工業大学 情報理工学研究科

概要

ソフトウェアをディペンダブルに動作させるための基盤ソフトウェアの研究をおこなっている。近年の web アプリケーションの普及によって、過負荷時に web アプリケーション・サーバの安定性を確保することが求められている。従来技術で作られた web アプリケーション・サーバは、利用者からのリクエストが殺到すると急激に性能が悪化して、最悪の場合はシステム・ダウンに至る。そのような急激な性能悪化は望ましくない。ディペンダブルなシステムとするためには、リクエスト数の増加に対して性能が比例的に悪化する程度におさえる必要がある。我々は web アプリケーション・サーバ自体の改良により信頼性を高める技術と、アプリケーション・ソフトウェアのプログラムを変換し、信頼性を高めるための機能を自動的に埋め込む技術の組み合わせにより、この問題を解決しようとしている。

1　今年度の主な成果

今年度、我々は、ソフトウェアをディペンダブルに動作させる基盤ソフトウェアを開発するための要素技術を主に研究した。主な成果のひとつは実用的に使われているオペレーティングシステムを用いても、web アプリケーションの過負荷時の挙動が必ずしも望ましいものになるとは限らないことを、実機による実験で確かめ、その原因がオペレーティングシステムのタイムスライスにの長さであるという知見を得たことである。この知見をもとに、我々は来年度以降、オペレーティングシステムが変わってもほぼ同一の挙動を示すようにアプリケーション・ソフトウェアの挙動を調整するミドルウェアを開発する予定である。

また、アスペクト指向技術にもとづいた性能チューニングのためのツールを開発した。そのひとつは、オペレーティングシステムのカーネル内での各種の処理時間を細かな単位で計測するカーネル・プロファイラ KLAS である。我々は C 言語で書かれたカーネルのための動的アスペクト指向システムの実装技術を新たに開発し、FreeBSD オペレーティングシステムの上で試作をおこなった。この他に、アプリケーション・ソフトウェアのプログラムを変換し、信頼性を高めるための機能を自動的に埋め込めるようにするため、アスペクト指向言語の研究も進めた。

2　オペレーティングシステムによるサービス性能の違いの研究

現実に運用されている web アプリケーションはさまざまなソフトウェアコンポーネントからなる重層的なシステムである。このため、その性能を決める要因は多岐にわたり、web サーバ単体の性能だけでなく、Java 仮想機械、オペレーティングシステム、そしてハードウェアなど、様々なコンポーネントの性能が関係する。このため、特に利用者からのリクエストが殺到している過負荷の状況では、まったく同一の web アプリケーション・ソフトウェアのプログラムであっても、実行環境によって、その性能上の挙動が大きく異なることがよくある。極端な場合、オペレーティングシステムのマイナーバージョンアップにより、挙動が変わってしまうこともある。このような不安定さは、過負荷時にも高い安定性を求められるシステムの開発を難しくする一因となっている。

我々は、過負荷時の web アプリケーション・

ソフトウェアの挙動のうち、異なるページの間の性能劣化の程度の関係にとくに注目している。実用的な web アプリケーションは複数のページからなり、それぞれのページは異なるプログラムによって、利用者からのリクエストを入力として動的に生成される。ページの中には、常に内容の変わらない (したがってプログラムによって生成されない) 静的なページもあるが、動的に生成されるページの中にも、生成のための処理が軽いものや重いものがある。過負荷の際には、各ページの性能を、処理の軽重にかかわらず、一律に下げた方がよい場合もあるが、必ずしも常にそれが望ましいわけではない。しばしば、利用者は生成処理が軽いページは反応が速く、重いページはそれほど速くないことを暗黙のうちに期待している場合がある。その場合は、過負荷であっても軽いページを優先して処理した方が、利用者の満足度が上がる可能性が高い。元々生成に時間のかかる重いページは、多少余計に時間がかかっても利用者は気づかないが、生成時間が短いページの場合、わずかな処理の遅れでも利用者は非常に遅いと感じがちである。

