Sun Java System Web Server 7.0 パフォーマンスのチューニング、サイジング、およびスケーリング

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–1615

Copyright 2007 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 U.S.A. All rights reserved.

本書で説明する製品で使用されている技術に関連した知的所有権は、Sun Microsystems, Inc. に帰属します。特に、制限を受けることなく、この知的所有権には、米国特許、および米国をはじめとする他の国々で申請中の特許が含まれています。

U.S. Government Rights – Commercial software. Government users are subject to the Sun Microsystems, Inc. standard license agreement and applicable provisions of the FAR and its supplements.

本製品には、サードパーティーが開発した技術が含まれている場合があります。

本製品の一部は Berkeley BSD システムより派生したもので、カリフォルニア大学よりライセンスを受けています。UNIX は、X/Open Company, Ltd. が独占的にライセンスしている米国ならびにほかの国における登録商標です。

Sun、Sun Microsystems、Sun のロゴマーク、Solaris のロゴマーク、Java Coffee Cup のロゴマーク、docs.sun.com、 JavaServer Pages、JSP、JVM、JDBC、Java HotSpot、 Java、Solaris は、米国およびその他の国における米国 Sun Microsystems, Inc. (以下、米国 Sun Microsystems 社とします) の商標もしくは登録商標です。Sun のロゴマークおよび Solaris は、米国 Sun Microsystems 社の登録商標です。すべての SPARC 商標は、米国 SPARC International, Inc. のライセンスを受けて使用している同社の米国およびその他の国における商標または登録商標です。SPARC 商標が付いた製品は、米国 Sun Microsystems 社が開発したアーキテクチャーに基づくものです。

OPEN LOOK および Sun™ Graphical User Interface は、米国 Sun Microsystems 社が自社のユーザーおよびライセンス実施権者向けに開発しました。米国 Sun Microsystems 社は、コンピュータ産業用のビジュアルまたはグラフィカルユーザーインタフェースの概念の研究開発における米国 Xerox 社の先駆者としての成果を認めるものです。米国 Sun Microsystems 社は米国 Xerox 社から Xerox Graphical User Interface の非独占的ライセンスを取得しており、このライセンスは、OPEN LOOK GUI を実装するか、または米国 Sun Microsystems 社の書面によるライセンス契約に従う米国 Sun Microsystems 社のライセンス実施権者にも適用されます。

この製品は、米国の輸出規制に関する法規の適用および管理下にあり、また、米国以外の国の輸出および輸入規制に関する法規の制限を受ける場合があります。核、ミサイル、生物化学兵器もしくは原子力船に関連した使用またはかかる使用者への提供は、直接的にも間接的にも、禁止されています。このソフトウェアを、米国の輸出禁止国へ輸出または再輸出すること、および米国輸出制限対象リスト (輸出が禁止されている個人リスト、特別に指定された国籍者リストを含む) に指定された、法人、または団体に輸出または再輸出することは一切禁止されています。

本書は、「現状のまま」をベースとして提供され、商品性、特定目的への適合性または第三者の権利の非侵害の黙示の保証を含みそれに限定されない、明示的であるか黙示的であるかを問わない、なんらの保証も行われないものとします。

070611@17466

目次

図目次

表目次

例目次

はじめに

このマニュアルでは、Sun Java™ System Web Server (以降、Web Server と呼ぶ) のパフォーマンスを改善する可能性のある、実行可能な調整について説明します。このマニュアルでは、チューニング、スケーリング、およびサイジングに関するヒントや提案、一般的なパフォーマンス問題で実施可能な解決方法、およびスケーラビリティー調査から得られたデータを提供します。また、その他の構成およびプラットフォーム固有の問題についても説明します。

対象読者

このマニュアルの対象読者は、上級管理者のみです。変更を行う前に、必ずこのマニュアルとその他の関連サーバーマニュアルを読んでください。サーバーのチューニングを行う場合は細心の注意を払い、変更前に必ず設定ファイルをバックアップするようにしてください。

このマニュアルをお読みになる前に

Web Server は、スタンドアロン製品としてインストールすることが可能です。あるいは、ネットワークまたはインターネット環境にわたって分散しているエンタープライズアプリケーションをサポートするソフトウェアインフラストラクチャーである Sun Java Enterprise System (Java ES) のコンポーネントとして購入することもできます。Web Server を Java ES のコンポーネントとしてインストールする場合は、`http://docs.sun.com/coll/1657.1` にあるシステムマニュアルをよく読むことをお勧めします。

Web Server のマニュアルセット

Web Server のマニュアルセットでは、Web Server をインストールおよび管理する方法について説明しています。Web Server のマニュアルの URL は、`http://docs.sun.com/coll/1664.1` です。Web Server への導入としては、次の表に示されている順序でマニュアルを参照してください。

表 P–1 Web Server のマニュアルセットの内容

マニュアル名	内容
『Sun Java System Web Server 7.0 Documentation Center』	タスクや主題ごとに整理された Web Server のマニュアルのトピック
『Sun Java System Web Server 7.0 リリースノート (UNIX 版)』	■ ソフトウェアおよびマニュアルについての最新情報 ■ Web Server をインストールするためのサポートされているプラットフォームとパッチ要件
『Sun Java System Web Server 7.0 Installation and Migration Guide』	以下のインストールおよび移行作業の実行 ■ Web Server とその各種コンポーネントのインストール ■ Sun ONE Web Server 6.0 または 6.1 から Sun Java System Web Server 7.0 へのデータの移行
『Sun Java System Web Server 7.0 管理ガイド』	次の管理タスクの実行 ■ 管理およびコマンド行インタフェースの使用 ■ サーバー環境の設定 ■ サーバーインスタンスの使用 ■ サーバーアクティビティーの監視およびログ ■ サーバー保護のための証明書および公開鍵暗号の使用 ■ サーバー保護のためのアクセス制御の設定 ■ Java Platform Enterprise Edition (Java EE) のセキュリティー機能の使用 ■ アプリケーションの配備 ■ 仮想サーバーの管理 ■ パフォーマンスニーズに合わせたサーバー作業負荷の定義およびシステムのサイズ決定 ■ サーバードキュメントのコンテンツと属性の検索、およびテキスト検索インタフェースの作成 ■ コンテンツ圧縮のためのサーバー設定 ■ WebDAV を使用した Web 発行およびコンテンツオーサリングのためのサーバー設定
『Sun Java System Web Server 7.0 Developer's Guide』	以下を実行するためのプログラミングテクノロジおよび API の使用 ■ Sun Java System Web Server の拡張および変更 ■ クライアント要求に応答したコンテンツの動的な生成、およびサーバーのコンテンツの変更

表 P–1 Web Server のマニュアルセットの内容 (続き)

マニュアル名	内容
『Sun Java System Web Server 7.0 Update 1 NSAPI Developer's Guide』	カスタム NSAPI (Netscape™ Server Application Programmer's Interface) プラグインの作成
『Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications』	Sun Java System Web Server における Java Servlet および JavaServer Pages™ (JSP™) テクノロジの実装
『Sun Java System Web Server 7.0 Administrator's Configuration File Reference』	設定ファイルの編集
『Sun Java System Web Server 7.0 パフォーマンスのチューニング、サイジング、およびスケーリング』	パフォーマンス最適化のための Sun Java System Web Server のチューニング
『Sun Java System Web Server 7.0 Troubleshooting Guide』	Web Server のトラブルシューティング

関連マニュアル

Sun Java Enterprise System (Java ES) とそのコンポーネントに関するすべてのマニュアルの URL は、`http://docs.sun.com/app/docs/prod/entsys.06q4` です。

デフォルトのパスとファイル名

次の表は、このマニュアルで使用するデフォルトのパスやファイル名について説明したものです。

表 P–2 デフォルトのパスとファイル名

プレースホルダ	説明	デフォルト値

install_dir	Web Server のベースインストールディレクトリを表します。	Solaris™ プラットフォームへの Sun Java Enterprise System (Java ES) のインストール:
		`/opt/SUNWwbsvr7`
		Linux および HP-UX プラットフォームへの Java ES のインストール:
		`/opt/sun/webserver7`
		Windows プラットフォームへの Java ES のインストール:
		System Drive:`\Program Files\Sun\JavaES5\WebServer7`
		Solaris、Linux、および HP-UX へのその他のインストールで、root ユーザーでない場合:
		user's home directory`/sun/webserver7`
		Solaris、Linux、および HP-UX へのその他のインストールで、root ユーザーの場合:
		`/sun/webserver7`
		Windows のすべてのインストールの場合:
		System Drive:`\Program Files\Sun\WebServer7`

instance_dir	インスタンス固有のサブディレクトリを含むディレクトリ。	Java ES インストールの場合、Solaris 上でのインスタンスのデフォルトの場所は、次のとおりです。
		`/var/opt/SUNWwbsvr7`
		Java ES インストールの場合、Linux および HP-UX 上でのインスタンスのデフォルトの場所は、次のとおりです。
		`/var/opt/sun/webserver7`
		Java ES インストールの場合、Windows 上でのインスタンスのデフォルトの場所は、次のとおりです。
		System Drive:`\Program Files\Sun\JavaES5\WebServer7`
		スタンドアロンインストールの場合、Solaris、Linux、および HP-UX 上でのインスタンスのデフォルトの場所は、次のとおりです。<*install_dir*>
		スタンドアロンインストールの場合、Windows 上でのインスタンスのデフォルトの場所は、次のとおりです。
		System Drive:`\Program Files\sun\WebServer7`

表記上の規則

このマニュアルでは、次のような字体や記号を特別な意味を持つものとして使用します。

表 P–3 表記上の規則

字体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コード例を示します。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 machine_name% you have mail.
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して示します。	machine_name% **su** Password:
aabbcc123	変数を示します。実際に使用する特定の名前または値で置き換えます。	ファイルを削除するには、rm *filename* と入力します。
『』	参照する書名を示します。	『コードマネージャー・ユーザーズガイド』を参照してください。
「」	参照する章、節、ボタンやメニュー名、強調する単語を示します。	第 5 章「衝突の回避」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合に、継続を示します。	sun% **grep '^#define ** **XV_VERSION_STRING'**

コード例は次のように表示されます。

- C シェル

  machine_name% **command y|n** [*filename*]

- C シェルのスーパーユーザー

  machine_name# **command y|n** [*filename*]

- Bourne シェルおよび Korn シェル

  $ **command y|n** [*filename*]

- Bourne シェルおよび Korn シェルのスーパーユーザー

```
# command y|n [filename]
```

[ ] は省略可能な項目を示します。上記の例は、*filename* は省略してもよいことを示しています。

| は区切り文字 (セパレータ) です。この文字で分割されている引数のうち 1 つだけを指定します。

キーボードのキー名は英文で、頭文字を大文字で示します (例: Shift キーを押します)。ただし、キーボードによっては Enter キーが Return キーの動作をします。

ダッシュ (-) は 2 つのキーを同時に押すことを示します。たとえば、Ctrl-D は Control キーを押したまま D キーを押すことを意味します。

記号の規則

次の表では、このマニュアルで使用されている記号について説明します。

表 P–4 記号の規則

記号	説明	例	意味
[ ]	省略可能な引数やコマンドオプションが含まれます。	ls [-l]	-l オプションは必須ではありません。
{ | }	必須のコマンドオプションの選択肢を囲みます。	-d {y|n}	-d オプションには y 引数か n 引数のいずれかを使用する必要があります。
${ }	変数参照を示します。	${com.sun.javaRoot}	com.sun.javaRoot 変数の値を参照します。
-	同時に押すキーを示します。	Control-A	Control キーを押しながら A キーを押します。
+	順番に押すキーを示します。	Ctrl+A+N	Control キーを押してから放し、それに続くキーを押します。
→	グラフィカルユーザーインタフェースでのメニュー項目の選択順序を示します。	「ファイル」→「新規」→「テンプレート」	「ファイル」メニューから「新規」を選択します。「新規」サブメニューから「テンプレート」を選択します。

マニュアル、サポート、およびトレーニング

Sun のサービス	URL	内容
マニュアル	http://jp.sun.com/documentation/	PDF 文書および HTML 文書をダウンロードできます。
サポートおよびトレーニング	http://jp.sun.com/supportraining/	技術サポート、パッチのダウンロード、および Sun のトレーニングコース情報を提供します。

◆ ◆ ◆ 第 1 章

1 パフォーマンスと監視の概要

Sun Java System Web Server (以降、Web Server と呼ぶ) は、世界でもっとも要求の厳しい、高トラフィックサイトのニーズを満たすように設計されています。Web Server は、静的に生成されるコンテンツと、動的に生成されるコンテンツの両方に対応できます。Web Server を SSL モードで実行して、セキュリティー保護された情報転送を実現することもできます。

このマニュアルは、パフォーマンスニーズに合わせたサーバー作業負荷の定義やシステムのサイジングに役立ちます。ただし、環境はそれぞれ固有であるため、ここで示す提案の影響も環境ごとに異なります。最終的には、独自の判断と観察に基づいて、もっとも適した調整方法を選択してください。

この章では、サーバーパフォーマンスの注意点の一般的な説明をしたあと、サーバーパフォーマンスの監視に関するより具体的な情報を提供します。

この章の内容は、次のとおりです。

パフォーマンスの問題

サーバーのサイジングのための最初の手順は、要件の決定です。パフォーマンスが意味するものは、その対象がユーザーと Web マスターとでは異なります。ユーザーは、速い応答時間 (一般には 100 ミリ秒未満)、高可用性 (「接続拒否」のメッセージが表示されないこと)、およびインタフェースをできるだけ制御できることを望んでいます。これに対して、Web マスターやシステム管理者は、高い接続レート、高い

データスループット、および100% 近い稼働時間を実現したいと考えています。さらに、仮想サーバーの場合は、価格設定ごとに異なったレベルのパフォーマンスを提供することが目標である可能性があります。特定の状況について、パフォーマンスがどういう意味を持つかを定義してください。

次のいくつかの点を考慮する必要があります。

- 並行ユーザーのピーク数
- セキュリティーの要件

  Web Server のデータストリームを SSL で暗号化すると、電子商取引やその他の高いセキュリティーが必要なアプリケーションに対するサイトの信頼性は大幅に向上しますが、CPU 負荷にも大きな影響を与える可能性があります。詳細については、23 ページの「SSL パフォーマンス」を参照してください。
- ドキュメントツリーのサイズ
- 動的または静的コンテンツ

  サービス対象のコンテンツによって、サーバーのパフォーマンスが影響を受けます。大部分が静的な HTML を提供している Web Server は、クエリーごとに CGI を実行する必要のあるサーバーよりはるかに高速に実行できます。

構成

構成に基づくすべてのサーバーインスタンスに同じチューニング情報が含まれるように、構成レベルで特定のチューニングパラメータが設定されます。さらに、構成に基づくすべてのインスタンスのパフォーマンスを監視できるように、構成レベルで一部の監視情報を表示できます。ただし、監視情報の全体は、個別のサーバーインスタンスまたは仮想サーバーレベルで表示できます。構成ごとに 1 つの Web Server インスタンスを使用している (サーバーがサーバーファームに含まれていない) 場合、構成レベルの統計情報には、その構成に基づく 1 つのサーバーインスタンスの情報が示されます。

仮想サーバー

仮想サーバーによって、パフォーマンス向上のプロセスに別のレイヤーが追加されます。特定の設定が構成に対してチューニング可能であるのに対して、その他の設定は個別の仮想サーバーに基づきます。

また、サービスの品質 (QoS) 機能を使用して、個別の仮想サーバーに対する資源利用状況の制約を設定することもできます。たとえば、QoS 機能を使用すると、仮想サーバーに対して許可される帯域幅の量や接続数を制限できます。これらのパフォーマンスの制限を設定したり、追跡したり、必要に応じて適用したりすることができます。

サービスの品質機能の使用の詳細については、『Sun Java System Web Server 7.0 管理ガイド』を参照してください。

サーバーファーム

Web Server のクラスタ分布機能を使用すると、サーバーファームへの配備が容易になります。サーバーファーム内のすべてのサーバーが同一の構成を共有するため、チューニングはサーバー単位には実行されません。

64 ビットサーバー

64 ビットの Web Server のパフォーマンスが必ずしも 32 ビットの Web Server のパフォーマンスより高いわけではありませんが、64 ビットサーバーの方が拡張性に優れています。32 ビットの Web Server のプロセスは 4G バイトのアドレス空間に制限されているため、特定の制限を超える同時セッション数をサポートしようとすると、アドレス空間が不足する場合があります。使用可能なメモリーや CPU リソースがホストマシンに存在しても、アドレス空間の制限のために、32 ビットの Web Server がそれらを利用できない可能性があります。64 ビットの Web Server は、32 ビットサーバーより多くのアプリケーションやサーブレットを実行できます。また、64 ビットの Web Server が数 G バイトの静的コンテンツをキャッシュできるのに対して、32 ビットの Web Server は 4G バイトのアドレス空間に制限されます。

一般に、64 ビットの Web Server に対するチューニングは、32 ビットの Web Server に対するチューニングとほぼ同じです。その違いの大部分は、オペレーティングシステムのレベルでチューニングされます。チューニングの詳細については、105 ページの「パフォーマンスベンチマークのための UltraSPARC T1 ベースシステムのチューニング」で説明されています。

SSL パフォーマンス

SSL は常にスループットに大きな影響を与えるため、最適なパフォーマンスを得るには、SSL の使用を最小限に抑えるか、またはマルチ CPU サーバーを使用して処理することを検討してください。

SSL では、Web Server が NSS ライブラリを使用します。ただし、SSL に使用可能なオプションはほかにもあります。

- Solaris 10 オペレーティングシステムを使用している場合は、カーネル SSL (KSSL) が使用できます。NSS とは異なり、使用可能なすべてのアルゴリズムは含まれていませんが、多くの場合はより優れたパフォーマンスが提供されます。
- SSL 用の暗号化カードハードウェアアクセラレータを使用してパフォーマンスを向上させることもできます。
- Solaris で 64 ビットの Web Server を使用している場合は、UltraSPARC T1 プロセッサの暗号化アクセラレータを使用できます。

サーバーパフォーマンスの監視

このマニュアルで説明されている調整を、それらの効果を測定せずに行なっても意味はありません。変更を行う前とあとにシステムの動作を測定しなければ、その変更が良い考えだったか、悪い考えだったか、あるいは無関係だったかがわかりません。Web Server のパフォーマンスは、次に示す何種類かの方法で監視できます。

表 1–1 パフォーマンスの監視方法

監視方法	有効にする方法	アクセス方法	利点と要件
管理コンソール経由の統計情報	デフォルトで有効	管理コンソールで、構成の「監視」タブをクリックします	セッションスレッドがハングアップしている場合にアクセス可能。管理サーバーが実行されている必要があります。

個別の `wadm` コマンド経由の統計情報	デフォルトで有効	次の `wadm` コマンドを経由します
		`get-config-stats`
		`get-virtual-server-stats`
		`get-webapp-stats`
		`get-servlet-stats`
			セッションスレッドがハングアップしている場合にアクセス可能。管理サーバーが実行されている必要があります。

ブラウザ経由の XML 形式の統計情報 (`stats-xml`)	管理コンソール、または設定ファイルの編集を経由して有効にします	URI を経由します	管理サーバーが実行されている必要はありません。

表 1–1 パフォーマンスの監視方法 (続き)

監視方法	有効にする方法	アクセス方法	利点と要件
コマンド行インタフェース経由の XML 形式の統計情報 (`stats-xml`)	デフォルトで有効	`wadm` コマンド `get-stats-xml` を経由します	セッションスレッドがハングアップしている場合にアクセス可能。管理サーバーが実行されている必要があります。
ブラウザ経由の `perfdump`	管理コンソール、または設定ファイルの編集を経由して有効にします	URI を経由します	管理サーバーが実行されている必要はありません。
コマンド行インタフェース経由の `perfdump`	デフォルトで有効	`wadm` コマンド `get-perfdump` を経由します	セッションスレッドがハングアップしている場合にアクセス可能。管理サーバーが実行されている必要があります。
Java ES の監視	デフォルトで有効	Java ES 監視コンソールを経由します	Java ES インストールでのみ使用可能。管理サーバーが実行されている必要があります。

サーバーの監視は、コンピューティングリソースにある程度の影響を与えます。一般に、影響がもっとも少ないのは URI を経由した `perfdump` の使用であり、次が URI を経由した `stats-xml` の使用です。管理サーバーの使用はコンピューティングリソースを消費するため、コマンド行インタフェースと管理コンソールがもっとも影響の大きい監視方法です。

これらの監視方法の詳細については、以降の節を参照してください。

- 25 ページの「統計情報について」
- 28 ページの「管理コンソールを使用した現在のアクティビティーの監視」
- 28 ページの「CLI を使用した現在のアクティビティーの監視」
- 32 ページの「stats.xml を使用した現在のアクティビティーの監視」
- 34 ページの「perfdump を使用した現在のアクティビティーの監視」
- 41 ページの「Java ES 監視コンソールを使用した現在のアクティビティーの監視」

統計情報について

管理コンソールのユーザーインタフェース、コマンド行インタフェース、`stats-xml` URI、および `perfdump` を経由して、多くのパフォーマンス統計情報を監視できま

す。これらの監視方法のすべてについて、サーバーは収集した統計情報を使用します。統計情報が収集されないかぎり、これらのどの監視方法も機能しません。

これらの統計情報によって、構成レベル、サーバーインスタンスレベル、または仮想サーバーレベルの情報が得られます。統計情報は、機能別に分類されます。

構成については、次の項目の統計情報を表示できます。

- 要求
- エラー
- 応答時間

サーバーインスタンスについては、次の項目の統計情報を表示できます。

- 要求
- エラー
- 応答時間
- 一般
- Java 仮想マシン (JVM™)
- 接続キュー
- キープアライブ
- DNS
- ファイルキャッシュ
- スレッドプール
- セッションレプリケーション
- プロファイルデータを含むセッションスレッド (プロファイリングが有効になっている場合に表示可能)
- Java Database Connectivity (JDBC™) (JDBC リソースが作成され、接続プールにアクセスしている場合に表示可能)

仮想サーバーについては、次の項目の統計情報を表示できます。

- 一般
- 応答
- Web アプリケーション
- プロファイルデータ (プロファイリングが有効になっている場合に表示可能)
- サーブレットとサーブレット応答キャッシュ (`sun.web.xml` でサーブレットキャッシュが有効になっている場合に表示可能)

サービスの品質 (QoS) が有効になっていない場合は、一部の統計情報、たとえば、開いている接続の数、開いている接続の最大数、転送レート (バイト数)、最大転送レート (バイト数) などはデフォルトで 0 になります。

統計情報の有効化

Web Server では、統計情報はデフォルトでアクティブになります。ただし、いったん無効にした場合、サーバーのパフォーマンスを監視するにはもう一度有効にする必要があります。統計情報を有効にするには、管理コンソールまたは wadm コマンド行ユーティリティー (CLI) を使用します。

注 – 統計情報の収集は、パフォーマンスに若干の影響を与えます。

▼ 管理コンソールから統計情報を有効にする

1. 管理コンソールの「共通操作」ページで、構成を選択します。
2. 「構成を編集」をクリックします。
3. 「一般」タブをクリックします。
4. 「監視設定」サブタブをクリックします。
5. 「監視設定」ページの「一般設定」で、「統計コレクション」の「有効」チェックボックスを選択します。
6. 間隔とプロファイリングを設定します。
   - この間隔は、統計情報の更新の間の期間 (秒単位) です。この設定を大きくする (頻度を少なくする) ほど、パフォーマンスは向上します。最小値は 0.001 秒、デフォルト値は 5 秒です。
   - プロファイリングは、デフォルトでアクティブになります。プロファイリングを非アクティブにすると、監視のオーバーヘッドは若干減ります。
7. サーバーを再起動します。

▼ CLI から統計情報を有効にする

1. 統計情報の収集を有効にするには、次の **CLI** コマンドを入力します。

   ./wadm set-stats-prop --user=*admin_user* **–password-file=** *password-file* **--config=***myconfig* **enabled=true**

   統計情報を無効にするには、enabled を false に設定します。

2. 間隔を設定したり、プロファイリングを有効にしたりするには、set-stats-prop interval および profiling プロパティーを使用します。詳細については、set-stats-prop のヘルプを参照してください。

3 サーバーを再起動します。

管理コンソールを使用した現在のアクティビティーの監視

頻繁に使用される統計情報は管理コンソールから表示でき、一般統計、インスタンスの統計、および仮想サーバーの統計として表示されます。

▼ 管理コンソールから統計情報を監視する

1 管理コンソールの「共通操作」ページで、「監視」タブを選択します。

2 構成を選択します。
構成の統計が表示されます。

3 ドロップダウンリストの「ビュー」から間隔を選択します。
ブラウザに表示される統計情報は、この間隔で自動的に更新されます。

4 表示する統計情報の種類を選択します。
統計情報の種類の初期リストには、「一般統計」、「インスタンスの統計」、および「仮想サーバーの統計」が含まれています。

「インスタンスの統計」を選択する場合は、監視するインスタンスの名前をクリックします。それにより、プロセスやセッションレプリケーションに関する情報を含む、詳細な統計情報が表示されます。

「仮想サーバーの統計」を選択する場合は、監視する仮想サーバーの名前をクリックします。応答の統計や Web アプリケーションの統計を含む、仮想サーバーの統計情報が表示されます。この情報は、`perfdump` では表示されません。

CLI を使用した現在のアクティビティーの監視

`wadm` コマンド `get-config-stats`、`get-virtual-server-stats`、`get-webapp-stats`、および `get-servlet-stats` を使用して統計情報を表示することもできます。次の例には、可能性のあるすべてのコマンドオプションは含まれていないことに注意してください。完全な構文については、コマンドのヘルプを参照してください。

▼ CLI から統計情報を監視する

1 1 つのノードに配備されている構成の統計情報を取得するには、次のコマンドを入力します。

./wadm get-config-stats --user=*admin-user* **--password-file=***admin-password-file* **--config=** *config-name* **--node=***node-name*

この構文で node オプションを使用すると、出力が 1 つのノードに制限されます。構成レベルの統計情報を取得するには、このコマンドを node オプションを指定せずに使用します。

