パフォーマンスへの目: ストレス負荷

ストレス・テストと、正しいツール選択に関わる要素

Jack Shirazi
Director
JavaPerformanceTuning.com

Kirk Pepperdine
CTO
JavaPerformanceTuning.com

2003年 10月 28日

勇敢なる最適化推進者、JavaPerformanceTuning.com のディレクターJack ShiraziとCTOの Kirk Pepperdineはインターネット中で起きているパフォーマンスに関する議論を追跡しており、気が付いた問題をこのコラムで分析、明確化しています。最近TheServerSide.comのメッセージ・ボードを見て、ストレス・テストと負荷テストについて疑問がわいてきました。二人はこの問題を精査し、ツールの選び方でどれほど大きな結果の差が生じるかをお話します。

TheServerSide.comのディスカッション・ボードは普段から非常に活発です。そこで今月はここでしばらくの間立ち止まり、パフォーマンスの領域で何が起きているのかを見てみました。その名前からして、TheServerSideでのパフォーマンスの議論がJ2EEシステムを焦点としたものに偏りがちなのは驚くにあたりません。もちろんこれはJavaプラットフォームのほとんど全てにまたがるので、非常に範囲の広い主題と言えます。またJ2MEシステムでさえJ2EEシステムのクライアントになるのは頻繁にあることなので、時にはJ2MEシステム最適化についての質問も出ることがあります。

ストレス・テスト、負荷テスト

パフォーマンス・リストに関する質問で一番頻繁に聞かれるのは、「私のJ2EEアプリケーションをストレス・テストできるようなツールはありませんか」というものです。その質問に答える前に、まず自問してみましょう。ストレス・テストとは何なのか。開発者はなぜストレス・テストをしたがるのか。（読者の中には、何でもいいからとにかくストレス・テストをとっくの昔に終えていなければという状況の人も少なくないでしょう。しかし、今ここではそういう話をしているのではありません。）ストレス・テストの目的は、アプリケーションのパフォーマンスがどんな条件下で非実用的な程度にまで低下するのかを見つけることにあります。それを調べるためにアプリケーションの負荷がどんどん重くなるように入力を変え、入力の変化でパフォーマンスがどう低下するかを測ったりするわけです。こうした作業は負荷テストとも言われます。（普通、

負荷テストというのは特定形式でのストレス・テスト、つまりユーザーの数を増やしてアプリケーションのストレス・テストをすることを言うのですが。）

アプリケーションのストレス・テストをするのに一番簡単なのは、手動で入力（クライアントの数、リクエストのサイズ、リクエストの頻度、リクエストの混ざり具合、等々）を変化させ、パフォーマンスがどう変化するかプロットしてみることです。アプリケーションによっては、必要なのはこれで全てです。しかし入力が多数あったり、その入力への入力値の範囲が非常に広かったりする場合は多分、自動ツールが必要になってきます。手動テストでは、何かを変更後に再テストしたいようなとき、一巡のテストを正確に再現するのが困難です。複数のユーザーにアプリケーションをテストしてもらう場合などは、一貫した手動テストを走らせるのはほとんど不可能ですし、失業中の友人が沢山いるのでもない限り、アプリケーションのテストをするユーザーの数を増やしていくのも簡単ではないはずです。

万能ツールは無い

残念ながらアプリケーションによって入力の内容、その入力に対する処理が異なるので、汎用のストレス・テスト・ツールというのはありません。ただJ2EEアプリケーションでは、クライアントからの通信はHTTPプロトコルでサーバーに到達します。幸いなことにHTTP上のユーザー動作を、動作制御かつ再現可能な形でシミュレーションする負荷テストツールは数多くあります。無料のApache JMeter、The Grinder、PushToTestなどから、非常に高額なMercury Astraloadのようなものまで様々ですが、一般的に言って性能は値段次第です。つまり高ければ高いなりに、できるテストも多くなります。違いを理解するために、一番基本的な負荷テストツールでできることを見てみましょう。

自分用の負荷テストツールを書く場合を考えると、まずは疑似クライアント毎にスレッドが走るプログラムから始めるでしょう。各スレッドはおそらく`java.net.URL`クラスでサーバーと通信する必要があります。このやり方で、GETとPUTをするだけの、まったく単純なHTTPシミュレーションができます。各スレッドがするのはHTTPリクエストを送信し、戻りを受信し、（思考時間を似せて）少し待ち時間を入れ、と言う繰り返しをすることだけです。一連の動作は別のコンフィギュレーション・ファイルとして、容易に抽出できます。するとあら不思議！基本負荷テストツールの出来上がりです。ただしおそらく、コンフィグレーションに少しオプションが必要になるでしょう。例えばスレッド（つまり疑似クライアント）を何本走らせるか、それらを同時に走らせるのか少しずつ負荷を増やしていくのか、などのオプションです。そして当然ながら、そもそもの目的である、サーバーとの応答時間を測る必要があるでしょう。