我々は、実用的に使われているオペレーティングシステムを使った実験により、まったく同じ web アプリケーション・ソフトウェアであっても、オペレーティングシステムが異なると、上記の挙動が異なることを確かめた。Web アプリケーションは多数のコンポーネントからなるシステムであるため、オペレーティングシステムの違いにより、実際に有意な挙動の変化があることは、必ずしも自明でなかった。例えばオペレーティングシステムのスケジューリングが、そのまま各ページを生成するプログラムのスケジューリングに反映されるわけではない。一般的な実装では thread pool と呼ばれる技術が使われるので、個々のスレッドはある瞬間、軽いページを生成していたとしても、別の瞬間は重いページを生成しているかもしれない。常に軽いページを生成しているスレッドや、常に重いページを生成しているスレッドがあるわけではないので、オペレーティングシステムから見ると、どのスレッドも平均的には同じような処理をしていることになり、スケジューリングの差をつけにくい。

2.1 実験

我々は Java で servlet として書かれた 2 つのページを用いて実験をおこなった。片方のページは軽いプログラムで、25 番目のフィボナッチ数を計算して表示するというものである。もう一方は重いページで、XML ファイルをメモリ中に読み込み DOM ツリーを作成してファイル全体をトラバースするというものである。後者の処理は CPU とメモリ資源を大量に消費し、ガベージコレクションを引き起こす。

実験では、これらのページに対して多数のクライアント・マシンから多数の要求を同時に出し、サーバを過負荷状態にして、それぞれのページを単位時間当たり、どれくらい処理できたかを測定した。軽いページにリクエストを出すスレッドの数を 30 に固定し、重いページにリクエストを出すスレッドの数を 0 から 40 に変えて測定をおこなった。全てのスレッドは、8 台のクライアント・マシンの上に分散配置した。実験に使ったサーバ・マシンは Sun Fire V60x であり、これは Intel Xeon 3.06GHz×2、2 GB メモリ、1 Gbps Ethernet を搭載する。クライアント・マシンは Intel Pentium 733MHz、512 MB メモリ、100 Mbps Ethernet を搭載する。クライアント・マシンは 8 台用い、すべて Linux 2.4.19 をオペレーティングシステムとした。サーバ・マシンのオペレーティングシステムには、Solaris 9、Linux 2.6.7、FreeBSD 5.2.1、そして Windows 2003 Server を用い、それぞれの場合の性能を測定した。なお servlet サーバは全て Tomcat 5.0.25 である。Tomcat のリクエストの同時処理数の最大値 (max thread) は 150 である。

実験の結果は図 1 から図 4 のとおりである。重いページを要求するスレッド数が 40 である過負荷時に、重いページの処理性能はどのオペレーティングシステムの場合も、毎秒 2.5 から 3.5 ページ程度であまり変わりがない。しかし軽いページの処理性能を見ると、Solaris では軽いページへのリクエストだけの場合の 22% の性能