1 つのノードの出力の例を次に示します。

```
timeStarted=1168035653
secondsRunning=1404
countRequests=690546
rpsLast1MinAvg=4491.7666
rpsLast5MinAvg=1844.6061
rpsLast15MinAvg=637.37305
countErrors=0
epsLast1MinAvg=0.0
epsLast5MinAvg=0.0
epsLast15MinAvg=0.0
maxResponseTime=0.30789953
rtLast1MinAvg=5.3970284
rtLast5MinAvg=5.208407
rtLast15MinAvg=35.56042
countBytesReceived=96800935
countBytesTransmitted=689929574
countChildDied=0
countVirtualServers=2
instanceName=https-test
process.1.countThreadPools=2
process.1.jdbcPoolCount=1
process.1.countThreads=64
process.1.fractionSystemMemoryUsage=2887.0
process.1.countConnectionQueues=1
process.1.sizeResident=0
process.1.countIdleThreads=32
process.1.mode=1
process.1.sizeVirtual=0
process.1.countConfigurations=1
process.1.pid=15874
process.1.timeStarted=Jan 5, 2007 2:20:53 PM
process.1.DNSCache.countCacheHits=687804
process.1.DNSCache.countAsyncNameLookup=0
process.1.DNSCache.countAsyncLookupsInProgress=0
process.1.DNSCache.flagAsyncEnabled=false
process.1.DNSCache.countAsyncAddrLookups=0
```

```
process.1.DNSCache.flagCacheEnabled=true
process.1.DNSCache.countCacheMisses=75
process.1.JDBCPool.1.countQueued=32
process.1.JDBCPool.1.countFreeConnections=0
process.1.JDBCPool.1.peakConnections=32
process.1.JDBCPool.1.millisecondsPeakWait=72
process.1.JDBCPool.1.countWaitQueueTimeouts=288
process.1.JDBCPool.1.peakQueued=64
process.1.JDBCPool.1.maxConnections=32
process.1.JDBCPool.1.currentConnections=32
process.1.JDBCPool.1.millisecondsAverageQueued=1.0
process.1.JDBCPool.1.countTotalFailedValidationConnections=0
process.1.JDBCPool.1.countLeasedConnections=32
process.1.JDBCPool.1.countTotalLeasedConnections=414
process.1.JDBCPool.1.countConnectionIdleTimeouts=1
process.1.JDBCPool.1.name=jdbc/jdbc-simple_1
process.1.connectionQueue.1.countQueued15MinuteAverage=4.3203125
process.1.connectionQueue.1.countQueued=0
process.1.connectionQueue.1.countQueued1MinuteAverage=0.046875
process.1.connectionQueue.1.countTotalQueued=79171
process.1.connectionQueue.1.countQueued5MinuteAverage=4.03125
process.1.connectionQueue.1.countOverflows=0
process.1.connectionQueue.1.maxQueued=1288
process.1.connectionQueue.1.ticksTotalQueued=724956383
process.1.connectionQueue.1.countTotalConnections=863
process.1.connectionQueue.1.peakQueued=64
process.1.connectionQueue.1.name=cq1
process.1.fileCache.countContentMisses=7
process.1.fileCache.maxMmapCacheSize=0
process.1.fileCache.sizeHeapCache=27520
process.1.fileCache.countMisses=22
process.1.fileCache.countContentHits=620662
process.1.fileCache.maxEntries=1024
process.1.fileCache.flagEnabled=true
process.1.fileCache.secondsMaxAge=30
process.1.fileCache.sizeMmapCache=0
process.1.fileCache.countInfoHits=1862013
process.1.fileCache.maxHeapCacheSize=10747924
process.1.fileCache.countOpenEntries=0
process.1.fileCache.countHits=2482682
process.1.fileCache.maxOpenEntries=1024
process.1.fileCache.countEntries=12
process.1.fileCache.countInfoMisses=19
process.1.jvm.countGarbageCollections=96
process.1.jvm.sizeHeap=67762048
process.1.jvm.countThreads=79
process.1.jvm.countClassesUnloaded=0
process.1.jvm.vMVendor=Sun Microsystems Inc.
```

```
process.1.jvm.countTotalClassesLoaded=3170
process.1.jvm.vMName=Java HotSpot(TM) Server VM
process.1.jvm.countTotalThreadsStarted=81
process.1.jvm.countClassesLoaded=3170
process.1.jvm.peakThreads=79
process.1.jvm.millisecondsGarbageCollection=1981
process.1.jvm.vMVersion=1.5.0_09-b03
process.1.keepalive.countConnections=32
process.1.keepalive.maxConnections=200
process.1.keepalive.countFlushes=0
process.1.keepalive.countRefusals=0
process.1.keepalive.countTimeouts=6
process.1.keepalive.countHits=686943
process.1.keepalive.secondsTimeout=30
process.1.threadPool.1.countQueued=0
process.1.threadPool.1.countThreadsIdle=1
process.1.threadPool.1.threadPoolId=NativePool
process.1.threadPool.1.maxThreads=128
process.1.threadPool.1.countThreads=1
process.1.threadPool.1.maxQueued=0
process.1.threadPool.1.peakQueued=0
process.1.threadPool.1.name=NativePool
process.1.threadPool.2.countQueued=0
process.1.threadPool.2.countThreadsIdle=1
process.1.threadPool.2.threadPoolId=my-custom-pool
process.1.threadPool.2.maxThreads=128
process.1.threadPool.2.countThreads=1
process.1.threadPool.2.maxQueued=0
process.1.threadPool.2.peakQueued=0
process.1.threadPool.2.name=my-custom-pool
```

2 仮想サーバーの統計情報を取得するには、次のコマンドを入力します。

./wadm get-virtual-server-stats --user=*admin-user* **--password-file=***admin-password-file* **--config=** *config-name* **--vs=***virtual-server-name*

node オプションが使用されていないため、この構文によって、この構成が配備されているすべてのノードにわたる仮想サーバーの全体的な統計情報が得られます。node オプションを使用すると、出力が 1 つのノードに制限されます。

3 配備されている **Web** アプリケーションの統計情報を取得するには、次のコマンドを入力します。

./wadm get-webapp-stats --user=*admin-user* **--password-file=** *admin-password-file* **--config=***config-name* **--node=** *node-name* **--vs=***virtual-server-name* **--uri=** *URI*

この構文によって、特定のインスタンスの特定の仮想サーバーに配備されている特定の Web アプリケーションの統計情報が取得されます。特定の構成について、その構成が配備されているすべてのノードにわたる Web アプリケーションの全体的な統計情報を取得するには、このコマンドを node オプションを指定せずに使用します。

次の例は、URI hello の出力を示しています。

```
countActiveSessions=1
countExpiredSessions=0
countJsps=1
countRejectedSessions=0
countReloadedJsps=1
countSessions=1
peakActiveSessions=1
secondsSessionAliveAverage=0
secondsSessionAliveMax=0
uri=/hello
vsName=myvs.sun.com
```

stats.xml を使用した現在のアクティビティーの監視

stats-xml を使用して統計情報を表示することもできます。この場合は、統計情報が XML 形式で表示されます。stats-xml の出力は XML 形式であるため、各種のツールを使用して統計情報を容易に解析できます。URI からの stats-xml 出力の表示 (これは有効にする必要がある)、または CLI からの stats-xml 出力の表示 (これはデフォルトで有効になっている) のどちらかが可能です。

▼ 管理コンソールから stats-xml URI を有効にする

stats-xml URI を有効にすると、ブラウザを経由してサーバーの XML 形式の統計情報にアクセスできます。stats-xml URI を使用する場合は、管理サーバーが実行されていなくても統計情報にアクセスできることに注意してください。また、stats-xml URI がアクティブになっていると、アクセスを拒否するための予防策を取らないかぎり、ユーザーもサーバーの統計情報を表示できるようになります。

1. 「共通操作」ページで、左側のプルダウンメニューから構成を選択します。
2. 右側のプルダウンメニューから仮想サーバーを選択し、「仮想サーバーを編集」をクリックします。
3. 「サーバー設定」タブで、「監視設定」サブタブをクリックします。
4. 「**XML** レポート」の「有効」チェックボックスを選択します。
5. **URI** を指定します。たとえば、/stats-xml と入力します。
6. 「保存」をクリックします。
7. 構成を配備します。

8 stats-xml **URI** にアクセスします。次に例を示します。

`http://`*yourhost*`:` *port*`/stats-xml`

統計情報が XML 形式で表示されます。

▼ CLI から stats-xml URI を有効にする

1 次のコマンドを使用して stats-xml を有効にします。

./wadm enable-stats-xml --user=*admin-user* **--password-file=** *admin-password-file* **[--uri-prefix=***prefix***]--config=** *config-name* **--vs=***virtual-server-name*

uri-prefix オプションを使用して stats-xml URI を設定します。

2 wadm deploy-config コマンドを使用して構成を配備します。

3 stats-xml **URI** にアクセスします。次に例を示します。

`http://`*yourhost* `:`*port*`/stats-xml`

統計情報が XML 形式で表示されます。

▼ URI で表示される stats-xml 統計情報を制限する

stats-xml URI を変更することにより、その URI で表示されるデータを制限できます。

- 情報を制限するように stats-xml **URI** を変更するには、各要素を **0** または **1** に設定します。**0** に設定された要素は、stats-xml 出力には表示されません。次に例を示します。

  `http:// `*yourhost*`:`*port* `/stats-xml?thread=0&process=0`

  この構文では、stats-xml 出力が、スレッドとプロセスの統計情報を含まないように制限されます。デフォルトでは、すべての統計情報が有効になっています (1 に設定される)。

  ほとんどの統計情報はサーバーレベルで表示できますが、一部の統計情報はプロセスレベルで表示できます。

  stats-xml を制限するには、次の構文要素を使用します。

  - cache-bucket
  - connection-queue
  - connection-queue-bucket (プロセスレベル)
  - cpu-info
  - dns-bucket
  - jdbc-resource-bucket
  - keepalive-bucket
  - process

- profile
- profile-bucket (プロセスレベル)
- request-bucket
- servlet-bucket
- session-replication
- thread
- thread-pool
- thread-pool-bucket (プロセスレベル)
- virtual-server
- web-app-bucket

▼ CLI から stats-xml 出力を表示する

URI に加えて、コマンド行インタフェース経由でも stats-xml 出力にアクセスできます。これはデフォルトで有効になっています。URI 経由の stats-xml 出力の表示とは異なり、コマンド行で stats-xml 出力を表示するには、管理サーバーが実行されている必要があります。ただし、要求を処理するスレッドがサーバー内で (たとえば、ビジー状態のために) ハングアップしており、URI を使用できない場合でも、引き続き CLI 経由で stats-xml 出力にアクセスできます。

- コマンド行インタフェース経由で stats-xml 出力を表示するには、次のコマンドを入力します。

  ./wadm get-stats-xml --user=*admin-user* **--password-file=** *admin-password-file* **--config=***config-name* **--node=** *node-name*

perfdump を使用した現在のアクティビティーの監視

perfdump ユーティリティーは Web Server に組み込まれた Server Application Function (SAF) であり、Web Server の内部統計情報からさまざまなパフォーマンスデータを収集し、ASCII テキストで表示します。perfdump 出力には、コマンド行の統計情報または管理コンソールで表示可能なすべての統計情報は表示されませんが、依然として便利なツールです。たとえば、管理サーバーが実行されていない場合でも、引き続き perfdump を使用できます。CLI 経由の perfdump 出力の表示 (これはデフォルトで有効になっている)、または URI 経由の perfdump 出力の表示 (これは有効にする必要がある) のどちらかが可能です。URI を有効にする場合は、perfdump URI へのアクセスを制御してください。そうしないと、ユーザーがその URI を表示できるようになります。

perfdump ユーティリティーでは、統計情報が統合されます。単一のプロセスを監視するのではなく、統計情報をプロセス数で乗算するため、サーバーの全体像を正確に把握することができます。

perfdump ユーティリティーで表示される情報のチューニングについては、53 ページの「監視データに基づくサーバーのチューニング」を参照してください。

▼ 管理コンソールから perfdump URI を有効にする

管理コンソール経由で、仮想サーバーの perfdump URI を有効にすることができます。

注 – perfdump で表示される統計情報は、そのサーバー全体の情報です。ある仮想サーバーで perfdump を有効にすると、個別の仮想サーバーではなく、サーバー全体の統計情報が表示されます。

---

1. 「共通操作」で、構成を選択します。
2. 仮想サーバーを選択し、「仮想サーバーを編集」をクリックします。
3. 「監視設定」タブをクリックします。
4. 「プレーンテキストレポート」の「有効」チェックボックスを選択します。
5. レポートにアクセスするための **URI** を指定します。たとえば、/.perf とします。
6. 「保存」をクリックします。
7. 構成を配備します。
8. perfdump にアクセスするには、仮想サーバー上の **URI** にアクセスします。

   次に例を示します。http://localhost:80/.perf

   perfdump 統計情報を要求し、統計情報がブラウザで自動的に更新される頻度 (秒) を指定できます。次の例では、更新が 5 秒ごとに設定されています。

   http:// *yourhost*/.perf?refresh=5

▼ CLI から perfdump URI を有効にする

1. 次のコマンドを使用して stats-xml を有効にします。

   ./wadm enable-perfdump --user=*admin-user* **--password-file=** *admin-password-file* **[--uri=***uri***]--config=** *config-name***--vs=***virtual-server-name*

   URI オプションを使用して pefdump URI を設定します。

2. wadm deploy-config コマンドを使用して構成を配備します。

3. perfdump にアクセスするには、仮想サーバー上の **URI** にアクセスします。
   次に例を示します。`http://localhost:80/.perf`

   perfdump 統計情報を要求し、統計情報がブラウザで自動的に更新される頻度 (秒) を指定できます。次の例では、更新が 5 秒ごとに設定されています。

   `http://` *yourhost*`/.perf?refresh=5`

▼ CLI から perfdump データを表示する

URI に加えて、コマンド行インタフェース経由でも perfdump 出力にアクセスできます。これはデフォルトで有効になっています。URI 経由の perfdump 出力の表示とは異なり、コマンド行で perfdump 出力を表示するには、管理サーバーが実行されている必要があります。ただし、要求を処理するスレッドがサーバー内で (たとえば、ビジー状態のために) ハングアップしており、URI を使用できない場合でも、引き続き CLI 経由で perfdump 出力にアクセスできます。

- コマンド行インタフェース経由で perfdump 出力を表示するには、次のコマンドを入力します。

  ./wadm get-perfdump --user=*admin-user* **--password-file=** *admin-password-file* **--config=***config-name* **--node=** *node-name*

  この出力は、コマンドウィンドウに表示されます。

サンプルの perfdump 出力

次に perfdump の出力例を示します。

```
webservd pid: 29133

Sun Java System Web Server 7.0 B07/13/2006 17:09 (SunOS DOMESTIC)

Server started Fri Jul 14 14:34:15 2006
Process 29133 started Fri Jul 14 14:34:17 2006

ConnectionQueue:
----------------------------------------
Current/Peak/Limit Queue Length           2/237/1352
Total Connections Queued                  67364017
Average Queue Length (1, 5, 15 minutes)   4.52, 4.73, 4.85
Average Queueing Delay                    13.63 milliseconds

ListenSocket ls1:
-----------------------
Address                  https://0.0.0.0:2014
Acceptor Threads         1
Default Virtual Server   https-test
```

```
KeepAliveInfo:
--------------------
KeepAliveCount        198/200
KeepAliveHits         0
KeepAliveFlushes      0
KeepAliveRefusals     56844280
KeepAliveTimeouts     365589
KeepAliveTimeout      10 seconds

SessionCreationInfo:
------------------------
Active Sessions           128
Keep-Alive Sessions       0
Total Sessions Created    128/128

Server cache disabled

Native pools:
----------------------------
NativePool:
Idle/Peak/Limit               1/1/128
Work Queue Length/Peak/Limit  0/0/0
TestPool:
Idle/Peak/Limit               5/5/10
Work Queue Length/Peak/Limit  0/0/15

DNSCacheInfo:
-----------------
enabled             yes
CacheEntries        4/1024
HitRatio            62854802/62862912 ( 99.99%)

Async DNS disabled

Performance Counters:
-----------------------------------------------
                           Average         Total      Percent

Total number of requests:               62647125
Request processing time:    0.0343  2147687.2500

default-bucket (Default bucket)
Number of Requests:                     62647125    (100.00%)
Number of Invocations:                3374170785    (100.00%)
Latency:                    0.0008    47998.2500    (  2.23%)
Function Processing Time:   0.0335  2099689.0000    ( 97.77%)
Total Response Time:        0.0343  2147687.2500    (100.00%)
```

```
Sessions:
-------------------------------------------------------------------------------------------------------------
Process  Status    Client     Age  VS          Method URI                                          Function

29133    response  192.6.7.7  115  https-test  GET   /qa_webapp/CheckNetwork.class                 service-j2ee
29133    response  192.6.7.7  8    https-test  GET   /qa_webapp/CheckNetwork.class                 service-j2ee
29133    response  192.6.7.7  4    https-test  GET   /qa_webapp/CheckNetwork.class                 service-j2ee
29133    response  10.5.8.19  4    https-test  GET   /perf                                         service-dump
29133    response  192.6.7.7  3    https-test  GET   /qa_webapp/CheckNetwork.class                 service-j2ee
29133    response  192.6.7.7  3    https-test  GET   /qa_webapp/CheckNetwork.class                 service-j2ee
29133    response  192.6.7.7  2    https-test  GET   /qa_webapp/CheckNetwork.class                 service-j2ee
29133    response  192.6.7.7  2    https-test  GET   /qa_webapp/CheckNetwork.class                 service-j2ee
29133    response  192.6.7.7  2    https-test  GET   /qa_webapp/CheckNetwork.class                 service-j2ee
29133    response  192.6.7.7  2    https-test  GET   /qa_webapp/CheckNetwork.class                 service-j2ee
29133    request   192.6.7.7  0
29133    request   192.6.7.7  0
29133    request   192.6.7.7  0
29133    request   192.6.7.7  0
29133    request   192.6.7.7  0
29133    response  192.6.7.7  0    https-test  GET   /file1.shtml                                  shtml_send
29133    request   192.6.7.7  0
29133    request   192.6.7.7  0
29133    response  192.6.7.7  0    https-test  GET   /find-pathinfo-forward/pathinfo.pl/p/info     send-cgi
29133    request   192.6.7.7  0
29133    updating  192.6.7.7
29133    updating  192.6.7.7
29133    updating  192.6.7.7
29133    updating  192.6.7.7
.
.
.
```

パフォーマンスバケットの使用

パフォーマンスバケットを使用すると、バケットを定義し、それらのバケットをさまざまなサーバー関数にリンクできます。これらの関数のいずれかを呼び出すごとに、サーバーは統計データを収集し、それをバケットに追加します。たとえば、`send-cgi` と `service-j2ee` はそれぞれ、CGI と Java サーブレットの要求にサービスを提供するために使用される関数です。2 つのバケットを定義して CGI とサーブレットの要求に対して別々のカウンタを保持するか、または両方のタイプの動的コンテンツに対する要求をカウントするバケットを 1 つ作成することができます。この情報を収集するためにかかる負担は最小限で済み、サーバーパフォーマンスへの影響も通常はわずかです。この情報へはあとで `perfdump` ユーティリティーを使用してアクセスできます。バケット内には、次の情報が格納されます。

- バケットの名前**:** この名前によって、バケットが関数に関連付けられます。
- 説明**:** このバケットが関連付けられている関数の説明。

- この関数に対する要求の数: この関数の呼び出しを引き起こした要求の総数。
- 関数が呼び出された回数: 1 つの要求に対して複数回実行される関数もあるため、この回数は、関数に対する要求の数とは一致しない可能性があります。
- 関数の待ち時間またはディスパッチ時間: サーバーが関数を呼び出すためにかかった時間。
- 関数の時間: 関数自体で費やされた時間。

  `default-bucket` は、サーバーで事前に定義されています。。ここには、ユーザーが定義したどのバケットにも関連付けられていない関数の統計情報が記録されます。

構成

パフォーマンスバケットのすべての設定情報を、`magnus.conf` および `obj.conf` ファイルに指定してください。自動的に有効になるのは、`default-bucket` だけです。

最初に、パフォーマンス統計の収集と `perfdump` を有効にする必要があります。

次の例は、`magnus.conf` で新しいバケットを定義する方法を示しています。

```
Init fn="define-perf-bucket" name="acl-bucket" description="ACL bucket"

Init fn="define-perf-bucket" name="file-bucket" description="Non-cached responses"

Init fn="define-perf-bucket" name="cgi-bucket" description="CGI Stats"
```

前述の例では、次の 3 つのバケットを作成しています。`acl-bucket`、`file-bucket`、および `cgi-bucket`。これらのバケットを関数に関連付けるには、パフォーマンスを測定する `obj.conf` 関数に `bucket=`*bucket-name* を追加します。

例

```
PathCheck fn="check-acl" acl="default" bucket="acl-bucket"
...
Service method="(GET|HEAD|POST)" type="*~magnus-internal/*"
fn="send-file" bucket="file-bucket"
...
<Object name="cgi">
ObjectType fn="force-type" type="magnus-internal/cgi"
Service fn="send-cgi" bucket="cgi-bucket"
</Object>
```

詳細については、『Sun Java System Web Server 7.0 Administrator's Configuration File Reference』の「The bucket Parameter」を参照してください。

パフォーマンスレポート

バケットのサーバー統計情報には、perfdump ユーティリティーを使用してアクセスできます。パフォーマンスバケット情報は、perfdump によって返されるレポートの最後のセクションに配置されています。

レポートには次の情報が含まれています。

- 平均 (Average)、合計 (Total)、パーセント (Percent) の各列には、要求された各統計情報のデータが表示されます。
- 要求処理時間 (Request Processing Time) はサーバーがそれまでに受信したすべての要求を処理するために要した合計時間です。
- 要求数 (Number of Requests) は関数の要求の合計数です。
- 呼び出し数 (Number of Invocations) は、関数が呼び出された合計回数です。1 つの要求の処理中に、関数が何度も呼び出される場合があるため、呼び出された回数は要求の数とは異なります。この行のパーセント列は、すべてのバケットの呼び出し回数の合計に基づいて計算されます。
- 遅延 (Latency) は Web Server が関数の呼び出しに要した時間 (秒) です。
- 関数処理時間 (Function Processing Time) は Web Server が関数の処理に費やした時間 (秒) です。Function Processing Time と Total Response Time のパーセンテージは、Request Processing Time の合計に基づいて計算されます。
- 合計応答時間 (Total Response Time) は Function Processing Time と Latency の合計 (秒) です。

perfdump で表示可能なパフォーマンスバケット情報の例を次に示します。

```
Performance Counters:
------------------------------------------------
                           Average         Total      Percent

Total number of requests:                62647125
Request processing time:    0.0343  2147687.2500

default-bucket (Default bucket)
Number of Requests:                      62647125    (100.00%)
Number of Invocations:                 3374170785    (100.00%)
Latency:                    0.0008    47998.2500    (  2.23%)
Function Processing Time:   0.0335  2099689.0000    ( 97.77%)
Total Response Time:        0.0343  2147687.2500    (100.00%)
```

Java ES 監視コンソールを使用した現在のアクティビティーの監視

Web Server 管理コンソールとコマンド行インタフェースで表示可能な統計情報は、Java ES 監視コンソールでも表示できます。情報は同じですが、CMM (Common Monitoring Data Model) を使用した別の形式で提供されます。このマニュアルでは、Web Server で利用可能なツールを使用した監視について説明していますが、Java ES 監視ツールを使用してサーバーを監視することもできます。Java ES 監視ツールの使用の詳細については、『Sun Java Enterprise System 5 Monitoring Guide 』を参照してください。使用する監視方法には関係なく、サーバーのチューニングには同じ設定を使用します。

◆ ◆ ◆ 第 2 章

Sun Java System Web Server のチューニング

この章では、Sun Java System Web Server のパフォーマンス改善のために実施可能な具体的な調整について説明します。チューニング設定に関する理解を深められるよう、Web Server の接続処理プロセスの概要について説明します。この章の内容は、次のとおりです。

注 – サーバーのチューニング時には細心の注意を払うようにしてください。変更を加える前に必ず、構成ファイルのバックアップを作成します。

一般的なチューニングのヒント

サーバーをチューニングする際には、ユーザー固有の環境は一意であることを忘れないことが重要です。このガイドに含まれる提案の影響は、ユーザー固有の環境に応じて異なります。最終的には、自身の判断や観察に基づいて最適な調整を選択する必要があります。

パフォーマンスの最適化作業を行う際には、次の指針を参考にしてください。

- 規則的に作業を行う

可能なかぎり、一度に 1 つの調整を行うようにしてください。変更を行うたびにその前後のパフォーマンスを測定し、測定可能な改善がみられない場合は常に、その変更を取り消します。

- 少しずつ調整する

  定量的なパラメータを調整する際には、一度に大幅な変更を行おうとするのではなく、何回かのステップに分けて継続的に変更を行うようにしてください。システムが異なれば直面する状況も異なります。したがって、この値をあまりにも急激に変更すると、システムの最適な設定値をスキップしてしまう恐れがあります。

- 一から見直す

  ハードウェアまたはソフトウェアのアップグレード、新しい主要アプリケーションの配備など、システムに大きな変更を加えた場合には必ず、それまでに行ったすべての調整を見直し、それらの調整が依然として有効であることを確認してください。Solaris のアップグレード後は、変更されていない `/etc/system` ファイルを使って最初から作業を開始するべきです。

- 継続的に最新情報を得る

  システムをアップグレードするときには必ず、『Sun Java System Web Server 7.0 リリースノート (UNIX 版)』およびオペレーティングシステムのリリースノートを参照してください。リリースノートには多くの場合、特定の調整に関する更新情報が含まれています。

スレッド、プロセス、および接続の理解

サーバーをチューニングする前に、Web Server の接続処理プロセスを理解しておくべきです。この節では、次の内容について説明します。

接続処理の概要

Web Server では、待機ソケット上のアクセプタスレッドが接続を受け付け、それらを接続キューに入れます。次に、スレッドプール内の要求処理スレッドがキューから接続を取り出し、その要求を処理します。

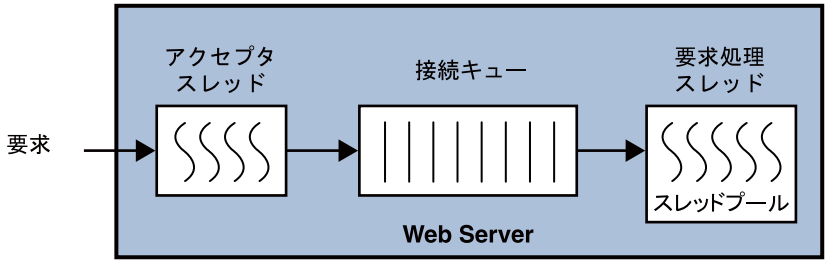


図 2–1 Web Server の接続処理

また、別のスレッドプールに要求を送信して処理を依頼するように、要求処理スレッドに指示することもできます。たとえば、要求処理スレッドがスレッドに対して安全でないある作業を実行する必要がある場合、その処理の一部を NativePool に送信するように指示することができます。NativePool は作業が完了すると、その結果を要求処理スレッドに伝え、要求処理スレッドが要求の処理を続行します。

サーバーの起動時に作成されるのは、スレッドプールの最小スレッド数に定義された数のスレッドだけであり、これはデフォルトで 16 個です。負荷が増えるとサーバーによって追加のスレッドが作成されます。新しいスレッドの追加ポリシーは、接続キューの状態に基づいています。

新しい接続が返されるたびに、キュー内で待機している接続の数 (接続のバックログ) が、すでに作成された要求処理スレッドの数と比較されます。待機している接続の数がスレッド数を上回った場合、次回の要求完了時に新しいスレッドが追加されるようにスケジュールされます。

新しいセッションスレッドの追加プロセスは、最大スレッド数の値に厳密に制限されます。最大スレッド数の詳細については、64 ページの「最大スレッド数 (最大同時要求数)」を参照してください。

スレッド、プロセス、および接続の数やタイムアウトに影響を与える設定を変更するには、管理コンソールで、構成の「パフォーマンス」タブ (HTTP 設定) および「HTTP リスナー」タブを使用します。また、`wadm` コマンド `set-thread-pool-prop`、`set-http-listener-prop`、および `set-keep-alive-prop` を使用することもできます。

短待ち時間モードと高並行性モード

サーバーは、負荷に応じて 2 つのモードのいずれかで稼働できます。負荷の増減にもっとも効率的に対応できるようにモードを切り替えます。

- 短待ち時間モードでは、キープアライブ接続に対して、セッションスレッド自身が新しい要求のポーリングを行う。

- 高並行性モードでは、要求の処理が完了すると、セッションスレッドがその接続をキープアライブサブシステムに渡す。高並行性モードでは、キープアライブサブシステムがすべてのキープアライブ接続に対して新しい要求のポーリングを行う。

サーバーの起動時には、短待ち時間モードが使用されます。負荷が増えると、サーバーは高並行性モードに移行します。短待ち時間モードから高並行性モードに移行したり、元の短待ち時間モードに戻ったりする判断は、接続キューの長さ、平均セッション合計数、平均アイドルセッション数、および現在のアクティブセッション数とアイドルセッション数に基づいて、サーバーによって行われます。

無効化されたスレッドプール

スレッドプールが無効化されると、スレッドがプール内に作成されず、接続キューやキープアライブスレッドも作成されません。スレッドプールが無効になると、アクセプタスレッド自身が要求を処理します。

magnus.conf の NSAPI 向けの接続処理指令

前述した設定に加え、`magnus.conf` ファイル内で次の指令を編集すれば、NSAPI プラグイン向けの要求処理を追加で構成できます。

- `KernelThreads` – カーネルによってスケジュールされたスレッド上で NSAPI プラグインを常に実行するかどうかを決定します (Windows のみ)
- `TerminateTimeout` – サーバーの停止時に、NSAPI プラグインが要求の処理を完了するのをサーバーが最大でどれだけの時間待つかを決定します