そんなに簡単ならいいんですが・・・

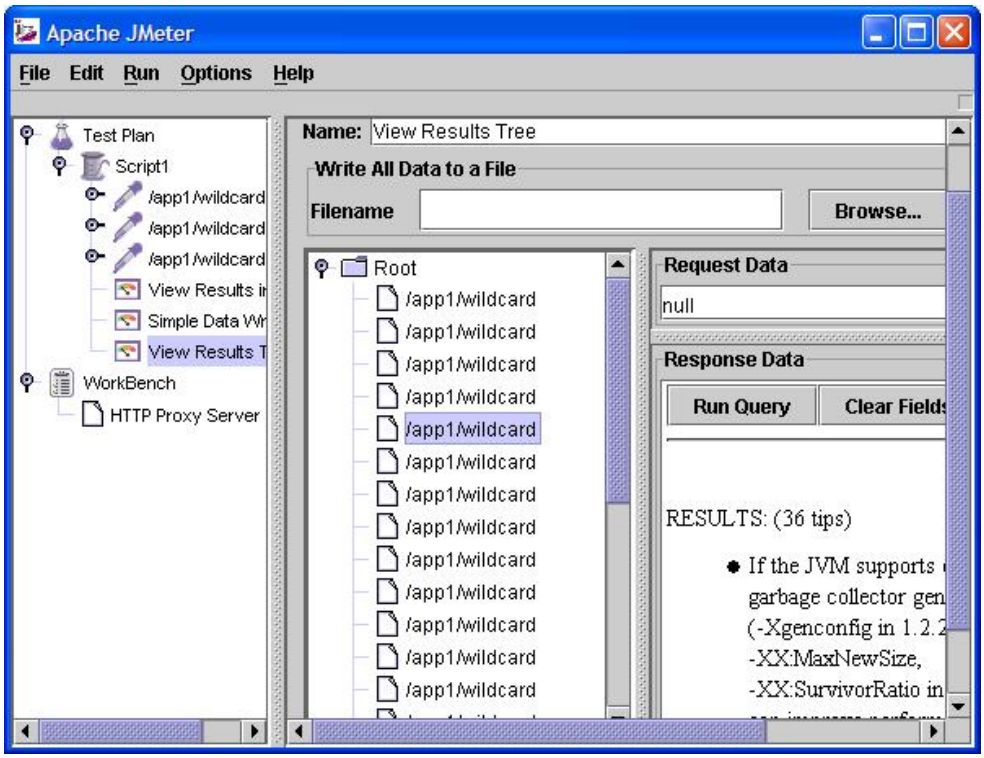
ではもうちょっと複雑なやり取りはどうでしょう。つまり次のリクエストが、その前のリクエストの結果次第で変わるような場合です。また、クッキーはどうでしょう。多くのセッション志向のJ2EEシステムではクッキーが重要になります。変動する入力に対しては？J2EEアプリケーション・クライアントが、次の通信に進むためにJavaScriptを処理する必要がある場合は？応答時間を収集したあと、その解析はどうしますか？CPU時間、ネットワーク利用、ヒープサイズ、ページング動作、データベース動作等についてはどうでしょう？

ハイエンドのテストツールと基本ツールが違うのは、こうしたもろもろの機能の有無に加えて、ブラウザ・セッションを記録し、テストスクリプトとして利用できるようにする機能の有無にあると言えます。自分に合ったツールはどう選ぶべきでしょうか。これはもちろん、何がいつまで

に、どのくらいの予算で必要かによります。基本的な機能が用意できたら、今度はツールの生産性を考えるべきです。一般的に言って、解析ツールが多くあればあるほど、つまりパフォーマンス・データを多く記録できればできるほど、生産性は上がるでしょう（そしてツールに費やすお金も多くなります）。一番ハイエンドの負荷テストツールなら、複数のブラウザをシミュレートでき、ほとんどのアプリケーション・サーバーに統合可能で、OS、JVM、データベース統計を含む複数のサーバー・ホストのパフォーマンス・データを収集し、データセットを生成し、高度な解析ツールで後から解析できるようにもなっています。一方ローエンドの負荷テストツールは無料です。予算が厳しい昨今にあっては「無料」の意味するところも大きいと言えます。

図1は無料の負荷テストツールの一つであるApache JMeterを示しています。ここでは自動記録したスクリプトを示しています。

図1. JMeterで、自動記録したスクリプトを示す



豊富な機能群

ストレス・テストツールの基本的な機能は見ましたが、他にはどんな追加機能があれば良いでしょう？各種の負荷テストツールはどう違うのでしょう？まず何よりも必要なのはもちろん、そのツールがアプリケーションのクライアントをシミュレートできる、ということでしょう。そのアプリケーションがブラウザ機能をある特別な組み合わせで使っていたり、標準でないクライアント技術を使っていたりする場合には、一部のツールは候補外となります。基本的な機能以外に、負荷テストの生産性を上げるような機能もあります。自分に合った負荷テストツールを選ぶに際して考慮すべき機能項目として、次のリストが役に立つでしょう。