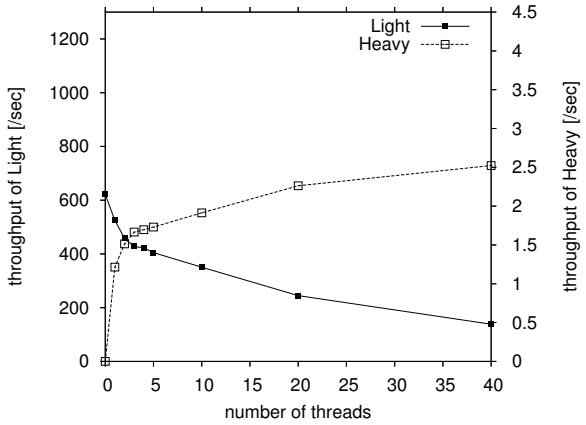

図 1: Solaris

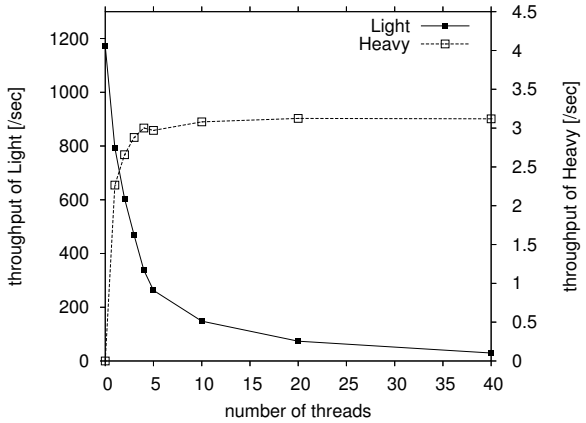

図 2: Linux

を維持しているが、他のオペレーティングシステムでは 1 〜 3% の性能しか出ていない。この結果はけして単純に Solaris の性能がよいことを意味しているわけではない。実際、重いページの処理性能を見ると、Solaris は 4 つのオペレーティングシステムの中で一番悪い。この実験結果は、Solaris は過負荷時にも軽いページに、他のオペレーティングシステムよりも多くの計算資源を割り当てていることを意味する。我々は軽いページを 2 つ、重いページを 1 つという条件でも実験をおこなったが、結果は同様であった。このことから、オペレーティングシステムの違いが、web アプリケーションの挙動の違いとして現れることが確認された。

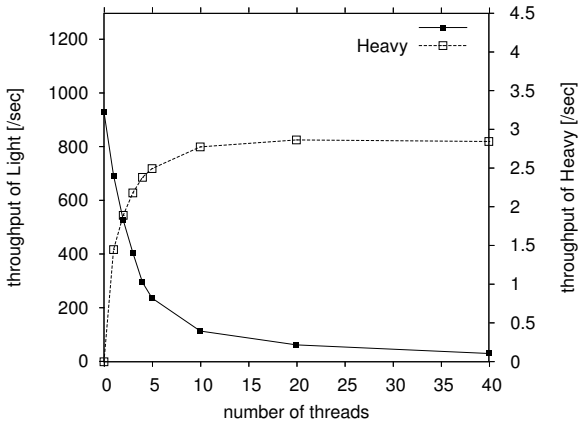

図 3: FreeBSD

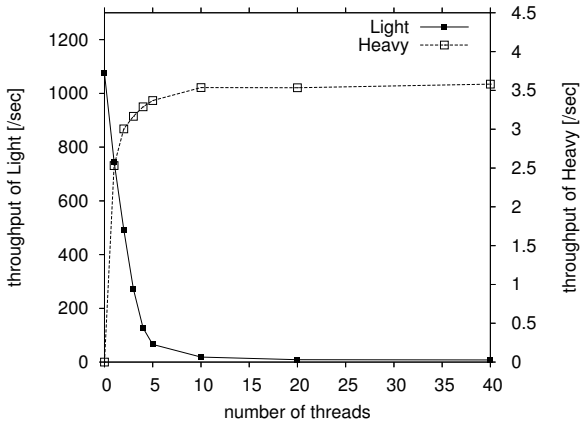

図 4: Windows

2.2 挙動の違いの原因

実験の結果から Solaris とそれ以外のオペレーティングシステムの間で web アプリケーションの挙動に違いがあることが確認できた。我々はさらに実験をおこない、この挙動の主な原因が各オペレーティングシステムのタイムスライスの長短にあることという知見を得た (詳しい実験の内容は [1])。Solaris は他のオペレーティングシステムに比べてタイムスライスが短いため、相対的に各スレッドが高い並列度 (concurrency) で動くためと考えられる。

この推測を裏付けるため、我々は Linux のカーネルを改造してタイムスライスを短くし、同じ実験をおこなってみた。結果を図 5 に示す。元の Linux と異なり、過負荷時も軽いページが一定の割合で処理されていることがわかる。