これらの指令の詳細については、『Sun Java System Web Server 7.0 Administrator's Configuration File Reference』を参照してください。

注 – `magnus.conf` などの構成ファイルを編集するためのもっとも安全な方法は、`wadm` コマンド `get-config-file` と `set-config-file` を使って編集用のローカルコピーを取得し、それを Web Server に戻すことです。これらのコマンドの詳細については、各コマンドのヘルプを参照してください。

カスタムスレッドプール

デフォルトでは、接続キューはデフォルトのスレッドプールに要求を送信します。ただし、`magnus.conf` 内でスレッドプールの `Init` 関数を使ってユーザー独自のスレッドプールを作成することもできます。これらのカスタムスレッドプールは、要求の全体を処理するためではなく、NSAPI サービスアプリケーション関数 (SAF) を実行するために使用されます。

SAF がカスタムスレッドプールの使用を必要とする場合、現在の要求処理スレッドは、要求をキューに入れ、カスタムスレッドプール内の別のスレッドがその SAF を完了するまで待機したあと、その要求の残りの部分を完了します。

たとえば、`obj.conf` ファイルの内容が次のようになっているとします。

```
NameTrans fn="assign-name" from="/testmod" name="testmod" pool="my-custom-pool"
...
<Object name="testmod">
ObjectType fn="force-type" type="magnus-internal/testmod"
Service method=(GET|HEAD|POST) type="magnus-internal/testmod"
fn="testmod_service" pool="my-custom-pool2"
</Object>
```

この例の場合、要求は次のように処理されます。

1. 要求処理スレッド (この例では A1 と呼ぶ) が要求を取り出し、NameTrans 指令の前の手順を実行します。
2. URI が `/testmod` で始まっている場合、A1 スレッドはその要求を `my-custom-pool` キューに入れます。A1 スレッドは待機します。
3. `my-custom-pool` 内の別のスレッドをこの例では B1 スレッドと呼びますが、A1 によってキューに入れられた要求をその B1 が取り出します。B1 がこの要求の処理を完了し、待機段階に戻ります。
4. A1 スレッドが呼び起こされ、その要求の処理を続行します。これは、ObjectType SAF を実行したあと、Service 関数に移ります。
5. この Service 関数は `my-custom-pool2` 内のスレッドによって処理される必要があるため、A1 スレッドが要求を `my-custom-pool2` キューに入れます。
6. `my-custom-pool2` 内の別のスレッドをこの例では C1 と呼びますが、キューに入れられた要求をその C1 が取り出します。C1 がこの要求の処理を完了し、待機段階に戻ります。
7. A1 スレッドが呼び起こされ、その要求の処理を続行します。

この例では、3 つのスレッド A1、B1、および C1 が動作して要求の処理を完了させます。

追加のスレッドプールは、スレッドに対して安全でないプラグインを実行するための手段の 1 つです。最大スレッド数が 1 に設定されたプールを定義すると、指定されたサービス関数内で 1 つの要求しか実行できなくなります。前述の例で `testmod_service` がスレッドに対して安全でない場合には、そのサービスを単一のスレッドを使って実行する必要があります。`my-custom-pool2` 内に単一のスレッドを作成すれば、この SAF がマルチスレッド化された Web Server 内で正常に動作するようになります。

スレッドプールの定義方法の詳細については、『Sun Java System Web Server 7.0 Administrator's Configuration File Reference』の「thread-pool-init」を参照してください。

ネイティブスレッドプール

Windows 上では、実行にネイティブスレッドを必要とする NSAPI 関数を実行する目的で、ネイティブスレッドプール (NativePool) がサーバーによって内部的に使用されます。

Web Server は NSPR (Netscape Portable Runtime) を使用します。これは移植性を高めるための基盤層であり、ホスト OS のサービスへのアクセス機能を提供します。この層はスレッドの抽象化を提供しますが、それは必ずしも、OS が提供するスレッドに対する抽象化と同一であるとは限りません。これらの非ネイティブスレッドのスケジューリングオーバーヘッドは比較的低いので、それらを使用すればパフォーマンスを改善できます。ただし、これらのスレッドは、入出力呼び出しなど、OS に対するブロック呼び出しの影響を受けやすい性質を持っています。ブロック呼び出しを使用する可能性のある NSAPI 拡張の記述を容易にするために、サーバーはブロック呼び出しを安全にサポートするスレッドのプールを保持します。これらのスレッドは通常、ネイティブ OS スレッドになります。要求の処理中に、非ネイティブスレッド上で安全に実行できる関数としてマークされていない NSAPI 関数はすべて、ネイティブスレッドプール内のいずれかのスレッド上で実行されるようにスケジュールされます。

ユーザーが `NameTrans`、`Service`、`PathCheck` 関数など、独自の NSAPI プラグインを記述した場合、それらの関数はデフォルトで、ネイティブスレッドプール内のスレッド上で実行されます。プラグインが NSAPI 関数を入出力のためだけに使用しているか、あるいは NSAPI 入出力関数をまったく使用していない場合、そのプラグインは非ネイティブスレッド上で実行できます。そのためには、その関数の読み込み時に `NativeThread="no"` オプションを指定し、ネイティブスレッドを必要としないことを示す必要があります。

たとえば、`magnus.conf` ファイル内の `load-modules` `Init` 行に、次のコードを追加します。

```
Init funcs="pcheck_uri_clean_fixed_init" shlib="C:/Sun/webserver7/lib/custom.dll"
fn="load-modules" NativeThread="no"
```

`NativeThread` フラグは `funcs` リスト内のすべての関数に影響を与えます。したがって、ライブラリ内に関数が 2 つ以上存在しているが、ネイティブスレッドを使用するのはその一部だけである場合には、複数の `Init` 行を使用してください。`NativeThread` を `yes` に設定すると、スレッドは直接 OS スレッドにマップされます。

`load-modules` 関数については、『Sun Java System Web Server 7.0 Administrator's Configuration File Reference』の「load-modules」を参照してください。

プロセスのモード

注 – マルチプロセスモードは非推奨です。したがって、次の各節の情報は、下位互換性を維持するためだけに含まれています。マルチプロセスモードが非推奨になっているのは、今日では大部分のアプリケーションがマルチスレッド化されており、マルチプロセスモードは通常、必要とされないからです。

Sun Java System Web Server は、次の 2 つのモードのいずれかで実行できます。

- 49 ページの「シングルプロセスモード」
- 49 ページの「マルチプロセスモード」

シングルプロセスモード

シングルプロセスモードでは、サーバーは Web クライアントからの要求を単一のプロセスを使って受信します。単一サーバープロセスの内側ではアクセプタスレッドが実行されており、新しい要求が到着するのを待っています。要求が到着すると、アクセプタスレッドが接続を受け付け、その要求を接続キューに入れます。要求処理スレッドが接続キューから要求を取り出し、その要求を処理します。

サーバーはマルチスレッド化されているため、サーバー用に記述された NSAPI 拡張はすべて、スレッドに対して安全でなければいけません。つまり、NSAPI 拡張がファイルへの共有参照やグローバル変数など、あるグローバルリソースを使用している場合には、一度に 1 つのスレッドしかそのリソースにアクセスしないように、そのリソースの使用を同期させる必要があります。Web Server に付属するプラグインはすべて、スレッドに対して安全であり、かつスレッドを認識するため、高いスケーラビリティーと並行性を実現できます。これに対し、旧バージョンのアプリケーションはシングルスレッドである可能性があります。サーバーがそうしたアプリケーションを実行する場合、一度に 1 つしか実行できません。このため、高負荷下ではサーバーパフォーマンスの問題が発生します。残念ながら、シングルプロセスの設計では、現実的な回避方法は存在しません。

マルチプロセスモード

複数のプロセスと各プロセス内の複数のスレッドを使って要求を処理するように、サーバーを構成することができます。この柔軟性により、スレッドを使用するサイトで最適なパフォーマンスを実現できるだけでなく、スレッド化された環境内で実行する準備が整っていない旧バージョンのアプリケーションを実行するサイトで、下位互換性を維持することができます。Windows 上のアプリケーションは一般に、すでにマルチスレッドを活用しているため、この機能は UNIX および Linux プラットフォームに適用されます。

マルチプロセスの利点は、スレッドを認識しないかスレッドに対して安全でない旧バージョンのアプリケーションを、Sun Java System Web Server 内でより効果的に実行できるという点です。ただし、Sun Java System 拡張はどれもシングルプロセスのス

レッド化環境をサポートするように構築されているため、それらの拡張がマルチプロセスモードでは動作しない可能性があります。サーバーがマルチプロセスモードになっていると検索プラグインの起動が失敗し、セッションレプリケーションが有効になっていると、マルチプロセスモードでのサーバーの起動が失敗します。

マルチプロセスモードの場合、サーバーは起動時に、複数のサーバープロセスを生成します。各プロセスには、受信要求を受け取るためのスレッドが、構成に応じて1つ以上含まれます。各プロセスは完全に独立しているため、グローバル変数、キャッシュ、およびその他のリソースの独自のコピーを持ちます。複数のプロセスを使用する場合、多くのリソースがシステムに必要です。また、共有状態を必要とするアプリケーションのインストールを試みる場合、そのアプリケーションは複数のプロセス間でその状態の同期を取る必要があります。NSAPI には、プロセス間の同期を実装するためのヘルパー機能は用意されていません。

`MaxProcs` 値として 1 より大きい値を指定すると、サーバーは、複数のサーバープロセス間で接続を分散する処理をオペレーティングシステムに依頼します (`MaxProcs` 指令については50 ページの「MaxProcs (UNIX/Linux)」を参照)。ただし、最近のオペレーティングシステムの多くは、同時接続数が少ない場合は特に、接続を均等に分散しません。

Sun Java System Web Server はサーバープロセス間での均等な負荷分散を保証できないため、スレッドに対して安全でない旧バージョンのアプリケーションに対応するために「最大スレッド数」を 1 に、`MaxProcs` を 1 より大きい値にそれぞれ設定すると、パフォーマンスの問題が発生する可能性があります。旧バージョンのアプリケーションにバックエンドデータベースが含まれている場合など、旧バージョンのアプリケーションが要求に応答するまでに長い時間がかかる場合には特に、この問題が顕著になります。このシナリオではおそらく、「最大スレッド数」にデフォルト値を使用し、旧バージョンのアプリケーションへのアクセスをスレッドプールを使って直列化するのが適切です。スレッドプールの作成方法の詳細については、『Sun Java System Web Server 7.0 Administrator's Configuration File Reference』の「thread-pool-init」を参照してください。

サーバー内で NSAPI を 1 つも実行しない場合には、次のデフォルト設定を使用するべきです。1 つのプロセスと多数のスレッド。スレッド化された環境内でのスケーラビリティーを持たないアプリケーションを実行する場合には、少数のプロセスと多数のスレッド、具体的には 4 個または 8 個のプロセスと、1 プロセスあたり 128 個または 512 個のスレッドなどを使用するべきです。

MaxProcs (UNIX/Linux)

注 – `MaxProcs` は非推奨であり、下位互換性を維持するためだけに含まれています。

UNIX/Linux サーバーをマルチプロセスモードに設定するには、この指令を使用します。このモードを使えば、マルチプロセッサマシン上でのスケーラビリティーが向上する可能性があります。これを 1 より小さい値に設定すると、その値は無視され、デフォルト値の 1 が使用されます。

MaxProcs の値を設定するには、magnus.conf 内の MaxProcs パラメータを編集します。

注 – サーバーを MaxProcs モードで実行すると、起動メッセージが重複して表示されます。

Web Server 6.1 チューニングパラメータから Web Server 7.0 へのマッピング

Web Server 6.1 で magnus.conf および nsfc.conf ファイルを編集してチューニング可能だったチューニングパラメータの多くは、server.xml ファイルに移されました。これらのチューニングパラメータは、管理コンソールとコマンド行インタフェースを使ってチューニング可能になりました。次の表では、いくつかのチューニングパラメータについて、Web Server 6.1 パラメータ、チューニングに使用する新しい server.xml の要素、およびユーザーインタフェース経由でのパラメータの変更方法を示します。server.xml ファイルを直接編集すると間違いが起こりやすいので、ユーザーインタフェースを使って値を設定することをお勧めします。server.xml のすべての要素の完全なリストについては、『Sun Java System Web Server 7.0 Administrator's Configuration File Reference』の第 3 章「Elements in server.xml」を参照してください。

表 2–1　server.xml へのパラメータマッピング

Web Server 6.1 のパラメータ	**Web Server 7.0** の server.xml の要素または属性	管理コンソールの場所	wadm コマンド
magnus.conf の AcceptTimeout	http 要素の io-timeout 要素	構成の「パフォーマンス」タブ ⇒「HTTP 設定」ページ	set-http-prop コマンドの io-timeout プロパティー
magnus.conf の ACLGroupCacheSize	acl-cache 要素の max-groups-per-user 要素	構成の「パフォーマンス」タブ ⇒「キャッシュ設定」ページ	set-acl-cache-prop コマンドの max-groups-per-user プロパティー
magnus.conf の ACLUserCacheSize	acl-cache 要素の max-users 要素	構成の「パフォーマンス」タブ ⇒「キャッシュ設定」ページ	set-acl-cache-prop コマンドの max-users プロパティー

表 2–1 server.xml へのパラメータマッピング (続き)

Web Server 6.1 のパラメータ	Web Server 7.0 の server.xml の要素または属性	管理コンソールの場所	wadm コマンド
magnus.conf の ConnQueueSize	thread-pool 要素の queue-size 要素	構成の「パフォーマンス」タブ ⇒「HTTP」タブ	set-thread-pool-prop コマンドの queue-size プロパティー
dns-cache-init Init SAF	dns-cache 要素の enabled 要素	構成の「パフォーマンス」タブ ⇒「DNS」タブ	set-dns-cache-prop コマンドの enabled プロパティー
dns-cache-init Init SAF のキャッシュサイズ	dns-cache 要素の max-entries 要素	構成の「パフォーマンス」タブ ⇒「DNS」タブ	set-dns-cache-prop コマンドの max-entries プロパティー
nsfc.conf の FileCacheEnabled	file-cache 要素の enabled 要素	構成の「パフォーマンス」タブ ⇒「キャッシュ」タブ	set-file-cache-prop コマンドの enabled プロパティー
magnus.conf の KeepAliveThreads	keep-alive 要素の threads 要素	構成の「パフォーマンス」タブ ⇒「HTTP」タブ	set-keep-alive-prop コマンドの threads プロパティー
magnus.conf の KeepAliveTimeout	keep-alive 要素の timout 要素	構成の「パフォーマンス」タブ ⇒「HTTP」タブ	set-keep-alive-prop コマンドの timeout プロパティー
magnus.conf の KernelThreads (Windows のみ)	変更なし		
magnus.conf の ListenQ	http-listener 要素の listen-queue-size 要素	構成の「HTTP リスナー」タブ	set-http-listener-prop コマンドの listen-queue-size
magnus.conf の LogVerbose	log 要素の log-level 要素	構成の「一般」タブ ⇒「ログ設定」	set-error-log-prop コマンドの log-level プロパティー
nsfc.conf ファイルの MaxAge	file-cache 要素の max-age 要素	構成の「パフォーマンス」タブ ⇒「キャッシュ」タブ	set-file-cache-prop コマンドの max-age プロパティー
nsfc.conf ファイルの MaxFiles	file-cache 要素の max-entries 要素	構成の「パフォーマンス」タブ ⇒「キャッシュ」タブ	set-file-cache-prop コマンドの max-entries プロパティー

表 2–1 server.xml へのパラメータマッピング (続き)

Web Server 6.1 のパラメータ	Web Server 7.0 の server.xml の要素または属性	管理コンソールの場所	wadm コマンド
magnus.conf の MaxKeepAliveConnections	keep-alive 要素の max-connections 要素	構成の「パフォーマンス」タブ ⇒「HTTP」タブ	set-keep-alive-prop コマンドの max-connections プロパティー
magnus.conf の MaxProcs	非推奨		
magnus.conf の NativePoolMaxThreads	変更なし		
magnus.conf の NativePoolMinThreads	変更なし		
magnus.conf の NativePoolQueueSize	変更なし		
magnus.conf の NativePoolStackSize	変更なし		
magnus.conf の RqThrottle	thread-pool 要素の max-threads 要素	構成の「パフォーマンス」タブ ⇒「HTTP」タブ	set-thread-pool-prop コマンドの max-threads プロパティー
magnus.conf の RqThrottleMin	thread-pool 要素の min-threads 要素	構成の「パフォーマンス」タブ ⇒「HTTP」タブ	set-thread-pool-prop コマンドの min-threads プロパティー
magnus.conf の TerminateTimeout	変更なし		

監視データに基づくサーバーのチューニング

この節では、管理コンソール、perfdump、コマンド行インタフェース、および stats-xml 経由で利用可能なパフォーマンス情報について説明します。ここでは、その情報を解析して一部のパラメータをチューニングすることで、サーバーのパフォーマンスを改善する方法について説明します。

デフォルトのチューニングパラメータはほとんどすべてのサイトに適していますが、扱うボリュームが非常に大きいサイトだけは例外です。大規模サイトでおそらく定期的に変更する必要のある設定は、スレッドプールとキープアライブの設定だけです。これらの設定を構成レベルでチューニングするには、管理コンソール、wadm コマンドのいずれかを使用します。server.xml ファイル内の要素を直接編集することでサーバーをチューニングすることも可能ですが、server.xml ファイルを直接編集すると作業が煩雑になる可能性があります。

perfdump は次のカテゴリの統計を監視しますが、これらの各カテゴリについては次の各節で説明します。ほとんどの場合、これらの統計は管理コンソール、コマンド行インタフェース、および stats-xml 出力にも表示されます。次の各節には、どの方法を使ってデータを監視するかにかかわらず、これらすべてのカテゴリのチューニング情報が含まれています。

- 54 ページの「接続キュー情報」
- 57 ページの「HTTP リスナー (待機ソケット) 情報」
- 58 ページの「キープアライブ情報」
- 63 ページの「セッション作成 (スレッド) 情報」
- 65 ページの「ファイルキャッシュ情報 (静的コンテンツ)」
- 71 ページの「スレッドプール情報」
- 74 ページの「DNS キャッシュ情報」

これに加え、管理コンソール、コマンド行インタフェース、および stats-xml 経由で表示される統計情報には、perfdump の出力に含まれない別のカテゴリも含まれています。それらの統計をチューニングする方法については、次の各節で説明します。

- 76 ページの「Java 仮想マシン (JVM) 情報」
- 77 ページの「Web アプリケーション情報」
- 78 ページの「JDBC リソース情報」

必要な統計を表示し終わったら、管理コンソールの「パフォーマンス」タブを使用することで、サーバーのパフォーマンスに関するさまざまな側面を構成レベルでチューニングできます。管理コンソールの「パフォーマンス」タブには、次のような多数のパフォーマンスカテゴリの設定が含まれています。

- HTTP 設定 (スレッドプールやキープアライブを含む)
- DNS 設定
- SSL および TLS の設定
- キャッシュ設定
- CGI 設定
- アクセスログバッファー設定

また、適切な wadm コマンドを使用することによっても、チューニングパラメータの表示や設定を行えます。一般に、wadm コマンドを使ってチューニングプロパティーを設定する場合、そのプロパティーの名前は stats.xml に表示されるものと同じになります。

接続キュー情報

Web Server では、接続はまず、HTTP リスナーに関連付けられたアクセプタスレッドによって受け付けられます。アクセプタスレッドが接続を受け付け、それを接続キューに入れます。次に、要求処理スレッドが接続キュー内の接続を取り出し、その要求を処理します。詳細については、44 ページの「接続処理の概要」を参照してください。

接続キュー情報には、接続キュー内のセッション数と、接続が要求処理スレッドによって受け付けられるまでの平均遅延が表示されます。

次に、perfdump でのそれらの統計の表示例を示します。

```
ConnectionQueue:
----------------------------------------
Current/Peak/Limit Queue Length            0/1853/160032
Total Connections Queued                   11222922
Average Queue Length (1, 5, 15 minutes)    90.35, 89.64, 54.02
Average Queueing Delay                     4.80 milliseconds
```

管理コンソールまたはコマンド行インタフェースではこれと同じ情報が異なる形式で表示されます。ただし、若干の相違はあります。次の表に、管理コンソールでサーバーインスタンスの監視情報にアクセスした際に表示される情報を示します。

表 2–2 接続キューの統計

現在の待機接続の数	0
待機接続の合計数	11222922
最近 1 分の平均接続数	90.35
最近 5 分の平均接続数	89.64
最近 15 分の平均接続数	54.02
最大キューサイズ	160032
最大キューサイズ	1853
オーバーフロー接続の数	0
消費したティック	5389284274
追加した接続の合計数	425723

現在/ピーク/制限のキュー長

Current/Peak/Limit queue length には、次の情報が順に表示されます。

- キュー内に現在存在している接続の数。
- キュー内に同時に存在していた接続の最大数。
- 接続キューの最大サイズ。この数値は次のようになります。

  最大キューサイズ = スレッドプールキューサイズ + 最大スレッド数 + キープアライブキューサイズ

  接続キューがいっぱいになると、新しい接続が無視されます。

チューニング

ピークキュー長 (最大キューサイズ) が制限に近くなった場合には、最大接続キューサイズを増やすことで、高負荷時に接続が無視されるのを回避できます。

管理コンソールで最大接続キューサイズを増やすには、構成の「パフォーマンス」タブ⇒「HTTP」サブタブにある、スレッドプールの「キューサイズ」フィールドの値を変更します。デフォルトは 1024 です。

コマンド行インタフェースを使ってキューサイズを変更するには、`wadm set-thread-pool-prop` コマンドの `queue-size` プロパティーを使用します。

待機接続の合計数

`Total Connections Queued` は、これまで接続がキューに入れられた回数の合計値です。この数値には、新しく受け付けられた接続と、キープアライブシステムからの接続が含まれます。

この設定はチューニング不可能です。

平均キュー長

`Average Queue Length` には、直前の 1 分間、5 分間、および 15 分間におけるキュー内の平均接続数が表示されます。

この設定はチューニング不可能です。

平均キュー遅延

`Average Queueing Delay` は、接続が接続キュー内で費やす時間の平均値です。これは、ある要求の接続がサーバーによって受け付けられてから、要求処理スレッドがその要求の処理を開始するまでの遅延を表します。これは、「消費したティック」を「待機接続の合計数」で割った結果をミリ秒に変換したものになります。

この設定はチューニング不可能です。

消費したティック

ティックはシステムに依存する値であり、`stats.xml` 内の `server` 要素の `tickPerSecond` 属性によって提供されます。消費したティックの値は、接続が接続キュー内で費やした時間の合計であり、平均キュー遅延を計算するために使用されます。

この設定はチューニング不可能です。

追加した接続の合計数

接続キューに追加された新しい接続の数。この設定はチューニング不可能です。

HTTP リスナー (待機ソケット) 情報

次の HTTP リスナー情報には、IP アドレス、ポート番号、アクセプタスレッドの数、およびデフォルト仮想サーバーが含まれています。この HTTP リスナー情報の中で、チューニングのためにもっとも重要なフィールドは、アクセプタスレッドの数です。

仮想サーバーに対して多数の HTTP リスナーを有効にすることができますが、デフォルトサーバーインスタンスでは少なくとも 1 つのリスナー (通常は `http://0.0.0.0:80`) が有効になります。管理コンソール経由で利用可能な監視情報には、HTTP リスナー情報が表示されません。なぜなら、管理コンソールでは、その情報は構成の「HTTP リスナー」タブで利用可能になっているからです。

次に、`perfdump` での HTTP リスナー情報の表示例を示します。

```
ListenSocket ls1:
-----------------------
Address                   https://0.0.0.0:2014
Acceptor Threads          1
Default Virtual Server    https-test
```

HTTP リスナーを複数個作成した場合、`perfdump` にはそれらがすべて表示されます。

管理コンソールを使って HTTP リスナーを編集するには、使用する構成で「HTTP リスナー」タブを選択します。リスナーの名前をクリックしてそのリスナーを編集します。

コマンド行インタフェースを使って HTTP リスナーを構成するには、コマンド `wadm set-http-listener-prop` を使用します。

待機ソケットを追加および編集する方法の詳細については、『Sun Java System Web Server 7.0 管理ガイド』を参照してください。

アドレス

`Address` フィールドには、この待機ソケットが待機する基底アドレスが含まれます。1 つのホストが複数のネットワークインタフェースと複数の IP アドレスを持つ可能性があります。アドレスには IP アドレスとポート番号が含まれます。

ホストマシンのすべてのネットワークインタフェース上で待機ソケットが待機する場合、アドレスの IP 部分は 0.0.0.0 になります。

チューニング

この設定は、HTTP リスナーの編集時にチューニング可能です。0.0.0.0 以外の IP アドレスを指定すると、サーバーが接続ごとに行うシステムコールの数が、1 つだけ少なくなります。実現可能な最高のパフォーマンスが得られるよう、0.0.0.0 以外の IP アドレスを指定してください。

アクセプタスレッド

アクセプタスレッドは、接続を待機するスレッドです。それらのスレッドが接続を受け付けてキューに入れたあと、ワークスレッドがそれらの接続を取り出します。詳細については、44 ページの「接続処理の概要」を参照してください。

ユーザーからの要求があったときにいつでも対応できるように、常に十分な数のアクセプタスレッドを用意しておくのが理想的ですが、システムに負荷がかかり過ぎない数に抑える必要もあります。推奨の規則は、システム上の CPU ごとにアクセプタスレッドを 1 つずつ用意することです。TCP/IP 待機キューのオーバーランが発生する場合には、この値を CPU 数の約 2 倍に増やすことができます。

チューニング

この設定は、HTTP リスナーの編集時にチューニング可能です。デフォルトは 1 です。

パフォーマンスに影響を与えるその他の HTTP リスナー設定は、送信バッファーと受信バッファーのサイズです。これらのバッファーの詳細については、使用しているオペレーティングシステムのマニュアルを参照してください。

デフォルト仮想サーバー

仮想サーバーは、HTTP 1.1 の Host ヘッダーを使って動作します。エンドユーザーのブラウザが Host ヘッダーを送信しなかった場合や、サーバーが Host ヘッダーに指定された仮想サーバーを見つけられなかった場合、Web Server はデフォルト仮想サーバーを使って要求を処理します。エラーメッセージを送信したり、特殊なドキュメントルートからのページを提供したりするように、デフォルト仮想サーバーを構成することができます。

チューニング

この設定は、HTTP リスナーの編集時にチューニング可能です。

キープアライブ情報

このセクションでは、サーバーの HTTP レベルキープアライブシステムに関する情報が提供されます。

注 – 「キープアライブ (keep alive)」という名前を TCP の `keep-alives` と混同しないようにしてください。また、HTTP 1.1 では「キープアライブ」という名前が `PersistentConnections` に変更されましたが、Web Server では引き続き、それらの接続をキープアライブ接続と呼んでいます。

次の例は、`perfdump` で表示されるキープアライブ統計を示したものです。

```
KeepAliveInfo:
--------------------
KeepAliveCount        198/200
KeepAliveHits         0
KeepAliveFlushes      0
KeepAliveRefusals     56844280
KeepAliveTimeouts     365589
KeepAliveTimeout      10 seconds
```

次の表に、管理コンソールで表示されるキープアライブ統計を示します。

表 2–3 キープアライブ統計情報

処理した接続の数	0
追加した接続の合計数	198
最大接続サイズ	200
フラッシュした接続の数	0
拒否された接続の数	56844280
終了したアイドル接続の数	365589
接続タイムアウト	10

HTTP 1.0 と HTTP 1.1 はどちらも、単一の HTTP セッションで複数の要求を送信する機能をサポートしています。Web サーバーは、新しい HTTP 要求を 1 秒間に数百件受信できます。すべての要求の接続をいつまでも開いたままにできると、サーバーは多くの接続で過負荷状態になってしまう可能性があります。UNIX および Linux システム上では、これにより、ほんのわずかな負荷でファイルテーブルのオーバーフローが発生する可能性があります。