クライアントをシミュレートできる
　　アプリケーションが使用する機能やプロトコルを処理できることが第一です。

複数の疑似クライアントを走らせることができる
: 負荷テストツールの一番基本的な機能であり、この機能で何が負荷テストツールで何がそうでないかを決められます（一部のフレームワークは負荷テストツールになりすましま す）。

スクリプトの編集ができ、スクリプトを実行できる
: クライアントとサーバー間のやり取りをスクリプトできないとすると、一番簡単なクライアント以外は処理できないことになってしまいます。スクリプトの編集機能は必須です（少しの変更であればスクリプトの再生成までは不要なはずです）。

セッションをサポートしている
: 負荷テストツールがセッションやクッキーをサポートできないと負荷テストツールとは言えませんし、ほとんどのJ2EEアプリケーションの負荷テストはできません。

ユーザー数を設定できる
: テストツールには、疑似ユーザー何人が各スクリプトを走らせられるかを規定でき、またその人数を時間と共に変化させられる機能が必要になります。負荷テストは多くの場合、少数のユーザーから始め、だんだん多数のユーザーになるように動作すべきものだからです。

成功、エラー、失敗のレポートができる
: 各スクリプトは、どういうやり取りであれば成功、どうであれば失敗かエラーかが定義している必要があります。（エラーは戻りページがまったく無い、失敗はページ上のデータが間違っている、などのように。）

ページ表示ができる
: テストが順調に動作していることが確認できるように、疑似ユーザーに送られるページの一部を負荷テストツールで見ることができれば便利です。

結果のエクスポートができる
: 表計算やカスタム設計した解析スクリプトなどを使ってテスト結果を解析したくなるものです。高度な解析機能を持った負荷テストツールもたくさんありますが、データをエクスポートできれば任意な手段でデータを解析して一覧できるので、より柔軟な処理ができると言えます。

思考時間
: 実際の使用状況では、ユーザーが一つのページ・リクエストを出すと瞬時に次のリクエストを出す、ということはあり得ません。普通は一つのページを見てから次のページを見るまでには時間があるものです。ユーザーの振る舞いをより現実的にシミュレートするために、スクリプトにディレイを入れるための表現として、思考時間 (Think time)は標準のものです。大部分の負荷テストツールでは、統計分布に基づいて、ランダムに思考時間を生成できるようになっています。

クライアントがリストからデータを選択できる
: ユーザーが同じデータのセットを扱うことは稀で、普通各ユーザーはサーバーとそれぞれ別々のやりとりをします。疑似ユーザーも同様に振舞うべきでしょう。ユーザーとの相互動作上で重要な点のデータをスクリプトがリストから選択でき、ユーザーが色々なデータを扱っているように見えるようにできていれば便利です。

手動実行されたセッションのスクリプトを記録できる
: スクリプトを書くよりも、ブラウザに対して手動でセッションを走らせ、ツールにそのセッションを記録させて後から編集できるようになっていた方がずっと便利です。

Javaスクリプト
: 一部のアプリケーションではJavaScriptを縦横に使っており、疑似クライアントもJavaScriptをサポートしている必要があります。ただし、クライアント側でJavaScriptを使うと、テストに使うシステムに対するシステム要求も増えることにもなります。

解析ツール
: パフォーマンスの測定は話の半分でしかなく、残り半分はパフォーマンスの解析になります。測定ツールを作る人以上に解析ツールをうまく作れる人がいるでしょうか？う～ん、理屈はその通り。いずれにせよ、ツールキットについてくる解析ツールは多いに越したことはありません。

サーバー側の統計がとれる
: 基本的な負荷テストツールはクライアントとサーバー間のやり取りを、クライアントから見た応答時間で測ります。他の統計、例えばCPU利用やページフォルト率なども取れれば便利です。統計がとれればとれただけ、負荷テストツールでいろいろなことができます。統計データがあれば、サーバー負荷やスループット統計から見たクライアントの応答時間を見ることなどもできるようになります。

最後に

どんなツールであれ、ツールの限界というのは多くの場合、そのツールを使う人の能力や知識、創造力によって変わると言うことができます。ここまで負荷テストツールに何を求めるかを説明しながら、同時にそのツールで何ができるかを説明してきました。そうしたツールでどこまでできるのか、読者の創造力に期待したいと思います。

著者について

Jack Shirazi

Jack ShiraziはJavaPerformanceTuning.comのディレクターであり、Java Performance Tuning, 2nd Edition（O'Reilly刊）の著者でもあります。

Kirk Pepperdine

Kirk PepperdineはJava Performance Tuning.comのCTO（Chief Technical Officer）であり、過去15年間、オブジェクト技術やパフォーマンス調整に注力してきました。Ant Developer's Handbook（MacMillan刊）の共著者でもあります。