2.3 今後の課題

実験により、過負荷時の web アプリケーションの挙動はオペレーティングシステムによって異なることがあることが確認できた。これは web アプリケーションの開発にあたって、実運用に使われる環境で性能のチューニングをおこなわなければならないことを示す。

オペレーティングシステムを改造してタイムスライスの長さを変更すれば、web アプリケーションの挙動が変えられることがわかった。しかし商用システムでは、改造したオペレーティングシステムの利用は、許容できない選択子であることが多い。そこで我々は、web アプリケーションのレベルで sleep システムコールを呼ぶなどして、オペレーティングシステムを改造せずに同等の効果を得る方法を現在考えている。これにより、web アプリケーションのディペンダビリティに関する挙動がオペレーティングシステムによらず一定になり、ソフトウェアの開発効率が向上すると予想される。

近年はハードウェアやオペレーティングシステムの仮想化技術が進み、それぞれのレイヤーのソフトウェアを特定の下位レイヤーを仮定してチューニングすることが難しくなっている。今後は、我々が考えているような、下位レイヤーによる性能上の挙動の違いを隠蔽する技術の重要性が増すと考えられる。

3 動的アスペクト指向技術によるカーネルプロファイラ

アスペクト指向技術がデバッグに役立つことはよく知られている。元のプログラムとは別に書かれたデバッグ出力のためのコードを、元のプログラムの中にそれを修正することなく組み込めるからである。しかし、この技術をオペレーティングシステム・カーネルのデバッグに用いるためにはさらなる技術革新が必要であった。カーネルに用いるには、カーネルの再コンパイル・再起動を避けるために、実行中のカーネルを止めずにデバッグコードを組み込む機能がが求められる。また主要なアスペクト指向システムは Java 言語用に実現されているが、アスペクト指向システムを比較的低機能な言語である C 言語上で実現する方法は自明ではない。

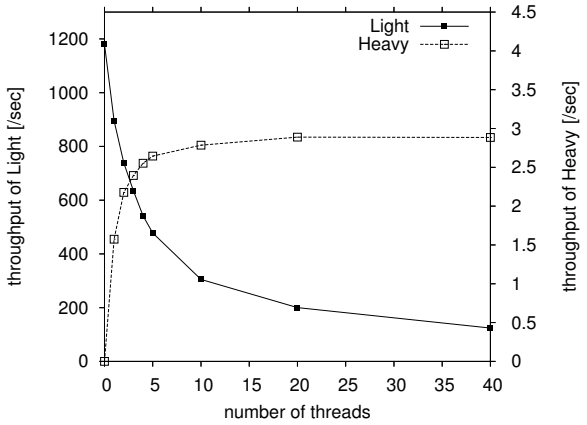

図 5: タイムスライスを短くした Linux

我々は、C コンパイラを拡張して Java 言語並に豊富なシンボル情報を出力させ、その情報をもとに、ブレークポイント・トラップ命令を実行中のカーネルに埋め込む方法で、カーネル用のアスペクト指向システムを実現した [3]。このシステムでは、デバッグコードを実行したい場所にトラップ命令を埋め込み、プログラムの実行がそのトラップ命令に達したら、割り込みハンドラ経由で対応するデバッグコードが実行される。我々は、このシステムを FreeBSD オペレーティングシステム上に実際に実装した。 拡張したコンパイラは GNU C コンパイラである。現在のところ、実装のチューニングが不十分なため、予想される性能が出ていないが、今後チューニングを進め、十分な性能が出るようにする予定である。

参考文献

[1] H. Hibino, K. Kourai, and S. Chiba. Difference of Degradation Schemes among Operating Systems — Experimental analysis for web application servers —. (submitted for publication)

[2] 日比野秀章・松沼正浩・光来健一・千葉滋. アクセス集中時の Web サーバの性能に対する OS の影響. 日本ソフトウェア科学会第２１回大会、2004 年.

[3] Y. Yanagisawa and S. Chiba. A Source-level Kernel Profiler based on Dynamic Aspect Orientation. Dynamic Aspects Workshop (DAW05), AOSD 2005 (to appear).