この問題に対処するため、サーバーは、待機キープアライブ接続の最大数のカウンタを維持します。待機キープアライブ接続は、以前の要求の処理を完了し、その同じ接続上で新しい要求が到着するのを待っています。サーバー上で最大待機接続数を超える接続が開いた状態で、新しい接続がキープアライブ要求を待機している場合、サーバーはもっとも古い接続を閉じます。このアルゴリズムにより、サーバーが維持できる開いた待機キープアライブ接続の上限が保たれます。

Sun Java System Web Server は必ずしも、クライアントからのキープアライブ要求に従うとは限りません。次の条件が成立する場合には、クライアントがキープアライブ接続を要求しても、サーバーは接続を閉じます。

- キープアライブのタイムアウトが 0 に設定されている。
- キープアライブの最大接続数を超えている。
- CGI などの動的コンテンツで HTTP `content-length` ヘッダーが設定されていない。このことは HTTP 1.0 要求だけに当てはまります。要求が HTTP 1.1 の場合には、`content-length` が設定されていなくても、サーバーはキープアライブ要求に従います。クライアントがチャンクエンコーディング (要求ヘッダー `transfer-encoding: chunked` で示される) を処理できる場合、サーバーはそれらの要求に対してこのエンコーディングを使用できます。
- 要求が `HTTP GET`、`HEAD` のいずれでもない。
- 要求が不正であると判定された。たとえば、クライアントがコンテンツを送信せず、ヘッダーのみを送信する場合などです。

Web Server のキープアライブサブシステムは、きわめて高いスケーラビリティーを実現できるように設計されています。デフォルトの構成は、作業負荷が持続的でない場合 (つまり、`KeepAlive` ヘッダーを持たない HTTP 1.0 の場合) や、主にキープアライブ接続へのサービスを提供する低負荷システムで使用する場合には、最適でない可能性があります。

キープアライブカウント

`perfdump` のこのセクションには、次の 2 つの数値が含まれます。

- キープアライブモードの接続の数 (追加された接続の合計数)
- キープアライブモードで許容される最大同時接続数 (最大接続サイズ)

チューニング

サーバーがもっとも古い接続を閉じる前に待機を同時に許可する接続の最大数を管理コンソールでチューニングするには、構成の「パフォーマンス」タブ⇒「HTTP」タブの、「キープアライブ設定」の下にある「最大接続数」フィールドを編集します。デフォルトは 200 です。コマンド行インタフェースでは、`wadm set-keep-alive-prop` コマンドで `max-connections` プロパティーを使用します。

注 – 最大接続数の設定に指定された接続数は、キープアライブスレッド間で均等に分割されます。最大接続数の設定がキープアライブスレッド数の設定で均等に分割できない場合、サーバーは、同時キープアライブ接続の最大数よりも若干多い値を許可する可能性があります。

キープアライブヒット数

キープアライブヒット数 (処理された接続の数) は、キープアライブ接続から要求が正常に受信された回数です。

この設定はチューニング不可能です。

キープアライブフラッシュ数

追加された接続の合計数がキープアライブ最大接続数の設定を超えたために、サーバーが接続を閉じなければいけなかった回数。このサーバーは、キープアライブ数が最大接続サイズを超えても既存の接続を閉じません。代わりに、新しいキープアライブ接続が拒否され、拒否された接続の数が増分されます。

キープアライブ拒否数

おそらく持続的接続が多すぎたために (あるいは追加された接続の合計数がキープアライブ最大接続数の設定を超えたときに)、サーバーが接続をキープアライブスレッドに渡せなかった回数。推奨のチューニングは、キープアライブ最大接続数を増やすことです。

キープアライブタイムアウト数

クライアント接続が何の活動も見られないままタイムアウトに達し、サーバーがそのアイドル状態のキープアライブ接続を閉じた回数。この統計は監視対象として有用ですが、推奨のチューニング方法は特にありません。

キープアライブタイムアウト

アイドル状態のキープアライブ接続が閉じられるまでの時間 (秒)。この値を管理コンソールで設定するには、構成の「パフォーマンス」タブ ⇒「HTTP」タブの、「キープアライブ設定」の下にある「タイムアウト」フィールドを使用します。デフォルトは 30 秒ですが、これは、アイドル状態が 30 秒を超えると接続がタイムアウトすることを意味します。最大値は 3600 秒 (60 分) です。コマンド行インタフェースでは、`wadm set-keep-alive-prop` コマンドで `timeout` プロパティーを使用します。

キープアライブポーリング間隔

キープアライブポーリング間隔は、システムがキープアライブ接続に対するポーリングを行なって要求の有無を確認する間隔 (秒) を指定します。デフォルトは、許可される最小値である 0.001 秒です。これが小さな値に設定されているのは、CPU の使用率を犠牲にしてパフォーマンスを向上させるためです。

ポーリング間隔をチューニングするには、構成の「パフォーマンス」タブ ⇒「HTTP」タブの、「キープアライブ設定」の下にある「ポーリング間隔」フィールドを編集します。コマンド行インタフェースでは、`wadm set-keep-alive-prop` コマンドで `poll-interval` プロパティーを使用します。

キープアライブスレッド数

キープアライブシステムで使用されるスレッドの数を管理コンソールで構成するには、構成の「パフォーマンス」タブ⇒「HTTP」タブの、「キープアライブ設定」の下にある「スレッド数」フィールドを編集します。デフォルトは1です。コマンド行インタフェースでは、`wadm set-keep-alive-prop` コマンドで `threads` プロパティーを使用します。

HTTP 1.0 スタイルの作業負荷に対するチューニング

HTTP 1.0 では新しい受信接続が多数発生するため、1 待機ソケットあたりのアクセプタスレッド数のデフォルト値 1 は、最適であるとは言えません。HTTP 1.0 スタイルの作業負荷では、これを大きな数値に増やすとパフォーマンスが改善されるはずです。たとえば、2 個の CPU を備えたシステムの場合、これを 2 に設定することをお勧めします。また、キープアライブ接続数を 0 などに減らすこともお勧めします。

HTTP 1.0 スタイルの作業負荷では、多数の接続が確立され、終了されます。

サーバーが高負荷状態のときにブラウザから Web Server への接続でタイムアウトが発生する場合には、HTTP リスナーの待機キューサイズを 8192 などの大きな値に設定することで、HTTP リスナーのバックログキューのサイズを増やすことができます。

HTTP リスナーの待機キューは、ある待機ソケット上で保留される接続の最大数を指定します。バックログキューがいっぱいになった待機ソケット上で接続がタイムアウトすると、その接続は失敗します。

HTTP 1.1 スタイルの作業負荷に対するチューニング

一般に、サーバーによる持続的接続の処理をチューニングする場合、スループットと待ち時間はトレードオフの関係になります。キープアライブポーリング間隔とタイムアウトによって待ち時間が決まります。これらの設定の値を小さくすることの狙いは、ページの読み込み時間を短縮するなど、低負荷システムでの待ち時間を短縮することです。これらの設定の値を増やすことの狙いは、サーバーが 1 秒間に処理できる要求の数を増やすなど、高負荷システムでの全体的なスループットを高めることです。ただし、待ち時間が長すぎるのにクライアントが少なすぎる場合には、サーバーが不必要にアイドル状態になるため、全体的なスループットが低下します。結論として、ある特定の負荷状態におけるキープアライブサブシステムの一般的なチューニング規則は、次のようになります。

- アイドル状態の CPU 時間が存在する場合は、ポーリング間隔を減らします。
- アイドル状態の CPU 時間が存在しない場合は、ポーリング間隔を増やします。

また、チャンクエンコーディングは、HTTP 1.1 の作業負荷でのパフォーマンスに影響を与える可能性があります。応答バッファーサイズのチューニングは、パフォーマンスに良い影響を与える可能性があります。構成の「パフォーマンス」タブ⇒「HTTP」タブで応答バッファーサイズを大きくすると、応答がチャンクに分割され

る代わりに、Content-length: ヘッダーが送信されるようになります。CLI を使ってバッファーサイズを設定するには、wadm set-http-prop コマンドの output-buffer-size プロパティーを使用します。

また、obj.conf ファイル内で Service クラスの関数のバッファーサイズを UseOutputStreamSize パラメータを使って設定することもできます。UseOutputStreamSize は、output-buffer-size プロパティーを使って設定された値よりも優先されます。UseOutputStreamSize が設定されていない場合、Web Server は output-buffer-size の設定を使用します。output-buffer-size が設定されていない場合、Web Server は output-buffer-size のデフォルト値 8192 を使用します。

次の例は、CLI を使って出力バッファーサイズを増やしたあと、構成を配備する方法を示したものです (obj.conf で UseOutputStreamSize が指定されていない場合に使用される)。

```
./wadm set-http-prop --user=admin-user --password-file=admin-password-file
--config=config-name output-buffer-size=16384
./wadm deploy-config --user=admin-user --password-file=admin-password-file
 --config=config-name
```

次の例は、nsapi_test Service 関数のバッファーサイズを設定する方法を示したものです。

```
<Object name="nsapitest">
 ObjectType fn="force-type" type="magnus-internal/nsapitest"
 Service method=(GET) type="magnus-internal/nsapitest" fn="nsapi_test"
 UseOutputStreamSize=12288
</Object>
```

セッション作成 (スレッド) 情報

セッション (スレッド) 作成の統計は、perfdump では次のように表示されます。

```
SessionCreationInfo:
------------------------
Active Sessions           128
Keep-Alive Sessions       0
Total Sessions Created    128/128
```

Active Sessions は、現在要求を処理しているセッションの数 (要求処理のスレッド) を示しています。

Keep-Alive Sessions には、キープアライブセッションを処理する HTTP 要求処理スレッドの数が表示されます。

perfdump の Total Sessions Created には、作成されたセッションの数と最大スレッド数が表示されます。

管理コンソールでこれに相当する情報である「スレッドの合計数」は、「監視」タブ⇒「インスタンス」サブタブの「一般統計」の下で利用可能になっています。許容される最大のスレッド数を表示するには、構成の「パフォーマンス」タブ⇒「HTTP」サブタブの、「スレッドプール設定」の下にある「最大スレッド数」フィールドを参照してください。

`perfdump` の `Active Sessions` に相当する情報を得るには、「スレッドの合計数」から「アイドルスレッドの数」を引きます。

最大スレッド数 (最大同時要求数)

最大スレッド数の設定は、Web Server が処理できる同時トランザクションの最大数を指定します。デフォルト値は 128 です。この値を変更するとサーバーのパフォーマンスを調整でき、実行されるトランザクションの待ち時間を最小限に抑えることができます。「最大スレッド数」の値は複数の仮想サーバーにまたがって適用されますが、その際に負荷分散が試みられることはありません。これは構成ごとに設定されます。

設定されたスレッドの最大数に達してしまってもかまいません。また、サーバーのスレッド数を反射的に増やす必要もありません。この最大限度に達したということは、サーバーがピークロード時にこれだけの数のスレッドを必要としたことを意味しています。しかし、要求がタイムリーに処理されているかぎり、サーバーは適切にチューニングされているといえます。ただし、この時点で接続は接続キューに入れられるため、オーバーフローする可能性もあります。サーバーのパフォーマンスを定期的に監視し、作成されたセッションの合計数が最大スレッド数に近くなることが多いようであれば、スレッドの上限を増やすことを検討するべきです。

同時処理する要求数を計算するために、サーバーはアクティブな要求数をカウントし、そこに新しい要求が届いたら 1 を足し、要求が終了したら 1 を引きます。新しい要求が到着すると、サーバーは処理中の要求がすでに最大数に達しているかどうかをチェックします。上限に達していた場合、サーバーは、アクティブな要求の数がその最大量を下回るまで、新しい要求の処理を遅らせます。

理論上は、最大スレッド数を 1 に設定しても、サーバーは正常に機能します。この値を 1 に設定することは、サーバーが一度に処理できる要求が 1 つだけであることを意味しますが、静的ファイルの HTTP 要求の処理時間は一般に非常に短い (応答時間はわずか 5 ミリ秒程度である可能性がある) ため、一度に 1 つの要求を処理したとしても、1 秒間に最大 200 件の要求を処理できることになります。

ただし、実際には、インターネットクライアントがサーバーに接続したあと、要求を完了しないことが頻繁に発生します。そうした場合には、サーバーはデータの到着を 30 秒またはそれ以上待ったあと、タイムアウトを発生させます。このタイムアウト時間を定義するには、構成の「パフォーマンス」タブ⇒「HTTP 設定」ページの「入出力タイムアウト」設定を使用します。また、コマンド `wadm set-http-prop` を使って `io-timeout` プロパティーを設定することもできます。デフォルトは 30 秒です。これをデフォルトより小さい値に設定すれば、スレッドがより早く解放される

ようになりますが、一方で低速な接続のユーザーを切断する可能性も出てきます。また、一部のサイトでは、完了までに数分かかる高負荷のトランザクションが実行されます。これらの要因はどちらも、必要とされる最大同時要求数を増加させます。多くの秒数のかかる要求を多数処理するサイトでは、最大同時要求数を増やさなければいけない可能性があります。

適切な最大スレッド値の範囲は、負荷によって 100 ～ 500 になります。「最大スレッド数」は同時に実行可能なアクティブスレッドの最大数に対する強い制限値を表しますが、これがパフォーマンス上の障害になる可能性があります。デフォルト値は 128 です。

スレッドプールの最小スレッド数は、サーバーが起動時に開始するスレッドの最小数です。デフォルト値は 16 です。

注 – SNCA (Solaris Network Cache and Accelerator) と組み合わせて使用するように Web Server を構成する場合、最大スレッド数とキューサイズを 0 に設定するとパフォーマンスが改善されます。SNCA がクライアント接続を管理するので、これらのパラメータを設定する必要はありません。SNCA を使用しない構成でも、これらのパラメータを 0 に設定できます。キープアライブを使用しないで待ち時間の短い応答を配信する必要がある場合は特にそうです。最大スレッド数とキューサイズの「両方」を 0 に設定する必要があることに注意してください。

SNCA の使用方法については、100 ページの「Solaris Network Cache and Accelerator (SNCA) の使用」を参照してください。

チューニング

管理コンソールでスレッドの上限を増やすには、構成の「パフォーマンス」タブ ⇒「HTTP」タブの、「スレッドプール設定」の下にある「最大スレッド数」フィールドを編集します。コマンド行インタフェースでは、`wadm set-thread-pool-prop` コマンドの `max-threads` プロパティーを使用します。デフォルトは 128 です。

ファイルキャッシュ情報 (静的コンテンツ)

キャッシュ情報セクションは、ファイルキャッシュがどのように使用されているかに関する統計を提供します。ファイルキャッシュには静的コンテンツが書き込まれるため、サーバーは静的コンテンツに対する要求をすばやく処理できます。ファイルキャッシュには、ファイルに関する情報と静的なファイルコンテンツが保存されます。ファイルキャッシュでは、サーバーにより解析される HTML の処理の高速化に使用される情報もキャッシュに書き込まれます。サーブレットや JSP では、別の種類のキャッシュが使用されます。

コンテンツへの更新がスケジュールに従って行われるサイトでは、コンテンツの更新中はキャッシュを停止し、更新完了後にキャッシュを再度起動することを検討してください。キャッシュを無効にすると、パフォーマンスは低下しますが、サーバーは正常に動作します。

パフォーマンス上の理由により、Web Server はキャッシュへの書き込みを次のように行います。

- ファイルが小さい場合、そのコンテンツがメモリー (ヒープ) 内のキャッシュに書き込まれる。
- ファイルが大きい場合、オープンファイル記述子がキャッシュに書き込まれる (ファイルのオープン/クローズを回避するため)。

次に、`perfdump` でのキャッシュ統計の表示例を示します。

```
CacheInfo:
------------------
enabled             yes
CacheEntries        12/1024
Hit Ratio           46/98 ( 46.94%)
Maximum Age         30
```

次の表に、管理コンソールで表示されるファイルキャッシュ統計を示します。

表 2–4 ファイルキャッシュ統計

キャッシュヒットの合計	46
キャッシュミスの合計	52
キャッシュコンテンツヒットの合計	0
ファイル検索エラーの数	9
ファイル情報検索の数	37
ファイル情報検索エラーの数	50
エントリ数	12
最大キャッシュサイズ	1024
オープンファイルエントリの数	0
許可されている最大オープンファイル数	1024
ヒープサイズ (Heap Size)	36064
最大ヒープキャッシュサイズ	10735636
メモリーマップファイルコンテンツのサイズ	0

表 2–4 ファイルキャッシュ統計 (続き)

最大メモリーマップファイルサイズ	0
エントリの最大継続時間	30

有効

キャッシュが無効になっていると、このセクションの残りの部分が perdump で表示されません。管理コンソールでは、「ファイルキャッシュ統計」セクションの値としてゼロが表示されます。

チューニング

キャッシュはデフォルトで有効になっています。管理コンソールでこれを無効にするには、構成の「パフォーマンス」タブ⇒「キャッシュ」サブタブの「ファイルキャッシュ」の下にある、「ファイルキャッシュ」の「有効」ボックスの選択を解除します。コマンド行インタフェースでこれを無効にするには、wadm set-file-cache-prop を使って enabled プロパティーを false に設定します。

キャッシュエントリ数

キャッシュエントリの現在の数と最大数の両方が、perfdump に表示されます。管理コンソールでは、それらは「エントリ数」および「最大キャッシュサイズ」と呼ばれます。単一のキャッシュエントリは単一の URI を表します。

チューニング

管理コンソールでキャッシュエントリの最大数を設定するには、構成の「パフォーマンス」タブ⇒「キャッシュ」タブの、「ファイルキャッシュ」の下にある「最大エントリ数」フィールドを使用します。コマンド行インタフェースでは、wadm set-file-cache-prop を使って max-entries プロパティーを設定します。デフォルトは 1024 です。値の範囲は 1 〜 1048576 です。

ヒット率 (キャッシュヒット数 / キャッシュ検索数)

perfdump 経由で利用可能なヒット率は、キャッシュ検索数に対するファイルキャッシュヒット数です。100% に近い数値が、ファイルキャッシュが効果的に機能していることを示しているのに対し、0% に近い数値は、ファイルキャッシュがあまり多くの要求を処理できていないことを示している可能性があります。

管理コンソール経由で提供される統計に基づいてこの数値を手動で計算するには、「キャッシュヒットの合計」と「キャッシュミスの合計」を足したもので、「キャッシュヒットの合計」を割ります。

この設定はチューニング不可能です。

最大継続時間

このフィールドには、有効なキャッシュエントリの最大継続時間が表示されます。このパラメータは、ファイルがキャッシュに書き込まれたあと、そのキャッシュ情報がどのくらいの期間使用されるかを制御します。最大継続時間を経過したエントリは、同じファイルの新しいエントリで置き換えられます。

チューニング

コンテンツを定期的に更新 (既存のファイルを変更) するかどうかに基づいて最大時間を設定します。たとえば、コンテンツが定期的に 1 日に 4 回更新される場合、最大時間を 21600 秒 (6 時間) に設定できます。それ以外の場合、最大時間には、ファイルが変更されたあと、以前のバージョンのコンテンツファイルを提供する最大時間を設定することを検討してください。コンテンツの変更頻度が低い Web サイトでは、パフォーマンスを改善するためにこの値を増やすことをお勧めします。

管理コンソールで最大継続時間を設定するには、構成の「パフォーマンス」タブ⇒「キャッシュ」タブの、「ファイルキャッシュ」の下にある「最大継続時間」フィールドを使用します。コマンド行インタフェースでは、`wadm set-file-cache-prop` を使って `max-age` プロパティーを変更します。デフォルトは 30 秒です。値の範囲は 0.001 ～ 3600 です。

最大ヒープキャッシュサイズ

最適なキャッシュヒープサイズは、システムメモリーの空き容量がどのくらい存在するかによって異なります。ヒープサイズを大きくすれば、Web Server がより多くのコンテンツをキャッシュに書き込めるため、ヒット率も向上することになります。ただし、オペレーティングシステムがキャッシュに書き込まれたファイルのページングを開始するほどヒープサイズを大きくすべきではありません。

チューニング

管理コンソールで最大ヒープサイズを設定するには、構成の「パフォーマンス」タブ⇒「キャッシュ」タブの、「ファイルキャッシュ」の下にある「最大ヒープスペースサイズ」フィールドを使用します。コマンド行インタフェースでは、`wadm set-file-cache-prop` を使って `max-heap-space` プロパティーを変更します。デフォルト値は 10485760 バイトです。値の範囲は 0 ～ 9223372036854775807 です。32 ビット Web Server では、プロセスはファイルキャッシュ用に 4G バイトのアドレス空間を使用するため、この値は 4G バイト未満にするべきです。

nocache パラメータの使用

`Service` 関数`send-file` でパラメータ `nocache` を使えば、ある特定のディレクトリ内のファイルをキャッシュに書き込まないように指定できます。この変更を行うには `obj.conf` を編集します。たとえば、変更頻度が高すぎるためにキャッシュに書き込んでもあまり役に立たないような一連のファイルが存在する場合、それらのファイ

ルをあるディレクトリに格納し、obj.conf を編集してそのディレクトリ内のファイルをキャッシュに書き込まないようにサーバーに指示することができます。

例

```
<Object name=default>
...
NameTrans fn="pfx2dir" from="/myurl" dir="/export/mydir"
name="myname"
...
Service method=(GET|HEAD|POST) type=*~magnus-internal/*
fn=send-file
...
</Object>
<Object name="myname">
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file
nocache=""
</Object>
```

この例では、URL プレフィックス /myurl によって /export/mydir/ 内の静的ファイルが要求された際に、サーバーはそれらのファイルをキャッシュに書き込みません。obj.conf の編集方法の詳細については、『Sun Java System Web Server 7.0 Administrator's Configuration File Reference』を参照してください。

ファイルキャッシュの動的な制御と監視

obj.conf にオブジェクトを追加すれば、サーバーの実行中にファイルキャッシュを動的に監視および制御できます。

▼ ファイルキャッシュを制御および監視する

1 NameTrans 指令をデフォルトオブジェクトに追加します。

```
NameTrans fn="assign-name" from="/nsfc" name="nsfc"
```

2 nsfc オブジェクトの定義を追加します。

```
<Object name="nsfc">
Service fn="service-nsfc-dump"
</Object>
```

これにより、ファイルキャッシュを制御および監視するための関数 (nsfc-dump) に、URI /nfsc 経由でアクセスできるようになります。別の URI を使用するには、NameTrans 指令の from パラメータを変更します。

次に、この URI へのアクセス時に表示される情報の例を示します。

```
Sun Java System File Cache Status (pid 3602)

The file cache is enabled.
Cache resource utilization

Number of cached file entries = 174968 (152 bytes each, 26595136 total bytes)
Heap space used for cache = 1882632616/1882632760 bytes
Mapped memory used for medium file contents = 0/1 bytes
Number of cache lookup hits = 47615653/48089040 ( 99.02 %)
Number of hits/misses on cached file info = 23720344/324195
Number of hits/misses on cached file content = 16247503/174985
Number of outdated cache entries deleted = 0
Number of cache entry replacements = 0
Total number of cache entries deleted = 0

Parameter settings

ReplaceFiles: false
ReplaceInterval: 1 milliseconds
HitOrder: false
CacheFileContent: true
TransmitFile: false
MaxAge: 3600 seconds
MaxFiles: 600000 files
SmallFileSizeLimit: 500000 bytes
MediumFileSizeLimit: 1000001 bytes
BufferSize: 8192 bytes

CopyFiles: false
Directory for temporary files: /tmp
Hash table size: 1200007 buckets
```

この URI へのアクセス時には、クエリー文字列を含めることができます。次の値が認識されます。

- `?list`: キャッシュ内のファイルを一覧表示する。
- `?refresh=`*n*: *n* 秒ごとにクライアントがページの再読み込みを行うようにする。
- `?restart`: キャッシュが停止後、再起動されるようにする。
- `?start`: キャッシュを起動する。
- `?stop`: キャッシュをシャットダウンする。

`?list` オプションを選択する場合、そのファイルリストには、ファイル名、一連のフラグ、現在のキャッシュエントリ参照数、ファイルのサイズ、および内部ファイル ID 値が含まれます。フラグは次のとおりです。

- `C`: ファイルのコンテンツがキャッシュに書き込まれている。
- `D`: キャッシュエントリが削除対象としてマークされている。

- E: PR_GetFileInfo() がこのファイルでエラーを返した。
- I: ファイルの情報 (サイズや変更日付など) がキャッシュに書き込まれている。
- M: ファイルのコンテンツが仮想メモリー内にマップされている。
- O: ファイル記述子がキャッシュに書き込まれている (TransmitFile が true に設定されている場合)。
- P: ファイルに非公開データが関連付けられている (shtml ファイルの場合に表示されるはず)。
- T: キャッシュエントリが一時ファイルを持っている。
- W: キャッシュエントリが書き込みアクセス用としてロックされている。

スレッドプール情報

ユーザーがデフォルト設定を使用している場合、デフォルトスレッドプール内のスレッドが要求を処理します。ただし、カスタムスレッドプールを作成し、それらを使ってカスタム NSAPI 関数を実行することもできます。Web Server はデフォルトで、NativePool という名前の追加プールを 1 つ作成します。ほとんどの場合、ネイティブスレッドプールが必要になるのは、Windows プラットフォームの場合だけです。スレッドプールの詳細については、44 ページの「スレッド、プロセス、および接続の理解」を参照してください。

ネイティブスレッドプール

次の例は、perfdump で表示されるネイティブスレッドプール情報を示したものです。

```
Native pools:
----------------------------
NativePool:
Idle/Peak/Limit               1/1/128
Work Queue Length/Peak/Limit  0/0/0
my-custom-pool:
Idle/Peak/Limit               1/1/128
Work Queue Length/Peak/Limit  0/0/0
```

ユーザーが追加のカスタムスレッドプールを定義した場合、それらのプールは perfdumpの「Native Pools」見出しの下に表示されます。

次の表に、管理コンソールで表示されるスレッドプール統計を示します。ユーザーが追加のスレッドプールを定義しなかった場合は、NativePool のみが表示されます。

表2–5 スレッドプールの統計

名前	NativePool
アイドル状態のスレッド	1
スレッド数	1
待機した要求	0
待機した最大要求数	0

アイドル/ピーク/制限

`Idle` は管理コンソールでは「アイドル状態のスレッド」として表示されますが、これは、現在アイドル状態になっているスレッドの数を示します。`Peak` は、プール内のピークスレッド数を示します。`Limit` は管理コンソールでは「スレッド数」として表示されますが、これは、スレッドプール内で許可されるネイティブスレッドの最大数を示しており、NativePool の場合は、`magnus.conf` ファイル内の `NativePoolMaxThreads` の設定によって決まります。

チューニング

NativePool の最大スレッド数を変更するには、`magnus.conf` 内の `NativePoolMaxThreads` パラメータを編集します。詳細については、74 ページの「NativePoolMaxThreads 指令」を参照してください。

ワークキューの長さ/ピーク/制限

これらの数値は、プール内のネイティブスレッドを使用するために待機しているサーバー要求のキューに関するものです。`Work Queue Length` はネイティブスレッドを待機している要求の現在の数であり、管理コンソールでは「待機した要求」として表示されます。

`Peak` (管理コンソールでは「待機した最大要求数」) は、ネイティブスレッドを使用するためにキュー内に同時に存在していた要求の、サーバーが起動されてからその時点までの最大数です。この値は、ネイティブスレッドを必要とする要求の最大並行性とみなすことができます。

`Limit` はネイティブスレッドを待機するためにキュー内に一度に格納できる要求の最大数であり、`NativePoolQueueSize` の設定によって決まります。

チューニング

NativePool のキューサイズを変更するには、`magnus.conf` 内の `NativePoolQueueSize` 指令を編集します。詳細については、73 ページの「NativePoolQueueSize 指令」を参照してください。

NativePoolStackSize 指令

NativePoolStackSize は、ネイティブ (カーネル) スレッドプール内の各スレッドのスタックサイズをバイト単位で決定します。

チューニング

NativePoolStackSize を変更するには、magnus.conf 内の NativePoolStackSize 指令を編集します。

NativePoolQueueSize 指令

NativePoolQueueSize は、スレッドプール用のキュー内で待機できるスレッドの数を決定します。プール内のすべてのスレッドがビジー状態になると、ネイティブプール内のスレッドを使用する必要のある次の要求処理スレッドは、キュー内で待機する必要があります。キューがいっぱいになると、次の要求処理スレッドがキューに入ろうとしても拒否され、最終的にそのスレッドはクライアントにビジー応答を返します。その後、そのスレッドは、キュー内で待機しながら拘束される代わりに、別の受信要求を自由に処理できるようになります。

NativePoolQueueSize を最大スレッド数の値よりも小さい値に設定すると、プールスレッドによるサービスを待機している要求の数がこの値を超えるたびに、サーバーは目的とする NSAPI 関数の代わりにビジー関数を実行します。デフォルトでは、「503 Service Unavailable」応答が返され、ログレベルの設定に応じたメッセージがログに記録されます。NativePoolQueueSize を最大スレッド数よりも大きい値に設定すると、ビジー関数が実行可能になる前にサーバーが接続を拒否します。

この値は、ネイティブスレッドを必要とするサービスを同時に求める要求の最大数を表します。負荷が高いためにシステムが要求を処理できない場合に、待機可能な要求の数を増やすと、要求の待ち時間が長くなり、利用可能なすべての要求スレッドが 1 つのネイティブスレッドを待機することになる可能性があります。一般に、この値は、ネイティブスレッドを必要とする要求を実行する同時ユーザーの最大数を予想し、その要求が拒否されるのを回避できるだけの大きな値に設定します。

この値と最大スレッド数の違いは、静的 HTML や画像ファイルなどの、非ネイティブスレッド要求のために予約される要求数にあります。予約を確保して要求を拒否することで、サーバーが静的ファイルの要求を処理し続けることを保証でき、動的コンテンツの高負荷時にサーバーが応答不能になるのを防ぐことができます。サーバーが常に接続を拒否する場合には、この値の設定が小さすぎるか、あるいはサーバーのハードウェアが過負荷状態になっています。

チューニング

NativePoolQueueSize を変更するには、magnus.conf 内の NativePoolQueueSize 指令を編集します。

NativePoolMaxThreads 指令

NativePoolMaxThreads は、ネイティブ (カーネル) スレッドプール内のスレッドの最大数を決定します。

値を大きくするほど、同時に実行可能な要求が多くなりますが、コンテキストの切り替えによるオーバーヘッドも大きくなります。したがって、値が大きいほど良いとは限りません。通常の場合、この数値を増やす必要はありませんが、CPU がまだ飽和状態に達しておらず、かつ待機中の要求が多数存在している場合には、この数値を増やすべきです。

チューニング

NativePoolMaxThreads を変更するには、magnus.conf 内の NativePoolMaxThreads パラメータを編集します。

NativePoolMinThreads 指令

ネイティブ (カーネル) スレッドプール内のスレッドの最小数を決定します。

チューニング

NativePoolMinThreads を変更するには、magnus.conf 内の NativePoolMinThreads パラメータを編集します。

DNS キャッシュ情報

DNS キャッシュでは、IP アドレスと DNS 名がキャッシュされます。Web Server は、DNS キャッシュを使ってロギングや IP アドレスによるアクセス制御を行います。DNS キャッシュはデフォルトで有効になっています。次の例は、perfdump で表示される DNS キャッシュ情報を示したものです。

```
DNSCacheInfo:
------------------
enabled             yes
CacheEntries        4/1024
HitRatio            62854802/62862912 ( 99.99%)

AsyncDNS Data:
------------------
enabled             yes
NameLookups         0
AddrLookups         0
LookupsInProgress   0
```

次の例に、管理コンソールで表示される DNS キャッシュ情報を示します。

表 2–6 DNS キャッシュの統計

キャッシュヒットの合計	62854802
キャッシュミスの合計	6110
非同期検索の数	0
検索中	4
非同期検索が有効	1
実行した非同期アドレス検索の数	0

有効

DNS キャッシュが無効になっていると、このセクションの残りの部分が `perfdump` で表示されません。管理コンソールではページにゼロが表示されます。

チューニング

DNS キャッシュはデフォルトで有効になっています。管理コンソールで DNS キャッシュを有効または無効にするには、構成の「パフォーマンス」タブ ⇒「DNS」サブタブの、「DNS キャッシュ設定」の下で、「DNS キャッシュ」の「有効」ボックスを選択または選択解除します。コマンド行インタフェースでこれを有効または無効にするには、`wadm set-dns-cache-prop` を使って `enabled` プロパティーを設定します。

キャッシュエントリ数 (現在のキャッシュエントリ数/最大キャッシュエントリ数)

`perfdump` のこのセクションには、キャッシュエントリの現在の数と最大数が表示されます。管理コンソールでは、現在のキャッシュエントリ数は「キャッシュヒットの合計」として表示されます。単一のキャッシュエントリは、単一の IP アドレスまたは単一の DNS 名の検索を表します。キャッシュのサイズは、Web サイトに同時にアクセスするクライアントの最大数と同程度にするべきです。キャッシュサイズが大きすぎるとメモリーが浪費され、パフォーマンスも低下します。

チューニング

管理コンソールで DNS キャッシュの最大サイズを設定するには、構成の「パフォーマンス」タブ ⇒「DNS」サブタブの、「DNS キャッシュ設定」の下にある「最大キャッシュサイズ」フィールドを使用します。コマンド行インタフェースでこれを設定するには、`wadm set-dns-cache-prop` を使って `max-entries` プロパティーを設定します。デフォルトのキャッシュサイズは 1024 です。値の範囲は 2 〜 32768 です。

ヒット率 (キャッシュヒット数/キャッシュ検索数)

perfdump のヒット率には、キャッシュ検索数に対するキャッシュヒット数が表示されます。管理コンソールの統計に基づいてこの数値を計算するには、「キャッシュヒットの合計」と「キャッシュミスの合計」を足したもので、「キャッシュヒットの合計」を割ります。

この設定はチューニング不可能です。

非同期 DNS が有効/無効

Async DNS enabled/disabled には、サーバーがオペレーティングシステムの同期リゾルバの代わりに独自の非同期 DNS リゾルバを使用するかどうかが表示されます。非同期 DNS はデフォルトで無効になっています。これが無効になっていると、このセクションが perfdump で表示されません。管理コンソールでこれを有効にするには、構成の「パフォーマンス」タブ ⇒「DNS」タブの「DNS 検索設定」の下で、「非同期 DNS」を選択します。コマンド行インタフェースでこれを有効にするには、wadm set-dns-prop を使って async プロパティーを true に設定します。

Java 仮想マシン (JVM) 情報

JVM の統計が表示されるのは、管理コンソール、CLI、および stats-xml を使用する場合だけです。perfdump ではそれらは表示されません。

次の表に、管理コンソールで表示される JVM 統計の例を示します。

表 2–7　Java 仮想マシン (JVM) の統計

仮想マシン名	Java HotSpot™ Server VM
仮想マシンベンダー	Sun Microsystems Inc.
仮想マシンのバージョン	1.5.0_06-b05
ヒープメモリーサイズ	5884856
ガベージコレクション経過時間 (ミリ秒)	51
読み込んだクラスの現在の数	1795
読み込んだクラスの合計数	1795
読み込み解除したクラスの合計数	0
発生したガベージコレクションの数	3
ライブスレッドの数	8

表 2–7 Java 仮想マシン (JVM) の統計 (続き)

開始したスレッドの数	9
最大ライブスレッドカウント	8

これらの統計のほとんどは、チューニング不可能です。これらは、JVM の動作に関する情報を提供します。

JVM に関するチューニング情報のもう 1 つの情報源は、JVM を監視および管理するための管理インタフェースを提供するパッケージ `java.lang.management` です。このパッケージの詳細については、`http://java.sun.com/j2se/1.5.0/docs/api/java/lang/management/package-summary.html` を参照してください。

Java ヒープチューニング

Web Server での Web アプリケーションのパフォーマンスは、ほかのすべての Java プログラムの場合と同じく、JVM が実行するヒープ管理に依存します。一時停止の時間とスループットの間には、トレードオフの関係が成立します。手始めに、`http://java.sun.com/docs/hotspot/index.html` から入手可能な Java HotSpot 仮想マシンのパフォーマンスドキュメントを読んでみてください。

具体的な関連ドキュメントとしては、『Tuning Garbage Collection with the 5.0 Java Virtual Machine』(`http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html`) や『Ergonomics in the 5.0 Java Virtual Machine』(`http://java.sun.com/docs/hotspot/gc5.0/ergo5.html`) などが挙げられます。

JVM オプションを管理コンソールで指定するには、構成の「Java」タブ ⇒「JVM 設定」サブタブを使用します。CLI では、`wadm` コマンド `set-jvm-prop` および `set-jvm-profiler-prop` を使用します。

Web アプリケーション情報

Web アプリケーション統計が表示されるのは、管理コンソール、`wadm get-config-stats` コマンド、および `stats-xml` を使用する場合だけです。`perfdump` ではそれらは表示されません。

▼ 管理コンソールから Web アプリケーション統計にアクセスする

1 「共通操作」ページから「監視」タブを選択します。

2 構成の **Web** アプリケーション統計を表示するには、構成名をクリックします。インスタンスの **Web** アプリケーション統計を表示するには、「インスタンス」サブタブをクリックし、インスタンス名をクリックします。

3 「監視統計」ページで、「仮想サーバーの統計」をクリックします。

4 仮想サーバー名をクリックします。

5 「仮想サーバーの監視統計」ページで「**Web** アプリケーション」をクリックします。

6 統計を表示する **Web** アプリケーションを、「**Web** アプリケーション」プルダウンメニューから選択します。

Web アプリケーションの統計

次の表に、管理コンソールで表示される Web アプリケーション統計の例を示します。

表 2–8 Web アプリケーションの統計

読み込んだ JSP の数	1
再読み込みした JSP の数	1
提供されたセッションの合計数	2
アクティブなセッションの数	2
アクティブセッションの最大数	2
拒否されたセッションの数	0
期限切れセッションの数	0
期限切れセッションが有効であった平均時間 (秒)	0
期限切れセッションが有効であった最長時間 (秒)	0

チューニング方法の詳細については、85 ページの「Java Web アプリケーションのパフォーマンスチューニング」を参照してください。また、『Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications』も参照してください。

JDBC リソース情報

JDBC リソースとは、データベースへの JDBC 接続の名前付きグループのことです。JDBC リソースは、接続プールの作成に使用されるプロパティーを定義します。各 JDBC リソースは、サーバー起動時に JDBC ドライバを使って物理データベースへの接続を確立します。接続のプールが作成されるのは、Web Server の起動後、そのプールに対して最初の接続要求が発行されたときです。

JDBC ベースのアプリケーションまたはリソースは、プールから接続を取り出し、その接続を使用し、その接続が不要になった時点でそれをクローズして接続プールに返します。2 つ以上の JDBC リソースが同じプール定義を指している場合、それらのリソースは実行時に同じ接続プールを使用します。

接続プールを使用すると、次の処理によってアプリケーションのパフォーマンスが改善されます。

- 接続を事前に作成します。接続の確立の負担が、パフォーマンスに重大な影響を及ぼすコードとは別になります。
- 接続を再利用します。接続の作成回数が大幅に低減されます。
- リソースの量を制御することによって、単一のアプリケーションがいつでも使用できます。

JDBC リソースの作成および編集は、管理コンソールの、構成に対する「Java」タブ ⇒「リソース」サブタブを使って行えます。また、 `wadm create-jdbc-resource` および `set-jdbc-resource-prop` コマンドを使用することもできます。詳細については、『Sun Java System Web Server 7.0 管理ガイド』を参照してください。

注 – Web Server の起動時に各定義済みプールがインスタンス化されます。ただし、プールへのアクセスがはじめて発生するまで、接続が作成されることはありません。プールに重い負荷をかける前に、プールを軽く使用しておくべきです。

JDBC リソースの統計が利用可能になるのは、管理コンソール、CLI、および `stats.xml` を使用する場合だけです。`perfdump` ではそれらは表示されません。一部の監視データは、管理コンソール経由では利用できず、CLI の `wadm get-config-stats` および `stats.xml` 出力経由でしか表示できません。

プールはオンデマンドで作成されます。つまり、プールははじめて使用される時点で作成されます。監視統計は、プールがはじめて使用されるまで表示されません。

管理コンソール経由で利用可能な JDBC リソース統計

次の表に、管理コンソール経由で表示される JDBC リソース統計の例を示します。

表 2–9 JDBC リソース統計

接続	32
未使用接続	0
リース接続	32
平均キュー時間	1480.00

表 2–9 JDBC リソース統計 *(続き)*

待機接続	40
接続タイムアウト	100

管理コンソール経由で JDBC リソースの設定を変更するには、対応する構成で、「Java」タブ ⇒「リソース」サブタブを選択します。JDBC リソースを選択します。「JDBC リソースを編集」ページで設定が利用可能になります。コマンド行インタフェース経由で JDBC リソースを変更するには、`wadm set-jdbc-resource-prop` を使用します。

接続数

この数値は現在の JDBC 接続数を示しており、未使用接続数とビジー接続数の両方を含みます。

チューニング – この設定はチューニング不可能ですが、プールの最新のアクティビティーを知るための良い指標になります。接続数が最小接続数よりも常に大きい場合、その現在の JDBC 接続数に近い値になるように最小接続数を増やすことを検討してください。管理コンソール経由で JDBC リソースの最小接続数を変更するには、「JDBC リソースを編集」ページで「最小接続数」設定を編集します。コマンド行インタフェース経由で JDBC リソースの最小接続数を変更するには、`wadm set-jdbc-resource-prop` を使って `min-connections` プロパティーを変更します。

未使用接続数

この数値は、プール内の未使用接続の現在数を示します。最小プールサイズを超える未使用接続はすべて、アイドル状態が最大アイドルタイムアウトを超えると閉じられます。未使用接続数はチューニング不可能です。

リース接続数

この数値は、プール内の使用中接続の現在数を示します。

チューニング – リース接続数が最小接続数よりも常に小さい場合、JDBC リソースの最小接続数を減らすことを検討してください。リース接続数が最小接続数よりも常に大きい場合、最小接続数を増やすことを検討してください。リース接続数が JDBC リソースの最大接続数と常に等しくなる場合、最大接続数を増やすことを検討してください。リース接続数の上限は最大接続数になります。

管理コンソール経由で JDBC リソースの最小接続数または最大接続数を変更するには、「JDBC リソースを編集」ページで「最小接続数」または「最大接続数」フィールドを編集します。コマンド行インタフェース経由で JDBC リソースの最小接続数または最大接続数を変更するには、`wadm set-jdbc-resource-prop` を使って `min-connections` または `max-connections` プロパティーを変更します。

待機接続数

この数値は、JDBC プールから接続を受け取るために待機している接続要求の現在数を示します。現在のリース接続数が最大接続数に達すると、接続要求がキューに入れられます。

チューニング – この数値がゼロより常に大きい場合、JDBC リソースの最大接続数を増やすことを検討してください。管理コンソール経由で JDBC リソースの最大接続数を変更するには、「JDBC リソースを編集」ページで「最大接続数」フィールドを編集します。コマンド行インタフェース経由で JDBC リソースの最大接続数を変更するには、`wadm set-jdbc-resource-prop` を使って `max-connections` プロパティーを変更します。

管理コンソールで利用不可能な JDBC リソース統計

一部の JDBC 統計は、`wadm get-config-stats` コマンド (`--node` オプションを使用)、`stats-xml`、および SNMP 経由では利用可能ですが、管理コンソール経由では利用不可能です。

maxConnections – 構成されたプールの最大サイズ。ほかの統計の基準値として使用してください。管理コンソール経由で JDBC リソースの最大接続数を変更するには、「JDBC リソースを編集」ページで「最大接続数」フィールドを編集します。コマンド行インタフェース経由で JDBC リソースの最大接続数を変更するには、`wadm set-jdbc-resource-prop` を使って `max-connections` プロパティーを変更します。

peakConnections – プールの履歴に含まれている、同時にリースされた接続の数のうちで、最大のもの。この数値は、プール使用時の上限を知るための良い指標になります。これは、最大接続数の設定によって制限されます。

countTotalLeasedConnections – プールによって接続が提供された回数の合計。プールの総合的なアクティビティーを示します。チューニング不可能です。

countTotalFailedValidationConnections – 接続の検証が有効になっている場合、プールによって接続が無効であると判定された回数を示します。この数値が比較的大きい場合、それは、データベースやネットワークで問題が発生していることを意味している可能性があります。チューニング不可能です。

peakQueued – プールの寿命内の任意の時点で同時にキューに入っていた接続要求の数のうちで、最大のもの。チューニング不可能です。

millisecondsPeakWait – 任意の接続要求が待機キュー内に存在していたミリ秒単位の時間のうちで、最長のもの。この数値が大きい場合、それは、プールのアクティビティーが活発であることを示しています。上限は、JDBC リソース設定の待機タイムアウトになります。

countConnectionIdleTimeouts – 構成された JDBC アイドルタイムアウトを超過したためにプールによって閉じられた未使用接続の数。管理コンソール経由で JDBC リソースのアイドルタイムアウトを変更するには、「JDBC リソースを編集」ページで「ア

イドルタイムアウト」フィールドを編集します。コマンド行インタフェース経由でJDBC リソースのアイドルタイムアウトを変更するには、`wadm set-jdbc-resource-prop` を使って `idle-timeout` プロパティーを変更します。

JDBC リソースの接続設定

アプリケーションのデータベースアクティビティーによっては、JDBC リソースの接続プール設定のサイズを変更しなければいけない可能性があります。パフォーマンスに影響を与える JDBC リソースの属性のリストと、その値を設定する際のパフォーマンス上の考慮点を、次に示します。

- 最小接続数

  サーバーインスタンスの寿命内でプールが維持する傾向のあるサイズ。プールの初期サイズでもあります。デフォルトは 8 です。この数値は、予期されるプールの平均サイズにできるだけ近い値にするべきです。高負荷下での使用が予期されるプールの場合、アプリケーションの寿命内での接続の作成やプールのサイズ変更が最小限に抑えられるよう、大きな数値を使用してください。小さい負荷が予期されるプールの場合、リソース消費が最小限に抑えられるよう、小さい数値を使用してください。

- 最大接続数

  プールが任意の時点で持つことのできる接続の最大数。デフォルトは 32 です。プールまたはアプリケーションが持つことのできる接続リソースの量に制限を加える場合に、この設定を使用します。また、この制限は、過剰なリソース消費によるアプリケーションエラーを回避するためにも役立ちます。

- アイドルタイムアウト

  未使用状態の接続がプール内にとどまることが保証される、秒単位の最大時間。このアイドルタイムアウトが経過すると、接続が自動的に閉じられます。必要であれば、閉じられた接続の代替として、最小接続数までの接続が新たに作成されます。この設定は、データベースサーバー側で適用される接続タイムアウトを制御するわけではありません。デフォルト値は 60 秒です。

  この属性を –1 に設定すると、接続が閉じられなくなります。この設定は、高い要求が継続的に見込まれるようなプールに適しています。それ以外の場合は、使用不可能な接続がプール内に蓄積されないよう、このタイムアウトがデータベースサーバー側のタイムアウトよりも短くなるようにしてください (特定ベンダーのデータベース上でそのようなタイムアウトが構成されている場合)。

- 待機タイムアウト

  タイムアウトの発生までに要求がキュー内で接続を待機する時間 (秒)。このタイムアウトが経過すると、ユーザーにエラーが表示されます。デフォルトは 60 です。

  この属性を 0 に設定すると、接続の要求が無制限に待機するようになります。また、この設定ではプールが接続タイマーを考慮する必要がなくなるため、パフォーマンスが改善する可能性もあります。

- 検証方法

  プール内の接続の健全性を判定する目的でプールによって使用される方法。デフォルトはオフです。

  ある検証方法が使用される場合、プールは、アプリケーションに接続をリースする前に、その接続の妥当性検査を実行します。 効果やパフォーマンスへの影響は、選択する方法ごとに異なります。

  - `meta-data` は `table` よりもパフォーマンス上のコストは低くなりますが、通常、その効果は低くなります。なぜなら、大部分のドライバが結果をキャッシュに書き込み、接続を使用しないため、間違った結果が提供されるからです。
  - `table` はほとんど常に効果的です。なぜなら、この方法では、データベースへの SQL 呼び出しの実行がドライバに強制されるからです。ただし、これはもっともコストが高い方法でもあります。
  - `auto-commit` は、効果とパフォーマンスコストとの最適バランスを提供できますが、一部のドライバではやはり、この方法の結果がキャッシュに書き込まれます。

- 検証表名

  検証方法が `table` の場合に検証に使用されるユーザー定義テーブル。デフォルトは `test` です。

  この方法を使用する場合、使用されるテーブルは検証専用にするべきであり、かつテーブル内の行数は最小限に抑えるべきです。

- すべての接続を再確立

  無効な接続が見つかった場合に、プール内のすべての接続を再作成するのか、あるいは無効な接続のみを再作成するのかを示します。ユーザーが接続検証方法を選択した場合にのみ適用されます。デフォルトは無効です。

  これを有効にした場合、すべての再作成が一度に行われ、接続を要求しているスレッドは多大な影響を受けます。これを無効にした場合、接続の再作成の負荷が、各接続を要求しているスレッド間で分散されます。

- トランザクション遮断レベル

  プール内のデータベース接続のトランザクション遮断レベルを指定します。

  デフォルトでは、接続のデフォルトの遮断レベルがそのまま使用されます。これを任意の値に設定すると、メソッド呼び出しによるパフォーマンス低下が若干発生します。

- 遮断を保証

  トランザクション遮断レベルが指定された場合にのみ適用されます。デフォルトは無効です。

  この設定を無効のままにすると、接続の作成時にのみ、遮断レベルが設定されます。有効にすると、接続がアプリケーションにリースされるたびに、レベルが設定されます。ほとんどの場合、この設定は無効のままにします。

ACL ユーザーキャッシュのチューニング

ACL ユーザーキャッシュはデフォルトで有効になっています。ACL ユーザーキャッシュは、そのデフォルトサイズ (200 件のエントリ) のために、障害の 1 つになるか、あるいは単純に高トラフィックのサイト上でその目的を果たせない可能性があります。ビジー状態のサイトでは、200 件を超えるユーザーが、ACL で保護されたリソースをキャッシュエントリの寿命内でヒットする可能性があります。そうした状況が発生すると、Web Server はユーザーを検証するために LDAP サーバーに対するクエリーをより頻繁に行う必要が生じますが、そのことがパフォーマンスに悪影響を及ぼします。

この障害を解消するには、構成の「パフォーマンス」タブ ⇒「キャッシュ」サブタブで、ACL キャッシュの最大ユーザー数を増やします。また、コマンド `wadm set-acl-cache-prop` を使って `max-users` プロパティーを設定することによっても、ユーザー数を設定することができます。キャッシュサイズを増やすとリソースの使用量も増えることに注意してください。キャッシュを大きくするほど、それを保持するために必要となる RAM も多くなります。

また、1 つのキャッシュエントリ内に格納されるグループの数 (デフォルトでは 4) が障害になる可能性もあります (ただし、その可能性は格段に低い)。5 つのグループに所属するユーザーが、ACL キャッシュの寿命内でそれらの異なるグループをチェックするための 5 つの ACL をヒットした場合、追加のキャッシュエントリが 1 つ作成され、そこに追加のグループエントリが格納されます。キャッシュエントリが 2 つ存在している場合、元のグループの情報を含むエントリは無視されます。

このパフォーマンス問題が発生することはきわめてまれですが、構成の「パフォーマンス」タブ ⇒「キャッシュ」サブタブの「最大グループ数」設定を使用すれば、単一の ACL キャッシュエントリ内にキャッシュされるグループの数のチューニングを行えます。あるいは、`wadm set-acl-cache-prop` コマンドの `max-groups-per-user` プロパティーを使用することもできます。

ACL キャッシュの最大継続時間の設定により、キャッシュエントリが期限切れになるまでの秒数が決まります。キャッシュのエントリが参照されるたびにその経過時間が計算され、最大継続時間の設定と照合されます。エントリの継続時間が最大継続時間よりも大きいかそれと等しい場合、そのエントリは使用されません。デフォルト値は 120 秒です。LDAP が頻繁に変更される可能性が低い場合には、大きい数値を最大継続時間として使用してください。これに対し、LDAP のエントリが頻繁に変更される場合には、小さい値を使用してください。たとえば、この値が 120 秒である場合、Web Server と LDAP サーバーの同期が 2 分間にわたって取られない可能性があります。環境によって、それが問題になる可能性も、ならない可能性もあります。

Java Web アプリケーションのパフォーマンスチューニング

この節には、Java Web アプリケーションのパフォーマンス改善に役立つ情報が含まれています。この節では、次の内容について説明します。

さらに次の各節も参照し、Java Web アプリケーションに関するその他のチューニング情報を確認してください。

プリコンパイルされた JSP の使用

JSP のコンパイルは、比較的長い時間のかかる、リソース集約型のプロセスです。Web Server はデフォルトで、JSP が変更されたかどうかを定期的にチェックし、それらを動的に再読み込みします。これにより、サーバーを再起動することなしに変更を配備できます。`sun-web.xml` 内の `jsp-config` 要素の `reload-interval` プロパティーは、サーバーによる JSP の変更有無のチェック頻度を制御します。ただし、このチェックを行うと、パフォーマンスが若干低下します。

サーバーがある `.jsp` ファイル内で変更を検出すると、その JSP だけの再コンパイルと再読み込みが行われます。Web アプリケーション全体の再読み込みは行われません。

JSP が変更されない場合には、それらの JSP をプリコンパイルすることで、パフォーマンスを改善できます。

管理コンソールまたは CLI のいずれかを使って Web アプリケーションを追加するときに、JSP をプリコンパイルするオプションを選択します。JSP のプリコンパイルを有効にすると、Web アプリケーション内に存在するすべての JSP がプリコンパイルされ、それらの対応するサーブレットクラスが、その Web アプリケーションの `WEB-INF/lib` または `WEB-INF/classes` ディレクトリ内にバンドルされます。ある JSP へのアクセスが発生すると、その JSP がコンパイルされるのではなく、代わりにそのプリコンパイルされたサーブレットが使用されます。JSP の詳細については、『Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications』を参照してください。また、86 ページの「クラス再読み込みの設定」も参照してください。

サーブレット/JSP キャッシュの使用

同じサーブレット/JSP の再実行に長時間を費やしている場合には、その結果をキャッシュに書き込んでおき、次回実行時にキャッシュからその結果を取り出して返すことができます。たとえば、これは、サイトを訪れたすべてのユーザーが実行する共通クエリーなどで役立ちます。クエリーの結果は日々変わる可能性があるので動的でなければいけませんが、そのロジックをユーザーごとに実行する必要はありません。

キャッシュを有効にするには、アプリケーションの `sun-web.xml` ファイル内のキャッシュパラメータを構成します。詳細については、『Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications』の「Caching Servlet Results」を参照してください。

Java セキュリティーマネージャーの設定

Web Server は Java セキュリティーマネージャーをサポートします。セキュリティーマネージャーを使って実行することの最大の欠点は、パフォーマンスに悪影響を及ぼす点です。Java セキュリティーマネージャーは、製品のインストール時にデフォルトで無効になります。セキュリティーマネージャーを使わずに実行すれば、アプリケーションの種類によっては、パフォーマンスが大幅に改善される可能性があります。実行時にセキュリティーマネージャーを使用するかどうかは、アプリケーションや配備のニーズに基づいて判断するようにしてください。詳細については、『Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications』を参照してください。

クラス再読み込みの設定

サーブレットコンテナの動的再読み込み間隔と、`sun-web.xml` 内の `class-loader` 要素の `dynamic-reload-interval` によって、サーバーがサーブレットクラスの変更有無をチェックする頻度が決まります。動的再読み込みが有効になっていて、かつある `.class` ファイルの変更をサーバーが検出した場合、その Web アプリケーション全体の再読み込みが行われます。

動的再読み込み間隔を設定するには、構成の「Java」タブ ⇒「サーブレットコンテナ」サブタブを開くか、`wadm set-servelt-container-props` コマンドを使用します。変更がスケジュールに従って行われるような本稼働環境では、この値を 0 に設定することで、サーバーが常に更新チェックを行うのを防ぎます。デフォルト値は 0 (つまりクラスの再読み込みが無効) です。`sun-web.xml` 内の要素の詳細については、『Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications 』を参照してください。

クラスパス内でのディレクトリの回避

特定のアプリケーション (特に Java セキュリティーマネージャーが有効になっている場合) では、クラスパス内に不要なディレクトリが存在しないようにすることで、パフォーマンスを改善できます。そうするには、構成の「Java」タブ ⇒「一般」サブタブにある「サーバークラスパス」、「クラスパスのプレフィックス」、および「クラスパスのサフィックス」フィールドを変更するか、コマンド `wadm set-jvm-prop` を使用します。さらに、Web アプリケーションの `.class` ファイルをそのまま `WEB-INF/classes` 内にパッケージ化する代わりに、それらの `.class` ファイルを `.jar` アーカイブとして `WEB-INF/lib` 内にパッケージ化し、`WEB-INF/classes` ディレクトリが `.war` アーカイブに含まれていないことを確認します。

Web アプリケーションのセッション設定の構成

セッションの寿命が比較的短い場合、`sun-web.xml` 内の `session-properties` 要素の下にある `timeOutSeconds` プロパティーの値をデフォルト値の 10 分から減らすことで、セッションタイムアウトを減らしてみてください。

セッションの寿命が比較的長い場合、`reapIntervalSeconds` プロパティーの値をデフォルト値の毎分 1 回から増やすことで、セッションリーパーの実行頻度を減らしてみることができます。

これらの設定やセッションマネージャーの詳細については、『Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications』を参照してください。

マルチプロセスモードで、`sun-web.xml` の `persistence-type` が `s1ws60` または `mmap` のいずれかに構成されている場合、セッションマネージャーはクロスプロセスロックを使ってセッションデータの整合性を維持します。これらを次の説明に従って設定すれば、パフォーマンスを改善できます。

注 – マルチプロセスモードは非推奨であり、下位互換性を維持するためだけに含まれています。

---

maxLocks のチューニング (UNIX/Linux)

`maxLocks` プロパティーに指定された数の意味は、`maxSessions` の値を `maxLocks` で割ってみれば理解できます。たとえば、`maxSessions = 1000` の場合に `maxLocks = 10` と設定すると、約 100 個のセッション (1000/10) が同じロック上で競合することになります。`maxLocks` を増やすと、同じロック上で競合するセッションの数が減るので、パフォーマンスが改善し、待ち時間が短縮される可能性があります。ただし、ロックの数を増やすとオープン状態のファイル記述子の数も増えます。そのため、ロックの数を増やさなければ受信接続要求に割り当てられるはずだった利用可能な記述子の数が、減ることになります。

これらの設定の詳細については、『Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications』の第 6 章「Session Managers」を参照してください。

MMapSessionManager のチューニング (UNIX/Linux)

次の例では、`manager-properties` プロパティーを使って `persistence-type="mmap"` と構成する場合に、プロセスサイズにどのような影響があるかについて説明します。詳細については、『Sun Java System Web Server 7.0 Developer's Guide to Java Web Applications』の「MMap Session Manager (UNIX Only)」を参照してください。

```
maxSessions = 1000
maxValuesPerSession = 10
maxValueSize = 4096
```

この例の場合、サイズ 1000 X 10 X 4096 バイト、つまり約 40M バイトのメモリーマップファイルが作成されます。これはメモリーマップファイルなので、起動時のプロセスサイズが 40M バイトだけ増えます。これらのパラメータに設定する値が大きいほど、プロセスサイズの増加量も増えます。

CGI スタブプロセスのチューニング (UNIX/Linux)

Web Server では、CGI エンジンが必要に応じて CGI スタブプロセスを作成します。CGI によって生成されるコンテンツに大きく依存する高負荷システムでは、CGI スタブプロセスがすべてのシステムリソースを消費してしまう可能性があります。サーバーでそうした状況が発生している場合、CGI スタブプロセスのチューニングを行うことで、作成可能な新しい CGI スタブプロセスの数、そのタイムアウト値、および任意の時点で実行される CGI スタブプロセスの最小数を制限することができます。

注 – `magnus.conf` ファイル内に `init-cgi` 関数が存在していて、かつサーバーがマルチプロセスモードで稼働している場合、その `init-cgi` 行に `LateInit = yes` を追加する必要があります。

マルチプロセスモードは非推奨であり、将来のリリースでは存在しない可能性があります。

CGI スタブを制御するには、次の設定をチューニングしてください。これらの設定は、構成の「パフォーマンス」タブ ⇒「CGI」サブタブにあります。

- 最小スタブサイズ: デフォルトで起動されるプロセスの数を制御します。CGI プログラムへのアクセスが発生するまで、CGI スタブプロセスは 1 つも起動されません。デフォルト値は 0 です。`magnus.conf` ファイルに `init-cgi` 指令が含まれている場合は、この最小数の CGI スタブプロセスは起動時に生成されます。

- 最大スタブサイズ: サーバーが生成できる CGI スタブプロセスの最大数を制御します。これは、実行中の CGI スタブプロセスの最大同時数であり、保留されている要求の最大数ではありません。デフォルト値は 16 ですが、ほとんどのシステムではこの値で十分です。この設定を大きくしすぎると、実際にはスループットが減少してしまう可能性があります。
- CGI スタブタイムアウト: この指令で設定された秒数の間、アイドル状態のままになっているすべての CGI スタブプロセスを終了するように、サーバーに指示します。プロセス数が最小スタブサイズに達すると、それ以上プロセスが終了されなくなります。デフォルトは 30 です。
- CGI タイムアウト: CGI プロセスに許される最大実行時間を秒単位で制限します。デフォルトは –1 ですが、これは、タイムアウトがないことを意味します。

find-pathinfo-forward の使用

obj.conf 内で find-pathinfo-forward パラメータを使用すると、パフォーマンスの改善を図りやすくなります。これは、PathCheck 関数 find-pathinfo、および NameTrans 関数 pfx2dir および assign-name で使用されます。find-pathinfo-forward パラメータは、PATH_INFO の検索を、サーバー関数 find-pathinfo 内のパスの末尾から逆方向に行う代わりに、ntrans-base のあとのパス内で順方向に行うように、サーバーに指示します。

注 – サーバー関数 find-pathinfo の呼び出し時に ntrans-base パラメータが rq->vars 内に設定されていないと、サーバーは find-pathinfo-forward パラメータを無視します。ntrans-base はデフォルトで設定されています。

例

```
NameTrans fn="pfx2dir" find-pathinfo-forward="" from="/cgi-bin"
dir="/export/home/cgi-bin" name="cgi"
NameTrans fn="assign-name" from="/perf"
find-pathinfo-forward="" name="perf"
```

この機能を使えば、サーバー関数 find-pathinfo での stat の実行回数が少なくなるため、特定 URL でのパフォーマンスの改善を図れます。また、Windows の場合、PathCheck サーバー関数 find-pathinfo の使用時にサーバーが「\\」を「/」に変更するのを回避するためにこの機能を使用することもできます。

obj.conf の詳細については、『Sun Java System Web Server 7.0 Administrator's Configuration File Reference』を参照してください。

nostat の使用

obj.conf の NameTrans 関数 assign-name でパラメータ nostat を指定すれば、指定された URL 上でサーバーが stat を実行するのを可能なかぎり回避できます。構文は次のとおりです。

```
nostat=virtual-path
```

例

```
<Object name=default>
NameTrans fn="assign-name" from="/nsfc" nostat="/nsfc" name="nsfc"
</Object>
<Object name=nsfc>
Service fn=service-nsfc-dump
</Object>
```

この例では、*ntrans-base* が設定されている場合、パス */ntrans-base*/nsfc および */ntrans-base*/nsfc/* でサーバーは stat を実行しません。*ntrans-base* が設定されていない場合、URL /nsfc および /nsfc/* でサーバーは stat を実行しません。*ntrans-base* はデフォルトで設定されています。この例では、デフォルトの PathCheck サーバー関数が使用されるものと仮定しています。

assign-name NameTrans で nostat=*virtual-path* を使用すると、指定された *virtual-path* 上で stat を実行しても失敗すると、サーバーは仮定します。したがって、nostat を使用するのは、NSAPI プラグインの URL のように、*virtual-path* のパスがシステム上に存在しない場合だけにしてください。そうした URL で nostat を使用すると、そうした URL での不要な stat 実行が避けられるため、パフォーマンスが改善されます。

obj.conf の詳細については、『Sun Java System Web Server 7.0 Administrator's Configuration File Reference』を参照してください。

ビジー関数の使用

デフォルトのビジー関数は「503 Service Unavailable」応答を返し、ログレベルの設定に応じたメッセージをログに記録します。アプリケーション向けにこの動作を変更することをお勧めします。obj.conf ファイル内の任意の NSAPI 関数でユーザー独自のビジー関数を指定するには、その構成ファイル内でサービス関数を次の形式で含めます。

```
busy="my-busy-function"
```

たとえば、次のサンプルサービス関数を使用できます。

```
Service fn="send-cgi" busy="service-toobusy"
```

これにより、いくつかのタイプ (Service、AddLog、PathCheck など) を含む要求処理の途中でサーバーが深刻なビジー状態に陥った場合に、さまざまな応答を返すことが可能となります。デフォルトのスレッドタイプが非ネイティブである場合、実行にネイティブスレッドを必要とするすべての関数にビジー関数が適用されます。

サーバー全体で、デフォルトのビジー関数の代わりにユーザー独自のビジー関数を使用するには、次のような、func_insert 呼び出しを含む NSAPI init 関数を記述します。

```
extern "C" NSAPI_PUBLIC int my_custom_busy_function
(pblock *pb, Session *sn, Request *rq);
my_init(pblock *pb, Session *, Request *){func_insert
("service-toobusy", my_custom_busy_function);}
```

プールスレッド上では、ビジー関数が実行されることは決してないため、スレッドのブロックを引き起こす可能性のある関数呼び出しを使用しないように注意する必要があります。

◆ ◆ ◆ 第 3 章

3 一般的なパフォーマンスの問題

この章では、Web サイトの一般的なパフォーマンスの問題について説明します。この章の内容は、次のとおりです。

注 – プラットフォーム固有の問題については、第 4 章を参照してください。

check-acl Server Application Function

サーバーの最適なパフォーマンスを得るために、ACL は、必要な場合にのみ使用してください。

サーバーには、サーバーへの書き込みアクセスを `all` にのみ許可するデフォルト ACL を含む ACL ファイルと、`anybody` への書き込みアクセスを制限するための `es-internal` ACL が設定されています。後者の ACL によって、サーバー内のマニュアル、アイコン、および検索 UI ファイルが保護されます。

デフォルトの `obj.conf` ファイルには、読み取り専用にする必要のあるディレクトリを `es-internal` オブジェクトにマップするための `NameTrans` 行が含まれています。このオブジェクトには、さらに、`es-internal` ACL のための `check-acl` SAF が含まれています。

また、デフォルトオブジェクトにも、デフォルト ACL のための `check-acl` SAF が含まれています。

ACL によって保護されていない URI のデフォルトオブジェクトから `check-acl` SAF を削除することによって、パフォーマンスを向上させることができます。

メモリー不足の状況

Web Server をメモリー不足の状況で実行する必要がある場合は、構成の「パフォーマンス」タブ ⇒「HTTP」サブタブで「最大スレッド数」の設定値を小さくすることによって、スレッドの制限を最低まで下げます。また、`wadm set-thread-pool-prop` コマンドの `max-threads` プロパティーを使用してこの値を設定することもできます。

高負荷の下で Web アプリケーションを実行していると、サーバーが Java VM ランタイムヒープスペース不足に陥る場合があります。この状態は、サーバーログファイル内の `java.lang.OutOfMemoryError` メッセージで示されます。この原因は、オブジェクトの過剰な割り当てなど、いくつか考えられますが、このような動作がパフォーマンスに影響を与える可能性があります。この問題に対処するには、アプリケーションのプロファイルを作成します。アプリケーションの割り当て (オブジェクトやそのサイズ) のプロファイリングに関する注意事項については、次の HotSpot VM のパフォーマンス FAQ を参照してください。

`http://java.sun.com/docs/hotspot/index.html`

場合によっては、アプリケーションが最大セッション数を使い切り (サーバーログファイル内の「too many active sessions」のメッセージで示される)、そのためコンテナが例外をスローすることで、さらにアプリケーションパフォーマンスに影響を与えることがあります。この状況に対処するには、セッションマネージャーのプロパティー、セッション作成アクティビティー (JSP のセッションはデフォルトで有効になることに注意)、およびセッションのアイドル時間を検討する必要があります。

少なすぎるスレッド

サーバーでは、アクティブなスレッドの数がスレッドの制限値を超えることは許可されません。同時要求の数がその制限に達すると、サーバーは、古い接続が解放されるまで新しい接続へのサービスを停止します。これによって、応答時間が増加する場合があります。

Web Server では、サーバーの最大スレッド数のデフォルト設定は 128 です。サーバーにより多くの要求を同時に処理させる場合は、スレッドの最大数を増やしてください。

サーバーのスレッドが少なすぎる場合の症状は、長い応答時間です。ブラウザから要求を発行するとサーバーへの接続はかなりすばやく確立されますが、サーバーに存在するスレッドが少なすぎる場合は、応答がクライアントに戻ってくるまでに長い時間がかかる可能性があります。

スレッドが少なすぎるためにサーバーが減速しているかどうかを検出するための最適な方法は、アクティブなセッションの数がスレッドの最大数に近いか、または等しくなっているかどうかを調べることです。これを実行するには、63 ページの「セッション作成 (スレッド) 情報」を参照してください。

キャッシュが活用されていない

ファイルキャッシュが活用されていない場合は、サーバーが最適に実行されていません。ほとんどのサイトには、常にキャッシュ可能であるはずの多数の GIF または JPEG ファイルが存在するため、キャッシュを効率的に使用する必要があります。

ただし、サイトによっては、ほぼすべての処理を CGI、SHTML、またはその他の動的なソースによって実行している場合があります。一般に、動的コンテンツはキャッシュ可能ではないため、本質的にキャッシュヒット率は低くなります。サイトのキャッシュヒット率が低い場合でも、それほど心配する必要はありません。もっとも重要な点は、応答時間が短いことです。キャッシュヒット率が非常に低くても、応答時間は依然として非常に良好な場合があります。応答時間が良好であるかぎり、キャッシュヒット率が低いことを心配する必要はありません。

`perfdump`、管理コンソールの「監視」タブ、または `wadm stats` コマンドからの統計情報を使用して、ヒット率をチェックしてください。ヒット率は、キャッシュが使用された回数の、サーバーへのすべてのヒットに対するパーセンテージです。50% を超えていれば、キャッシュヒット率は良好です。サイトによっては、98% 以上を達成している場合もあります。詳細については、65 ページの「ファイルキャッシュ情報 (静的コンテンツ)」を参照してください。

さらに、多数の CGI または NSAPI 呼び出しを実行している場合、キャッシュヒット率が低くなる可能性があります。また、カスタム NSAPI 関数を使用している場合も、キャッシュヒット率が低くなる可能性があります。

キープアライブ接続がフラッシュされる

キープアライブ接続なしで 1 秒あたり 75 の要求を処理できる可能性のある Web サイトは、キープアライブを有効にすると、1 秒あたり 200 ～ 300 の要求を処理できる可能性があります。そのため、クライアントが 1 つのページにさまざまな項目を要求する場合は、キープアライブ接続が効率的に使用されていることが重要です。`perfdump` で示される `KeepAliveCount` (管理コンソールに表示される「追加した接続の合計数」) がキープアライブ最大接続数を超えると、以降のキープアライブ接続は、キープアライブの状態を維持されずに閉じられます (「フラッシュされる」)。

`perfdump` からの統計情報を使用して `KeepAliveFlushes` および `KeepAliveHits` 値をチェックするか、または「監視統計」ページの「キープアライブ統計」の下にある「フラッシュした接続の数」と「処理した接続の数」をチェックしてください。詳細については、58 ページの「キープアライブ情報」を参照してください。

キープアライブ接続が適切に実行されているサイトでは、KeepAliveFlushes と KeepAliveHits の比率は非常に低くなります。この比率が (1:1 より) 高い場合、そのサイトではおそらく、キープアライブ接続が期待されるほどは適切に活用されていません。

キープアライブのフラッシュを削減するには、キープアライブ最大接続数 (構成の「パフォーマンス」タブ ⇒「HTTP」サブタブ、または wadm set-keep-ailve props コマンドで設定される) を増やしてください。デフォルト値は 200 です。この値を増やすことによって、より多くの待機中のキープアライブ接続が開いたままになります。

注意 – UNIX/Linux システムでは、キープアライブ最大接続数の値が大きすぎると、サーバーのオープンファイル記述子が不足する場合があります。一般に、UNIX/Linux では 1024 が開かれたファイルの制限であるため、この値を 500 を超えて増やすことはお勧めできません。

ログファイルモード

ログファイルを高い冗長レベルに維持すると、パフォーマンスに大きな影響を与える可能性があります。構成の「一般」タブ ⇒「ログ設定」ページで、適切なログレベルを選択し、「詳細」、「より詳細」、および「もっとも詳細」などのレベルを慎重に使用してください。CLI を使用してログレベルを設定するには、コマンド wadm set-log-prop を使用し、log-level を設定します。

◆ ◆ ◆ 第 4 章

4 プラットフォーム固有の問題と注意事項

この章では、プラットフォーム固有のチューニングの注意事項について説明します。この章の内容は、次のとおりです。

Solaris プラットフォーム固有の問題

ここでは、その他の Solaris 固有の問題やチューニングの注意事項について説明します。この章の内容は、次のとおりです。

1 つのプロセスで開いているファイルの数 (ファイル記述子の制限)

1 つのプロセスで一度に開くことのできるファイルの数に対する制限は、プラットフォームごとに異なります。ビジー状態のサイトでは、その数を増やすことが必要な場合があります。Solaris システムでは、`/etc/system` ファイル内の `rlim_fd_max` を設定することによってこの制限を制御します。Solaris 8 の場合、デフォルトは 1024

であり、65536 まで増やすことができます。Solaris 9 および 10 の場合、デフォルトは 65536 であり、この値を増やす必要はありません。

/etc/system ファイルでこの変更や、その他の任意の変更を行ったあと、新しい設定を有効にするには Solaris を再起動します。さらに、新しいバージョンの Solaris にアップグレードした場合は、/etc/system に追加した行をすべて削除し、その行がまだ有効なことを確認したあとでのみ、もう一度追加するようにしてください。

この変更を行う別の方法に、ulimit –n "*value*" コマンドの使用があります。このコマンドを使用した場合、システムの再起動は必要ありません。ただし、このコマンドではログインシェルだけが変更されます。これに対して、etc/system ファイルの編集はすべてのシェルに影響します。

HTTP サーバーへの接続の失敗

サーバーの負荷が高いときに、ブラウザから Web Server への接続タイムアウトが発生している場合は、HTTP リスナーのバックログキューのサイズを増やすことができます。この設定を増やすには、HTTP リスナーの待機キューの値を編集します。

また、この設定に加えて、Solaris TCP/IP ネットワークコード内の制限も増やしてください。次のコマンドを実行することによって変更される 2 つのパラメータが存在します。

```
/usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q 8192
/usr/sbin/ndd -set /dev/tcp tcp_conn_req_max_q0 8192
```

これらの 2 つの設定によって、待機中の接続でいっぱいになる可能性のある 2 つの Solaris 待機キューの最大数が増えます。tcp_conn_req_max_q によって、accept() 呼び出しからの復帰を待っている完了した接続の数が増えます。tcp_conn_req_max_q0 によって、ハンドシェークが未完了の接続の最大数が増えます。デフォルト値はそれぞれ、128 と 1024 です。これらの ndd コマンドがシステムの再起動のたびに自動的に実行されるようにするには、そのコマンドを /etc/init.d/network-tuning という名前のファイルに格納し、そのファイルへのリンクを /etc/rc2.d/S99network-tuning という名前で作成します。

これらの変更の効果は、netstat -s コマンドを使用し、tcpListenDrop、tcpListenDropQ0、および tcpHalfOpenDrop 値を確認することによって監視できます。調整を行う前に、これらの値を確認してください。これらの値が 0 でない場合は、最初に 2048 に調整し、netstat 出力の監視を続けます。

Web Server HTTP リスナーの待機キューの設定と、それに関連する Solaris の tcp_conn_req_max_q と tcp_conn_req_max_q0 の設定は Web Server のスループットに一致するはずです。これらのキューは、Web ユーザーから受信した接続の不規則な比率を管理するための「バッファー」として機能します。これらのキューによって、Solaris はそれらの接続を受け付け、それらが Web Server によって処理されるまで保持することができます。

Web Server が処理できる量より多くの接続を受け付ける必要はありません。これらのキューのサイズを制限し、過剰な接続を受け付けたために接続に対応できなくなるより過剰な接続を拒否することをお勧めします。一般に、これらの 3 つのパラメータの値を 2048 に設定すると接続要求の失敗が減少し、4096 までの値でパフォーマンス向上が確認されています。

この調整はどの Web ホスティング環境にも悪影響を与えないと予測されているため、システムが前述の症状を示していない場合でも、この提案を検討できます。

接続拒否のエラー

負荷の高いサーバーで接続拒否のエラーが発生している場合は、そのサーバー上でのネットワーク資源の使用をチューニングすることができます。

TCP/IP 接続が閉じられていると、そのポートは `tcp_time_wait_interval` (デフォルト値は 240000 ミリ秒) の間再利用されません。これは、残されたセグメントが発生しないようにするためです。`tcp_time_wait_interval` が小さければ小さいほど、貴重なネットワーク資源が早くふたたび使用可能になります。このパラメータは、次のコマンドを実行することによって変更されます。ただし、60000 未満にはしないでください。

```
usr/sbin/ndd -set /dev/tcp tcp_time_wait_interval 60000
```

この `ndd` コマンドがシステムの再起動のたびに自動的に実行されるようにするには、そのコマンドを `/etc/init.d/network-tuning` という名前のファイルに格納し、そのファイルへのリンクを `/etc/rc2.d/S99network-tuning` という名前で作成します。

システムが先に述べた症状を示しておらず、TCP プロトコルのチューニングにも精通していない場合は、前述のパラメータを変更しないようにすることをお勧めします。

TCP バッファリングのチューニング

継続的に負荷の高いサーバーからのネットワーク応答に、予測不可能な速度低下がときどき見られる場合は、`/etc/system` ファイルに次の行を追加することによって `sq_max_size` パラメータの設定を調査することもできます。

```
set sq_max_size=512
```

この設定は、パケットをハードウェアドライバから TCP/IP プロトコルドライバに転送する同期キューのサイズを調整します。512 の値を使用すると、キューがオーバーフローすることなく、大量のネットワークトラフィックを格納できるようになります。

Solaris Network Cache and Accelerator (SNCA) の使用

Solaris Network Cache and Accelerator (SNCA) は、Solaris オペレーティングシステムでの Web パフォーマンスを向上させるキャッシュサーバーです。

Web Server が実行されているシステムでは SNCA が設定されていることが前提になります。SNCA およびその構成とチューニングの詳細については、各システムの次のマニュアルページを参照してください。

- `ncab2clf(1)`
- `ncakmod(1)`
- `nca(1)`
- `snca(1)`
- `nca.if(4)`
- `ncakmod.conf(4)`
- `ncalogd.conf(4)`

▼ SNCA を Web Server と連動させる

前述したように、この手順は SNCA が設定されていることを前提にしています。

1. 「共通操作」ページで、構成を選択し、「構成を編集」をクリックします。
2. 「**HTTP** リスナー」タブをクリックし、編集する **HTTP** リスナーを選択します。
3. 「**HTTP** リスナーの編集」ページで、「プロトコルファミリ」を `nca` に設定します。
   これが機能するには、HTTP リスナーがポート 80 上で待機している必要があります。
4. 変更を保存します。
5. 「パフォーマンス」タブをクリックします。
6. 「キャッシュ」サブタブをクリックします。
7. 「キャッシュ設定」ページで、ファイルキャッシュが有効になっていることを確認し、「**Sendfile** を使用」を有効にします。
8. 変更を保存します。
9. 変更を有効にするには、構成を再配備します。

最大スレッド数とキューサイズ

Web Server を SNCA とともに使用するように設定する場合は、スレッドプールを無効にするとパフォーマンスが向上します。これらの設定は、構成の「パフォーマンス」タブ ⇒「HTTP」サブタブの「スレッドプール設定」の下にあります。スレッドプールを無効にするには、「スレッドプール」の「有効」チェックボックスを選択解除します。また、`wadm set-thread-pool-prop` コマンドの `enabled` プロパティーを使用してスレッドプールを無効にすることもできます。

SNCA 以外の構成で、特に、キープアライブなしで短い待ち時間の応答を実現する必要がある場合にも、スレッドプールを無効にすることができます。

Solaris ファイルシステムのチューニング

ここでは、ファイルシステムのチューニングのために行うことのできる変更について説明します。この節は次の問題に対応したトピックを含みます。

- 101 ページの「ファイルシステムのページイン率が高い」
- 102 ページの「ファイルシステムの状態監視の削減」
- 102 ページの「ビジー状態のディスクまたはボリュームのサービス時間が長い」

次のパラメータの説明を注意して読んでください。その説明が状況に一致している場合は、調整を行うことを検討します。

ファイルシステムのページイン率が高い

Solaris 8 または 9 でファイルシステムのページイン率が高い場合は、`segmap_percent` の値を増やすと効果が得られる可能性があります。このパラメータは、`/etc/system` ファイルに次の行を追加することによって設定されます。

```
set segmap_percent=25
```

`segmap_percent` は、カーネルがファイルシステムキャッシュ用のアドレス空間にマップするメモリーのパーセンテージを調整します。デフォルト値は 12 です。つまり、カーネルは、ファイルシステムキャッシュ用にメモリーの最大 12% をマップするための十分な空間を予約します。4G バイトの物理メモリーを備えた負荷の高いマシンでは、60 までの値でパフォーマンス向上が確認されています。25 程度から始めて、この値を試すようにしてください。大量の物理メモリーを備えたシステムでは、カーネルメモリーの要件を大幅に増加させる場合があるため、この値は少しずつ増やすようにしてください。

ファイルシステムの状態監視の削減

UNIX ファイルシステム (UFS) ボリュームには、各ファイルがアクセスされた時刻が保持されています。次の変更を行なっても、ファイルを変更したときのアクセス更新時刻は無効にならず、ファイルにアクセスしたときのアクセス更新時刻だけが無効になることに注意してください。使用する環境でファイルのアクセス更新時刻が重要でない場合は、`/etc/vfstab` 内のデータボリュームのマウントポイントに `noatime` パラメータを追加することによって無効にすることもできます。次に例を示します。

```
/dev/dsk/c0t5d0s6 /dev/rdsk/c0t5d0s6 /data0 ufs 1 yes noatime
```

ビジー状態のディスクまたはボリュームのサービス時間が長い

Web Server の応答性は、ディスクサブシステムのパフォーマンスに大きく依存します。ディスクのビジー状態と、入出力要求が完了する速度を監視するには、`iostat` ユーティリティーを使用します (それぞれ、`%b` および `svc_t` 列)。ビジー状態が約 30% 未満のディスクではサービス時間は重要でありませんが、それよりビジー状態の高いディスクではサービス時間が約 20 ミリ秒を超えないようにするべきです。ビジー状態のディスクのサービス時間が長い場合は、ディスクのパフォーマンス向上が Web Server のパフォーマンス向上に大きく役立つ可能性があります。

最初の手順は、負荷を均衡させることです。一部のディスクがビジー状態にある一方で、ほかのディスクの負荷が軽い場合は、ビジー状態のディスクの一部のファイルをアイドル状態のディスクに移動します。不均衡がある場合は、通常、それを修正するほうが過負荷のディスクをチューニングしようとするよりはるかにメリットがあります。

Solaris プラットフォーム固有のパフォーマンス監視

ここでは、システム動作の監視に使用できる Solaris 固有のツールおよびユーティリティーのいくつかについて説明します。この章の内容は、次のとおりです。

ここで説明されているツールは、Web Server によって生成される負荷にシステムがどのように応答するかという観点からパフォーマンスを監視します。Web Server の独自の機能を使用して、ユーザーが Web Server 自体に出した要求を追跡する方法については、24 ページの「サーバーパフォーマンスの監視」を参照してください。

短期的なシステム監視

Solaris は、システム動作の「スナップショット」を作成するためのいくつかのツールを提供しています。その出力をファイル内に取得してあとで分析することは可能ですが、次に示すツールは、主にリアルタイムでシステム動作を監視することを目的としています。

- `iostat -x 60` コマンドは、ディスクパフォーマンス統計を 60 秒間隔で報告します。

  各ディスクがどれだけの時間ビジー状態にあるかを調べるには、`%b` 列を監視します。ビジー状態がその時間の約 20% を超えるすべてのディスクでは、`svct` 列で報告されるサービス時間に注意してください。ほかの列では、入出力処理率、データ転送量などが報告されます。

- `vmstat 60` コマンドは、仮想記憶アクティビティーと一部の CPU 統計情報を 60 秒間隔で集計します。

  ページ走査率を追跡し、それが高すぎる場合に対処するには、`sr` 列を監視します (ここでいう「高すぎる」率は、Solaris 8 および 9 とそれ以前のリリースで非常に異なることに注意)。CPU がどれだけの負荷で使用されているかを調べるには、`us`、`sy`、および `id` 列を監視します。アクティビティーの突然のバーストに対処するには、十分な CPU パワーを確保しておく必要があることに注意してください。また、CPU 時間について競合しているスレッドの数を調べるために、`r` 列も監視してください。この数が CPU の数の約 4 倍を超えたままになっている場合は、サーバーの並行性の削減が必要になることがあります。

- `mpstat 60` コマンドが CPU 統計情報の詳細を提供するのに対して、`netstat -i 60` コマンドはネットワークアクティビティーを集計します。

長期的なシステム監視

前述のツールを使用してシステム性能を「抜き打ち検査する」だけでなく、傾向を検出できるように、より長期的なパフォーマンス履歴を収集することが重要です。少なくとも、正常に実行されているシステムのベースライン記録によって、システムのパフォーマンスが低下し始めた場合に、変更があった内容を調査するのに役立つ可能性があります。システムアクティビティーのレポートパッケージを有効にするには、次の手順に従います。

- ファイル `/etc/init.d/perf` を編集し、このファイルの終わりに近い行から `#` のコメント文字を削除します。Solaris 10 では、次のコマンドを実行します。

  ```
  svcadm enable system/sar
  ```

- コマンド `crontab -e sys` を実行し、`sa1` および `sa2` コマンドを含む行から `#` のコメント文字を削除します。また、サイトのアクティビティープロファイルによっては、これらのコマンドを実行する頻度や時刻の調整も必要になる可能性があります。このファイルの形式の説明については、`crontab` のマニュアルページを参照してください。

これにより、パフォーマンスデータは /var/adm/sa ディレクトリ内のファイルに格納されるようになります。デフォルトでは、これらのファイルはここに 1 か月間保持されます。次に、sar コマンドを使用して、目的の期間の統計情報を検査することができます。

「インテリジェント」監視

SE ツールキットは、Sun のパフォーマンス専門家によって開発された、無料でダウンロード可能なソフトウェアパッケージです。生のパフォーマンス統計の収集と監視に加えて、このツールキットでは、システムの全体的な健全性や、調整を必要とする重要な領域を特徴づけるために、ヒューリスティックを適用できます。このツールキットとマニュアルは、次の場所からダウンロードできます。

http://www.sunfreeware.com/setoolkit.html

Solaris 10 プラットフォーム固有のチューニング情報

DTrace は、Solaris オペレーティング環境用の総合的な動的トレースフレームワークです。DTrace ツールキットを使用してシステムを監視できます。このツールキットは、次の URL から入手できます。

http://www.opensolaris.org/os/community/dtrace/dtracetoolkit/

パフォーマンスベンチマークのための Solaris のチューニング

次の表は、パフォーマンスとスケーラビリティーのベンチマークに使用される、Solaris 用のオペレーティングシステムチューニングを示しています。これらの値は、希望する結果を達成するためにシステムをチューニングする場合の例を示しています。

表 4–1 パフォーマンスベンチマークのための Solaris のチューニング

パラメータ	スコープ	デフォルト値	チューニング値	コメント
rlim_fd_max	/etc/system	65536	65536	オープンファイル記述子の制限を処理します。関連付けられたソケット、ファイル、パイプ (存在する場合) などで予測される負荷を考慮するべきです。

表 4–1　パフォーマンスベンチマークのための Solaris のチューニング　　(続き)

パラメータ	スコープ	デフォルト値	チューニング値	コメント
sq_max_size	/etc/system	2	0	ストリームドライバのキューサイズを制御します。0 に設定すると無限大になるため、バッファー領域の不足によってパフォーマンスが影響されることはありません。クライアントにも設定します。sq_max_size を 0 に設定すると、ネットワークトラフィック負荷の高い本稼動システムにとって最適ではない可能性があることに注意してください。
tcp_time_wait_interval	ndd /dev/tcp	240000	60000	クライアントにも設定します。
tcp_conn_req_max_q	ndd /dev/tcp	128	1024	
tcp_conn_req_max_q0	ndd /dev/tcp	1024	4096	
tcp_ip_abort_interval	ndd /dev/tcp	480000	60000	
tcp_keepalive_interval	ndd /dev/tcp	7200000	900000	アクセスの多い Web サイトでは、この値を低く設定します。
tcp_rexmit_interval_initial	ndd /dev/tcp	3000	3000	再転送率が 30 〜 40% を超える場合は、必ずこの値を大きくしてください。
tcp_rexmit_interval_max	ndd /dev/tcp	240000	10000	
tcp_rexmit_interval_min	ndd /dev/tcp	200	3000	
tcp_smallest_anon_port	ndd /dev/tcp	32768	1024	クライアントにも設定します。
tcp_slow_start_initial	ndd /dev/tcp	1	2	データが少量であればやや高速に転送します。
tcp_xmit_hiwat	ndd /dev/tcp	8129	32768	送信バッファーを増やすため。
tcp_recv_hiwat	ndd /dev/tcp	8129	32768	受信バッファーを増やすため。

パフォーマンスベンチマークのための UltraSPARC® T1 ベースシステムのチューニング

チューニング可能なパラメータとその他のパラメータの組み合わせを使用して、パフォーマンスベンチマークのためにシステムをチューニングします。これらの値は、希望する結果を達成するためにシステムをチューニングする場合の例を示しています。

オペレーティングシステムと TCP 設定のチューニング

次の表は、UtraSPARC T1 ベースシステム (64 ビットシステム) 上のパフォーマンスとスケーラビリティーのベンチマークに使用される、Solaris 10 用のオペレーティングシステムチューニングを示しています。

表 4–2　パフォーマンスベンチマークのための 64 ビットシステムのチューニング

パラメータ	スコープ	デフォルト値	チューニング値	コメント
rlim_fd_max	/etc/system	65536	260000	オープンファイル記述子の制限を処理します。関連付けられたソケット、ファイル、パイプ (存在する場合) などで予測される負荷を考慮するべきです。
hires_tick	/etc/system		1	
sq_max_size	/etc/system	2	0	ストリームドライバのキューサイズを制御します。0 に設定すると無限大になるため、バッファー領域の不足によってパフォーマンスが影響されることはありません。クライアントにも設定します。sq_max_size を 0 に設定すると、ネットワークトラフィック負荷の高い本稼動システムにとって最適ではない可能性があることに注意してください。
ip:ip_squeue_bind			0	
ip:ip_squeue_fanout			1	
ipge:ipge_taskq_disable	/etc/system		0	
ipge:ipge_tx_ring_size	/etc/system		2048	
ipge:ipge_srv_fifo_depth	/etc/system		2048	
ipge:ipge_bcopy_thresh	/etc/system		384	
ipge:ipge_dvma_thresh	/etc/system		384	
ipge:ipge_tx_syncq	/etc/system		1	
tcp_conn_req_max_q	ndd /dev/tcp	128	3000	
tcp_conn_req_max_q0	ndd /dev/tcp	1024	3000	

表 4–2　パフォーマンスベンチマークのための 64 ビットシステムのチューニング　(続き)

パラメータ	スコープ	デフォルト値	チューニング値	コメント
tcp_max_buf	ndd /dev/tcp		4194304	
tcp_cwnd_max	ndd/dev/tcp		2097152	
tcp_xmit_hiwat	ndd /dev/tcp	8129	400000	送信バッファーを増やすため。
tcp_recv_hiwat	ndd /dev/tcp	8129	400000	受信バッファーを増やすため。

IPGE ドライバのバージョンは 1.25.25 にしてください。

ディスク構成

HTTP アクセスがログに記録されている場合は、ディスクに関する次のガイドラインに従ってください。

- アクセスログをより高速なディスクまたは接続されたストレージに書き込みます。
- 複数のインスタンスを実行している場合は、インスタンスごとのログをできるだけ別々のディスクに移動します。
- ディスクの読み取り/書き込みキャッシュを有効にします。ディスクの書き込みキャッシュを有効にすると、そのディスクに障害が発生した場合、一部の書き込みが失われる可能性があることに注意してください。
- 次のオプションを使用してディスクをマウントすることを検討します。これにより、ディスクパフォーマンスが向上する可能性があります。`nologging`、`directio`、`noatime`。

ネットワーク構成

複数のネットワークインタフェースカードが使用されている場合は、ネットワーク割り込みがすべて同じコアに行かないようにする必要があります。割り込みを無効にするには、次のスクリプトを実行します。

```
allpsr=‘/usr/sbin/psrinfo | grep -v off-line | awk ’{ print $1 }’‘
   set $allpsr
   numpsr=$#
   while [ $numpsr -gt 0 ];
   do
       shift
       numpsr=‘expr $numpsr - 1‘
       tmp=1
       while [ $tmp -ne 4 ];
```

```
        do
            /usr/sbin/psradm -i $1
            shift
            numpsr=‘expr $numpsr - 1‘
            tmp=‘expr $tmp + 1‘
        done
    done
```

すべてのネットワークインタフェースを 1 つのグループに配置します。次に例を示します。

```
$ifconfig ipge0 group webserver
$ifconfig ipge1 group webserver
```

Web Server の起動オプション

場合によっては、大きいページサイズの使用によってパフォーマンスが向上することがあります。32 ビットの Web Server を 4M バイトページで起動するには、次のコマンドを入力します。

```
LD_PRELOAD_32=/usr/lib/mpss.so.1 ; export LD_PRELOAD_32; export MPSSHEAP=4M;
./bin/startserv; unset LD_PRELOAD_32; unset MPSSHEAP
```

64 ビットサーバーの場合は、次のコマンドを入力します。

```
LD_PRELOAD_64=/usr/lib/64/mpss.so.1; export LD_PRELOAD_64; export MPSSHEAP=4M;
./bin/startserv; unset LD_PRELOAD_64; unset MPSSHEAP
```

◆ ◆ ◆ 第 5 章

サーバーのサイジングとスケーリング

この章では、サーバーのサブシステムを検査するとともに、最適なパフォーマンスのための推奨事項を提供します。この章の内容は、次のとおりです。

64 ビットサーバー

64 ビットサーバー (Solaris SPARC および AMD64 プラットフォームでのみ使用可能) は、32 ビットバージョンに比べて拡張性があります。システムに 4G バイトを超える RAM が実装されている場合は、64 ビットサーバーを使用できます。32 ビットサーバーと比較した場合の 64 ビットサーバーのいくつかの利点を次に示します。

- 静的コンテンツのためのファイルキャッシュの容量が大きい
- 64 ビット JVM のために同時サーブレットセッションの数が多い

プロセッサ

Solaris と Windows の場合、Web Server は複数の CPU を透過的に活用します。一般に、複数の CPU の有効性は、オペレーティングシステムや作業負荷によって異なります。動的コンテンツのパフォーマンスは、システムにプロセッサが追加されるに従って向上します。静的コンテンツは大部分が IO 処理であり、主記憶が増えるとコンテンツのキャッシュが増えることになる (サーバーがメモリーを活用するようにチューニングされていることが前提) ため、CPU ではなく IO アクティビティーにより多くの時間が費やされます。

メモリー

ベースラインとして、Web Server には 64M バイトの RAM が必要です。複数の CPU の場合は、CPU ごとに少なくとも 64M バイトが必要です。たとえば、CPU が 4 個の場合に最適なパフォーマンスを得るには、少なくとも 256M バイトの RAM をインストールするようにしてください。また、並行ユーザーのピーク数が多い場合も、追加のスレッドが余分な RAM を使用できるようにしてください。並行ユーザー数が 50 を超えたら、ピーク並行ユーザーごとに余分に 512K バイトを追加してください。

ドライブ領域

OS、ドキュメントツリー、およびログファイルに十分なドライブ領域を確保してください。ほとんどの場合は、全体で 2G バイトあれば十分です。

OS、スワップ/ページングファイル、Web Server ログ、およびドキュメントツリーを、それぞれ個別のハードドライブに配置します。ログドライブがログファイルでいっぱいになっても、OS には影響しません。また、たとえば、OS ページングファイルによってドライブアクティビティーが発生しているかどうかを検出することもできます。

OS ベンダーから、どの程度のスワップまたはページング領域を割り当てるべきかについての、特定の推奨事項が得られる可能性があります。テストによると、Web Server は、スワップ空間が RAM に等しい容量に加えて、ドキュメントツリーを十分にマップできるだけ存在する場合に最適なパフォーマンスを発揮します。

ネットワーキング

インターネットサイトの場合は、サーバーで処理する必要のある並行ユーザーのピーク数を特定し、そのユーザー数にサイトの平均要求サイズを掛けます。平均的な要求には、複数のドキュメントが含まれる可能性があります。はっきりしない場合は、ホームページと、それに関連付けられたすべてのサブフレームおよびグラフィックスを使用してみてください。

次に、ピーク使用時に、平均的なユーザーがドキュメントを我慢して待つことができる時間を特定します。その秒数で割ります。それが、サーバーに必要な WAN 帯域幅です。

たとえば、ピーク時に、平均ドキュメントサイズが 24K バイトで、各ドキュメントを平均 5 秒で転送している 50 ユーザーをサポートするには、240K バイト/秒 (1920 Kbps) が必要です。したがって、サイトには 2 つの T1 回線 (それぞれ 1544 Kbps) が必要です。また、これにより、成長のためのある程度のオーバーヘッドも可能になります。

サーバーのネットワークインタフェースカードでは、接続されている WAN を超える容量をサポートするようにしてください。たとえば、T1 回線 3 つまでは、10BaseT インタフェースで間に合います。T3 回線 (45 Mbps) 1 つまでは、100BaseT を使用できます。ただし、50 Mbps を超える WAN 帯域幅の場合は、複数の 100BaseT インタフェースの設定を考慮するか、またはギガビット Ethernet テクノロジを検討してください。

イントラネットサイトの場合は、ネットワークがボトルネックになる可能性は低くなります。ただし、これを確認するには、前述した計算と同じものを使用できます。

◆ ◆ ◆ 第 6 章

スケーラビリティー調査

この章では、スケーラビリティー調査の結果について説明します。これらの調査を参照することにより、サーバーのパフォーマンスや、Web Server の長所を最大限に活用するためのシステムの設定方法についてのサンプルを確認できます。

この章の内容は、次のとおりです。

調査の目標

調査での各テストの目標は、Sun Java System Web Server 7 のスケーラビリティーを示すことにありました。これらのテストはまた、異なる種類のコンテンツに対する構成やチューニングの要件を特定するためにも役立ちました。

調査は、次のコンテンツを使用して実行されました。

- 100% 静的
- 100% C CGI
- 100% Perl CGI
- 100% NSAPI
- 100% Java サーブレット
- 100% PHP/FastCGI
- 大量のインベントリを含む電子商取引 Web アプリケーション

調査の結論

チューニングした場合、動的および静的コンテンツに対するパフォーマンスで、Sun Java System Web Server 7.0 はほぼ直線的なスケーラビリティーを示しました。

ハードウェア

調査 (電子商取引の調査を除く) は、次のハードウェアを使用して実行されました。電子商取引の調査のためのハードウェア情報については、132 ページの「電子商取引テストのためのハードウェア」を参照してください。

静的コンテンツのための Web Server のシステム構成:

- Sun Microsystems Sun Fire T2000 (120 MHz、8 コア) (このテストには 6 コアのみを使用)
- 16256M バイトのメモリー
- Solaris 10 オペレーティングシステム
- Sun StoreEdge 3510 × 3

Web Server のシステム構成:

- Sun Microsystems Sun Fire T2000 (1000 MHz、6 コア)
- 16376M バイトのメモリー
- Solaris 10 オペレーティングシステム

ドライバのシステム構成:

- Sun Microsystems Sun Fire™ X4100 × 3
- Sun Microsystems Sun Fire V490 × 4 (2 × 1050 MHzUS-IV)
- Sun Fire T1000 × 3
- Sun Fire 880 (990 MHz US-III+)
- 8192M バイトのメモリー
- Solaris 10 オペレーティングシステム

ネットワーク構成:

Web Server とドライバマシンを複数のギガビット Ethernet リンクで接続しました。

ソフトウェア

これらのテストのための読み込みドライバは、Faban ドライバと呼ばれる、社内で開発された Java アプリケーションフレームワークでした。

構成とチューニング

次のチューニング設定は、この調査でのすべてのテストに共通しています。また、個別の調査にも、構成とチューニングの追加情報が含まれている可能性があります。

`/etc/system` のチューニング:

```
set rlim_fd_max=500000
set rlim_fd_cur=500000


set sq_max_size=0
set consistent_coloring=2
set autoup=60
set ip:ip_squeue_bind=0
set ip:ip_soft_rings_cnt=0
set ip:ip_squeue_fanout=1
set ip:ip_squeue_enter=3
set ip:ip_squeue_worker_wait=0

set segmap_percent=6
set bufhwm=32768
set maxphys=1048576
set maxpgio=128
set ufs:smallfile=6000000

*For ipge driver
set ipge:ipge_tx_ring_size=2048
set ipge:ipge_tx_syncq=1
set ipge:ipge_srv_fifo_depth=16000
set ipge:ipge_reclaim_pending=32
set ipge:ipge_bcopy_thresh=512
set ipge:ipge_dvma_thresh=1
set pcie:pcie_aer_ce_mask=0x1

*For e1000g driver
set pcie:pcie_aer_ce_mask = 0x1
```

TCP/IP のチューニング:

```
ndd -set /dev/tcp tcp_conn_req_max_q 102400
ndd -set /dev/tcp tcp_conn_req_max_q0 102400
ndd -set /dev/tcp tcp_max_buf 4194304
ndd -set /dev/tcp tcp_cwnd_max 2097152
ndd -set /dev/tcp tcp_recv_hiwat 400000
ndd -set /dev/tcp tcp_xmit_hiwat 400000
```

ネットワーク構成

テストでは複数のネットワークインタフェースを使用したため、すべてのネットワークインタフェースが同じコアに行かないようにすることが重要でした。次のスクリプトを使用して、ネットワーク割り込みをコアの 1 つのストランドでは有効に、残りの 3 つのストランドでは無効にしました。

```
allpsr=‘/usr/sbin/psrinfo | grep -v off-line | awk ’{ print $1 }’‘
  set $allpsr
  numpsr=$#
  while [ $numpsr -gt 0 ];
  do
      shift
      numpsr=‘expr $numpsr - 1‘
      tmp=1
      while [ $tmp -ne 4 ];
      do
          /usr/sbin/psradm -i $1
          shift
          numpsr=‘expr $numpsr - 1‘
          tmp=‘expr $tmp + 1‘
      done
  done
```

次の例は、スクリプトを実行する前の `psrinfo` 出力を示しています。

```
# psrinfo | more
0       on-line   since 12/06/2006 14:28:34
1       on-line   since 12/06/2006 14:28:35
2       on-line   since 12/06/2006 14:28:35
3       on-line   since 12/06/2006 14:28:35
4       on-line   since 12/06/2006 14:28:35
5       on-line   since 12/06/2006 14:28:35
.................
```

次の例は、スクリプトを実行したあとの `psrinfo` 出力を示しています。

```
0       on-line   since 12/06/2006 14:28:34
1       no-intr   since 12/07/2006 09:17:04
2       no-intr   since 12/07/2006 09:17:04
```

```
3       no-intr   since 12/07/2006 09:17:04
4       on-line   since 12/06/2006 14:28:35
5       no-intr   since 12/07/2006 09:17:04
          ..................
```

Web Server のチューニング

次の表は、Web Server のために使用したチューニング設定を示しています。

表 6–1　Web Server のチューニング設定

構成要素	デフォルト	チューニング値
アクセスログ	`enabled=true`	`enabled=false`

スレッドプール	`min-threads=16`
	`max-threads=128`
	`stack-size=131072`
	`queue-size=1024`
		`min-threads=128`
		`max-threads=200`
		`stack-size=262144`
		`queue-size=15000`

HTTP リスナー	ポート `80` 上のセキュリティー保護されていないリスナー `listen-queue-size=128`	ポート `80` 上のセキュリティー保護されていないリスナー ポート `443` 上のセキュリティー保護されたリスナー `listen-queue-size=15000`

キープアライブ	`enabled=true`
	`threads=1`
	`max-connections=200` `timeout=30` `sec`
		`enabled=true`
		`threads=2`
		`max-connections=15000` `timeout=180` `sec`

`default-web.xml`	JSP コンパイルが有効	JSP コンパイルが無効

次の表は、SSL テストのために使用した SSL セッションキャッシュのチューニング設定を示しています。

表 6–2 SSL セッションキャッシュのチューニング設定

構成要素	デフォルト
SSL セッションキャッシュ	enabled=true max-entries=10000 max-ssl2-session-age=100 max-ssl3-tls-session-age=86400

パフォーマンステストと結果

ここでは、次のテストのためのテスト固有の構成、チューニング、および結果を示します。

- 118 ページの「静的コンテンツテスト」
- 120 ページの「動的コンテンツテスト: サーブレット」
- 121 ページの「動的コンテンツテスト: C CGI」
- 123 ページの「動的コンテンツテスト: Perl CGI」
- 124 ページの「動的コンテンツテスト: NSAPI」
- 125 ページの「PHP のスケーラビリティーテスト」
- 128 ページの「SSL パフォーマンステスト: 静的コンテンツ」
- 129 ページの「SSL パフォーマンステスト: Perl CGI」
- 130 ページの「SSL パフォーマンステスト: C CGI」
- 131 ページの「SSL パフォーマンステスト: NSAPI」
- 132 ページの「電子商取引 Web アプリケーションテスト」

パフォーマンスの特性を示すために、次の測定基準を使用しました。

- 1 秒あたりの操作数 (ops/秒) = 1 秒あたりの正常なトランザクション数
- 1 つのトランザクションに対する応答時間 (ラウンドトリップ時間) (ミリ秒単位)

パフォーマンスとスケーラビリティーの図は、システムで有効になっているコアの数に対するスループット (ops/秒) を示しています。

静的コンテンツテスト

このテストは、それぞれ 1K ～ 1000K バイトのサイズの 36 のファイルを含む、10,000 のディレクトリのプールからランダムに選択されたファイルの静的なダウンロードを使用して実行されました。静的コンテンツテストの目標は、コアを飽和させ、それぞれのスループットと応答時間を調べることでした。

このテストでは、次の構成を使用しました。

- ストライプ化ディスクアレイ (Sun StorEdge 3510) 上に静的ファイルを作成しました。
- 複数のネットワークインタフェースを設定しました。
- Web Server を 64 ビットに設定しました。
- 次の表で説明されているチューニング設定を使用してファイルキャッシュを有効にしました。

表 6–3 ファイルキャッシュの構成

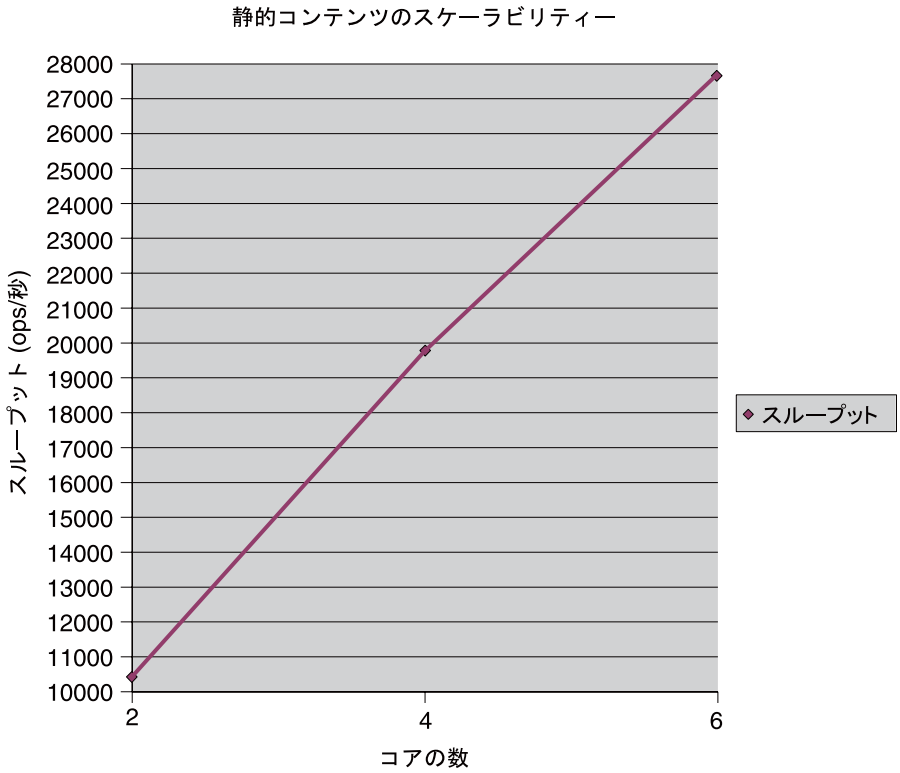
デフォルト	チューニング値
enabled=true	enabled=true
max-age=30 sec	max-age=3600
max-entries=1024	max-entries=1048576
sendfile=false	sendfile=true
max-heap-file-size=524288	max-heap-file-size=1200000
max-heap-space=10485760	max-heap-space=8000000000
max-mmap-file-size=0	max-mmap-file-size=1048576
max-mmap-space=0	max-mmap-space= l
	max-open-files=1048576

次の表は、静的コンテンツのスケーラビリティーの結果を示しています。

表 6–4 静的コンテンツのスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	10365	184
4	19729	199
6	27649	201

次の図は、静的コンテンツのスケーラビリティーの結果を示すグラフ表示です。



動的コンテンツテスト:サーブレット

このテストは、サーブレットを使用して実行されました。このテストは、サーブレットの初期化引数、環境、要求ヘッダー、接続とクライアントの情報、URL 情報、およびリモートユーザー情報を出力します。サーバーには、JVM のチューニング設定を適用しました。目標は、サーバー上のコアを飽和させ、それぞれのスループットと応答時間を調べることでした。

次の表は、このテストで使用された JVM のチューニング設定を示しています。

表 6–5　JVM のチューニング設定

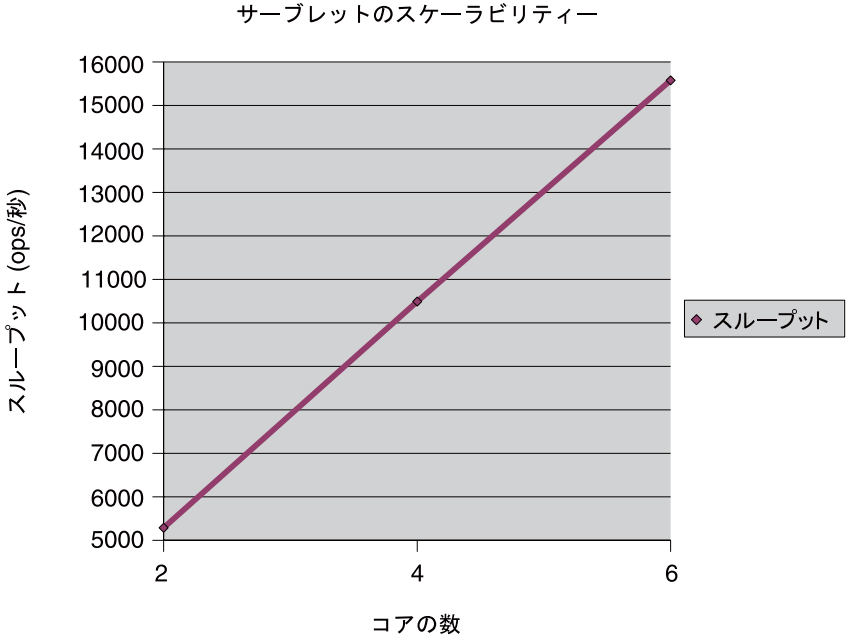
デフォルト	チューニング値
-Xmx128m -Xms256m	-server -Xrs -Xmx2048m -Xms2048m -Xmn2024m -XX:+AggressiveHeap -XX:LargePageSizeInBytes=256m -XX:+UseParallelOldGC -XX:+UseParallelGC -XX:ParallelGCThreads=<number of cores> -XX:+DisableExplicitGC

次の表は、動的コンテンツサーブレットテストの結果を示しています。

表 6–6　動的コンテンツテスト:サーブレットのスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	5287	19
4	10492	19
6	15579	19

次の図は、サーブレットのスケーラビリティーの結果を示すグラフ表示です。



動的コンテンツテスト:C CGI

このテストは、`printenv` という名前の C 実行可能ファイルにアクセスすることにより実行されました。この実行可能ファイルは、環境変数情報を出力します。サーバーには、CGI のチューニング設定を適用しました。目標は、サーバー上のコアを飽和させ、それぞれのスループットと応答時間を調べることでした。

次の表は、このテストで使用された CGI のチューニング設定を示しています。

表 6–7　CGI のチューニング設定

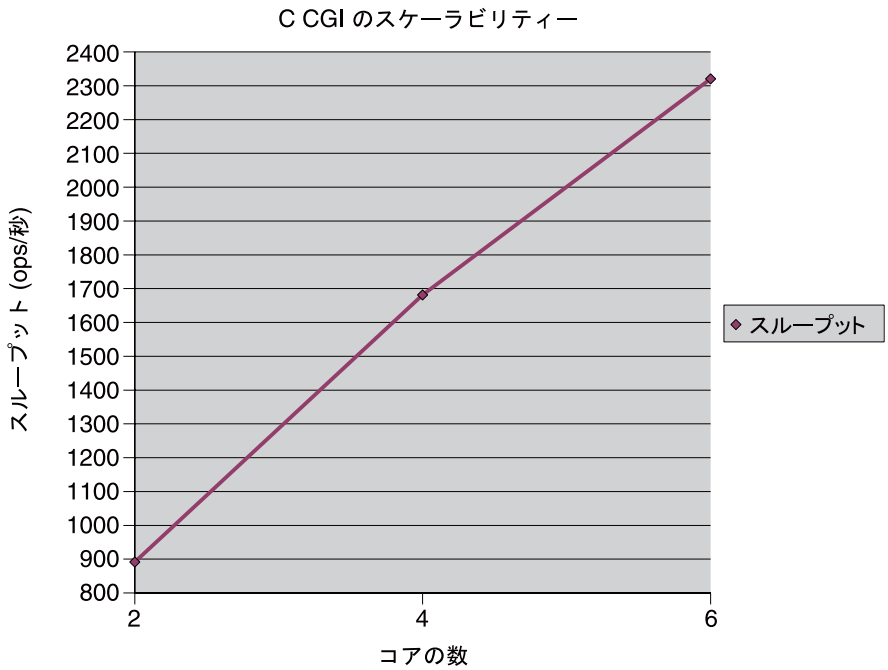
デフォルト	チューニング値
`idle-timeout=300`	`idle-timeout=300`
`cgistub-idle-timeout=30`	`cgistub-idle-timeout=1000`
`min-cgistubs=0`	`min-cgistubs=100`
`max-cgistubs=16`	`max-cgistubs=100`

次の表は、C CGI に対する動的コンテンツテストの結果を示しています。

表 6–8　動的コンテンツテスト: C CGI のスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	892	112
4	1681	119
6	2320	129

次の図は、C CGI のスケーラビリティーの結果を示すグラフ表示です。



動的コンテンツテスト: Perl CGI

このテストは、CGI 環境を出力する `printenv.pl` という名前の Perl スクリプトを使用して実行されました。サーバーには、CGI のチューニング設定を適用しました。目標は、サーバー上のコアを飽和させ、それぞれのスループットと応答時間を調べることでした。

次の表は、Perl CGI に対する動的コンテンツテストで使用された CGI のチューニング設定を示しています。

表 6–9 CGI のチューニング設定

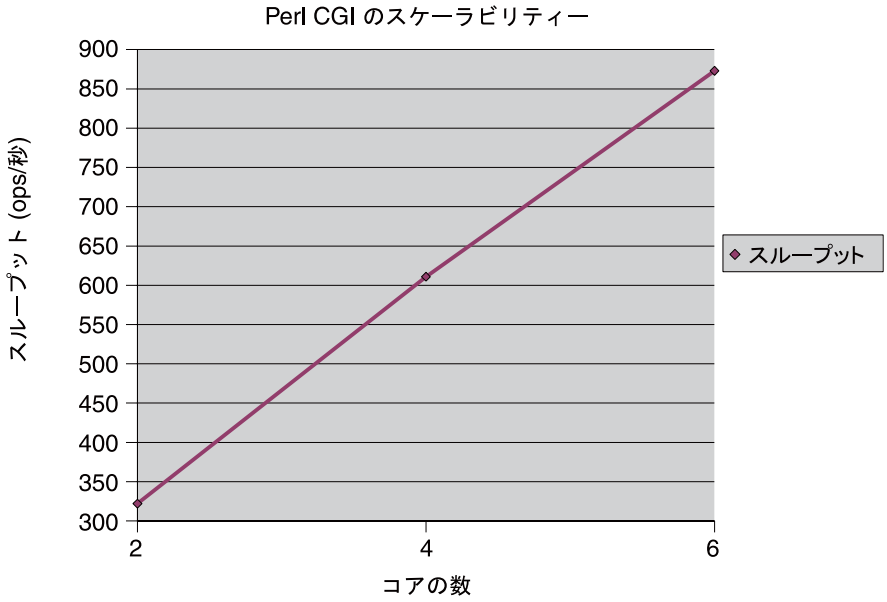
デフォルト	チューニング値
`idle-timeout=300`	`idle-timeout=300`
`cgistub-idle-timeout=30`	`cgistub-idle-timeout=1000`
`min-cgistubs=0`	`min-cgistubs=100`
`max-cgistubs=16`	`max-cgistubs=100`

次の表は、Perl CGI に対する動的コンテンツテストの結果を示しています。

表 6–10 動的コンテンツテスト: Perl CGI のスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	322	310
4	611	327
6	873	343

次の図は、Perl CGI のスケーラビリティーの結果を示すグラフ表示です。



動的コンテンツテスト: NSAPI

このテストで使用された NSAPI モジュールは `printenv2.so` です。このモジュールは、NSAPI 環境変数を、応答全体を 2K バイトにするためのテキストとともに出力します。目標は、サーバー上のコアを飽和させ、それぞれのスループットと応答時間を調べることでした。

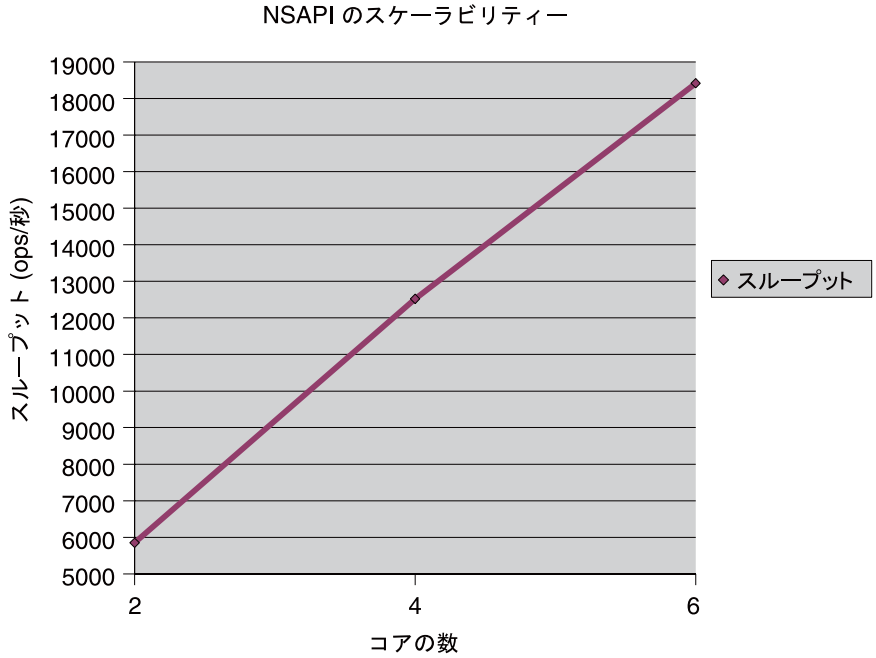
このテストのための唯一のチューニングは、未使用のパスチェックを削除することによる `obj.conf` 内のパスチェックの最適化でした。

次の表は、NSAPI に対する動的コンテンツテストの結果を示しています。

表 6–11　動的コンテンツテスト: NSAPI のスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	26264	14
4	12520	15
6	18417	16

次の図は、NSAPI のスケーラビリティーの結果を示すグラフ表示です。



PHP のスケーラビリティーテスト

PHP は、Web ベースの動的コンテンツの作成に特化した、広く使用されているスクリプト言語です。その単純性、アクセシビリティー、使用可能なモジュールの多さ、容易に使用可能なアプリケーションの多さなどで、インターネット上での使用がもっとも急速に拡大しているスクリプト言語です。

Web Server のスケーラビリティーと PHP エンジンの汎用性が組み合わされて、動的コンテンツのための、パフォーマンスの高い柔軟な Web 配備プラットフォームが提供されます。これらのテストでは、PHP バージョン 5.1.6 を使用しました。

テストは、次の 2 つのモードで実行されました。

- Sun Java System Web Server 7.0 で使用可能な FastCGI プラグインを使用して呼び出されるアウトプロセスの `fastcgi-php` アプリケーション (`http://www.zend.com/sun/` からダウンロード可能になる予定)。
- インプロセスの PHP NSAPI プラグイン。

テストでは、`phpinfo()` クエリーを実行しました。目標は、サーバー上のコアを飽和させ、それぞれのスループットと応答時間を調べることでした。

FastCGI を使用した PHP のスケーラビリティー

次の表は、FastCGI プラグインテストのために使用した Web Server のチューニング設定を示しています。

表 6–12 FastCGI プラグインテストのためのチューニング設定

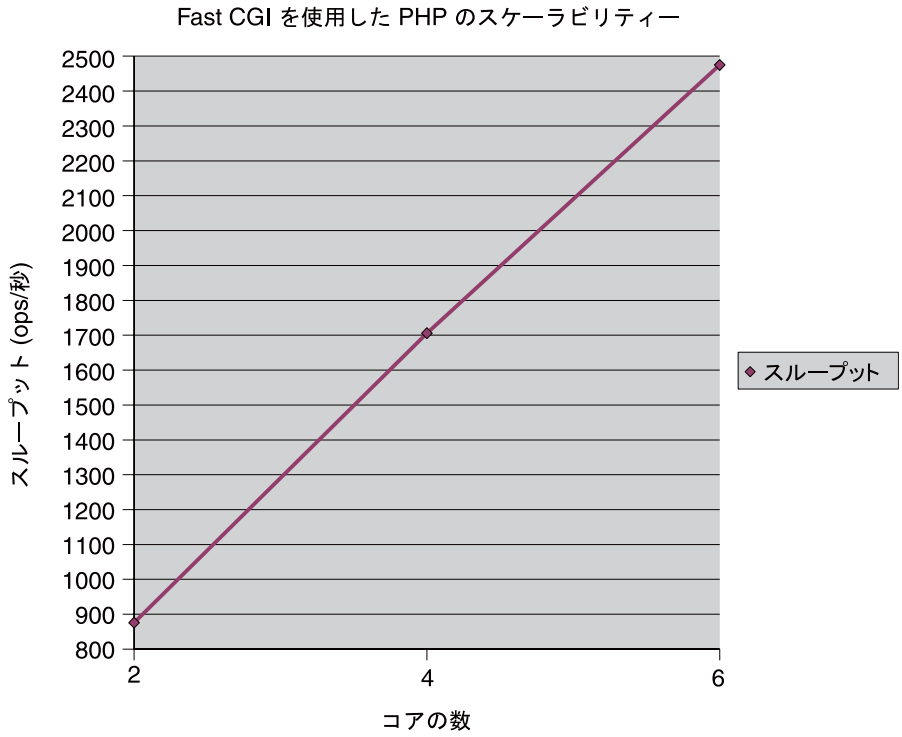
構成	チューニング
magnus.conf	Init fn="load-modules" shlib="*path_to_web_server_plugin_dir*/fastcgi/libfastcgi.so" funcs="responder_fastcgi" shlib_flags="(global|now)"
obj.conf	NameTrans fn="assign-name" from="/fcgi/*" name="fcgi.config" <Object name="fcgi.config"> Service type="magnus-internal/ws-php" fn="responder-fastcgi" app-path="*path_to_php*" bind-path="localhost:9000" app-env="PHP_FCGI_CHILDREN=128" app-env="PHP_FCGI_MAX_REQUESTS=20000" app-env="LD_LIBRARY_PATH=*path_to_php_lib*" listen-queue=8192 req-retry=2 reuse-connection=1 connection-timeout=120 resp-timeout=60 restart-interval=0 </Object>
mime.types	type=magnus-internal/ws-php exts=php,php3,php4

次の表は、FastCGI テストを使用した PHP の結果を示しています。

表 6–13 FastCGI を使用した PHP のスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	876	114
4	1706	117
6	2475	121

次の図は、FastCGI を使用した PHP のスケーラビリティーを示すグラフ表示です。



NSAPI を使用した PHP のスケーラビリティー

次の表は、NSAPI テストを使用した PHP のための Web Server のチューニング設定を示しています。

表 6–14　PHP のための NSAPI プラグイン設定

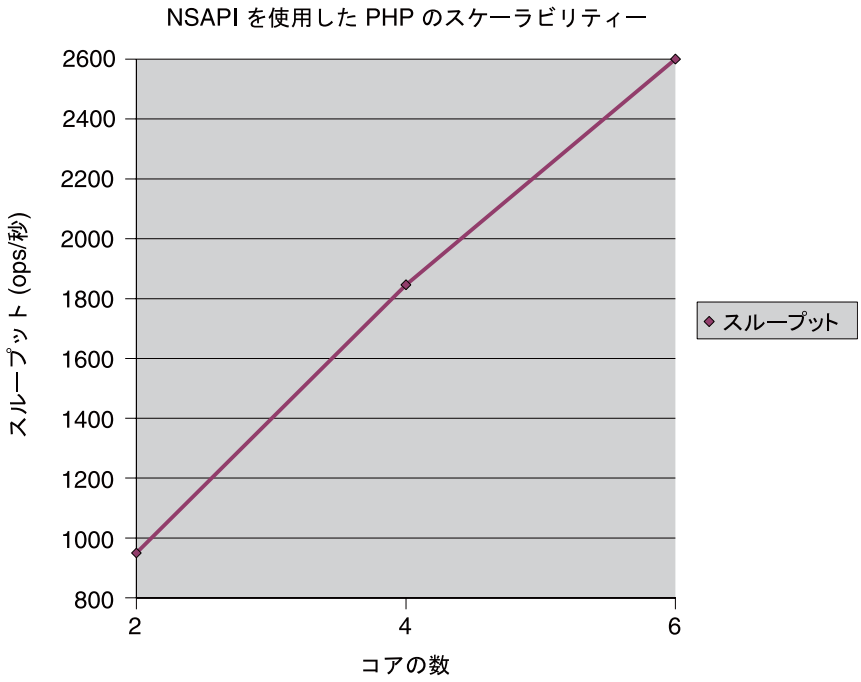
magnus.conf	Init fn="load-modules" shlib="libphp5.so" funcs="php5_init,php5_close,php5_execute" Init fn="php5_init" errorString="PHP Totally Blew Up!"
obj.conf	NameTrans fn="pfx2dir" from="/php-nsapi" dir="*path_to_php_script_dir*" name="php-nsapi" <Object name="php-nsapi"> ObjectType fn="force-type" type="magnus-internal/x-httpd-php" Service fn=php5_execute </Object>
mime.types	type=magnus-internal/ws-php exts=php,php3,php4

次の表は、NSAPI テストを使用した PHP の結果を示しています。

表 6–15　NSAPI を使用した PHP のスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	950	105
4	1846	108
6	2600	115

次の図は、NSAPI を使用した PHP のスケーラビリティーを示すグラフ表示です。



SSL パフォーマンステスト: 静的コンテンツ

このテストは、それぞれ 1K ～ 1000K バイトのサイズの 36 のファイルを含む、10,000 のディレクトリのプールからランダムに選択されたファイルの静的なダウンロードを使用して実行されました。SSL 静的コンテンツテストの目標は、コアを飽和させ、それぞれのスループットと応答時間を調べることでした。このテストには T2000 の 4 コアのみを使用しました。

このテストでは、次の構成を使用しました。

- ストライプ化ディスクアレイ (Sun StorEdge 3510) 上に静的ファイルを作成しました。
- 複数のネットワークインタフェースを設定しました。

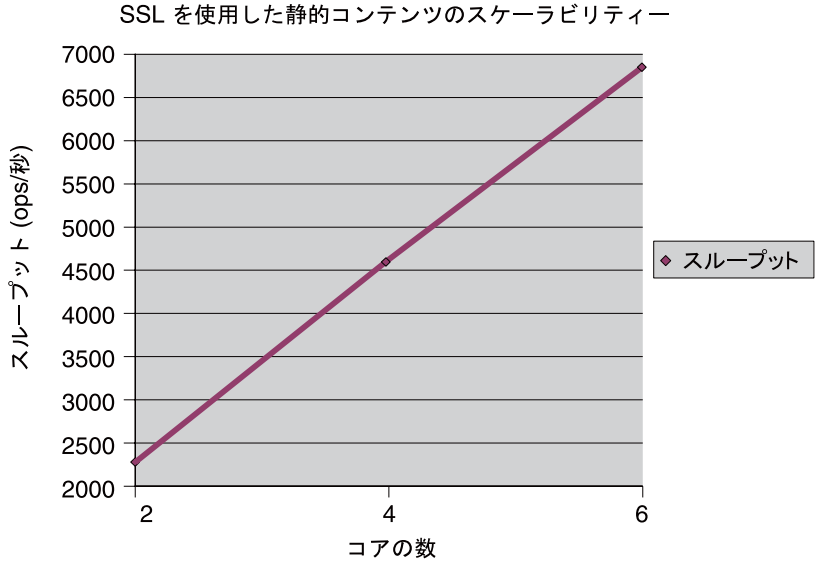
- 表 6–3 にある設定を使用してファイルキャッシュを有効にし、チューニングしました。
- 表 6–2 にある設定を使用して SSL セッションキャッシュをチューニングしました。
- Web Server を 64 ビットに設定しました。

次の表は、SSL 静的コンテンツテストの結果を示しています。

表 6–16　SSL パフォーマンステスト: 静的コンテンツのスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	2284	379
4	4538	387
6	6799	387

次の図は、SSL を使用した静的コンテンツのスケーラビリティーを示すグラフ表示です。



SSL パフォーマンステスト: Perl CGI

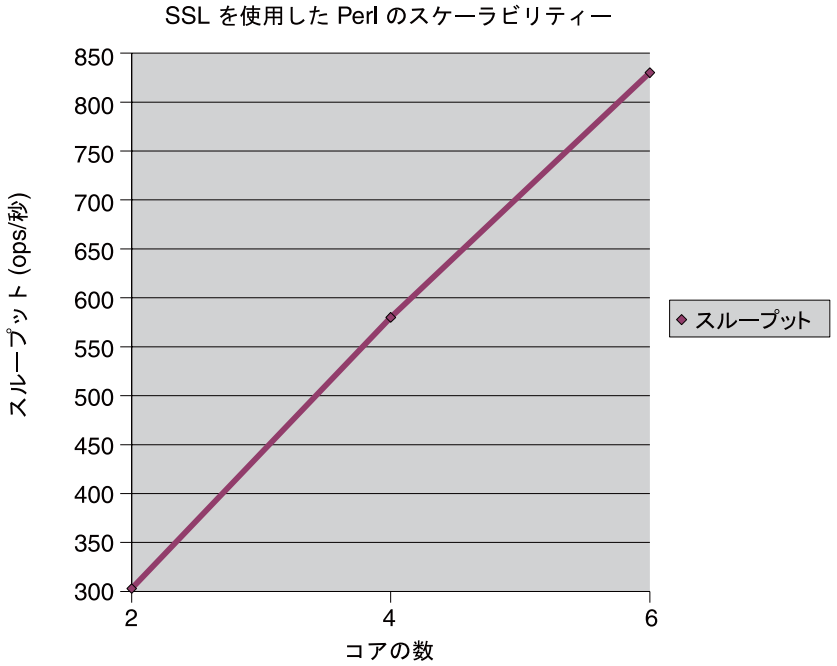
このテストは、SSL モードで CGI 環境を出力する `printenv.pl` という名前の Perl スクリプトを使用して実行されました。SSL セッションキャッシュを有効にした状態で、SSL モードでテストを実行しました。目標は、サーバー上のコアを飽和させ、それぞれのスループットと応答時間を調べることでした。

次の表は、SSL Perl CGI テストの結果を示しています。

表 6–17　SSL パフォーマンステスト: Perl CGI のスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	303	329
4	580	344
6	830	361

次の図は、SSL を使用した Perl のスケーラビリティーを示すグラフ表示です。



SSL パフォーマンステスト: C CGI

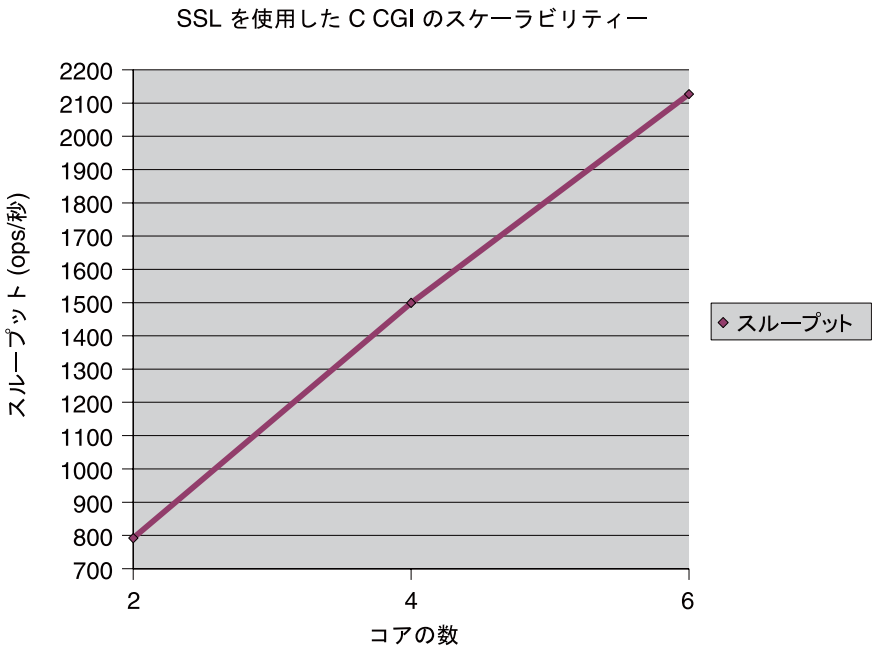
このテストは、SSL モードで `printenv` という名前の C 実行可能ファイルにアクセスすることにより実行されました。この実行可能ファイルは、環境変数情報を出力します。SSL セッションキャッシュを有効にした状態で、SSL モードでテストを実行しました。目標は、サーバー上のコアを飽和させ、それぞれのスループットと応答時間を調べることでした。

次の表は、SSL CGI テストの結果を示しています。

表 6–18　SSL パフォーマンステスト: C CGI のスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	792	126
4	1499	133
6	2127	141

次の図は、SSL を使用した C CGI のスケーラビリティーを示すグラフ表示です。



SSL パフォーマンステスト: NSAPI

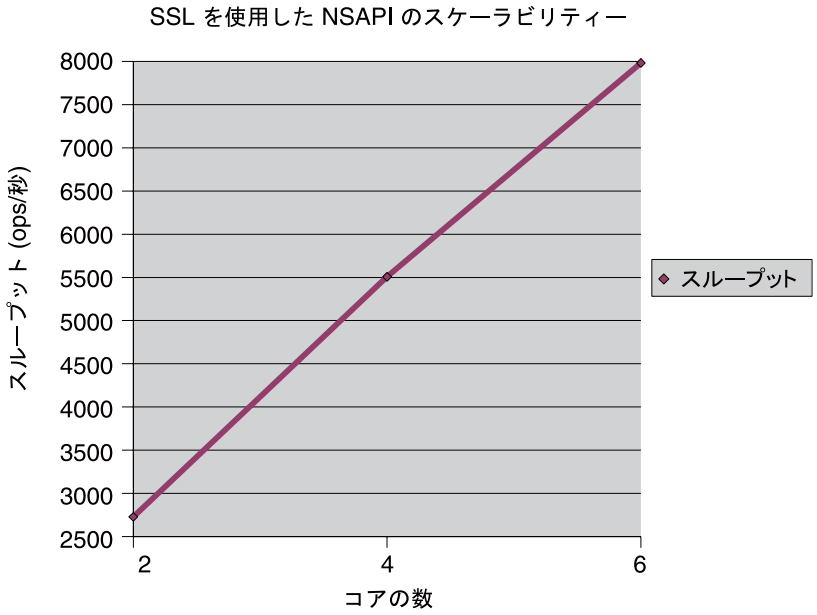
このテストで使用された NSAPI モジュールは `printenv2.so` です。このモジュールは、NSAPI 環境変数を、応答全体を 2K バイトにするためのテキストとともに出力します。SSL セッションキャッシュを有効にした状態で、SSL モードでテストを実行しました。目標は、サーバー上のコアを飽和させ、それぞれのスループットと応答時間を調べることでした。

次の表は、SSL NSAPI テストの結果を示しています。

表 6–19　SSL パフォーマンステスト: NSAPI のスケーラビリティー

コアの数	平均スループット (ops/秒)	平均応答時間 (ミリ秒)
2	2729	29
4	5508	30
6	7982	32

次の図は、SSL を使用した NSAPI のスケーラビリティーを示すグラフ表示です。



電子商取引 Web アプリケーションテスト

電子商取引アプリケーションは、データベースを利用してオンラインショッピングのシミュレーションを行う、より複雑なアプリケーションです。

電子商取引テストのためのハードウェア

電子商取引の調査は、次のハードウェアを使用して実行されました。

Web Server のシステム構成:

- Sun Microsystems Sun Fire 880 (900MHz US-III+)。このテストには 4 個の CPU のみを使用しました。
- 16384M バイトのメモリー
- Solaris 10 オペレーティングシステム

データベースのシステム構成:

- Sun Microsystems Sun Fire 880 (900MHz US-III+)
- 16384M バイトのメモリー
- Solaris 10 オペレーティングシステム
- Oracle 10.1.0.2.0

ドライバのシステム構成:

- Sun Microsystems Sun Fire 880 (900MHz US-III+)
- Solaris 10 オペレーティングシステム

ネットワーク構成:

Web Server、データベース、およびドライバマシンをギガビット Ethernet リンクで接続しました。

電子商取引テストのための構成とチューニング

電子商取引テストは、次のチューニング設定を使用して実行されました。

JDBC のチューニング:

```
<jdbc-resource>
    <jndi-name>jdbc/jwebapp</jndi-name>
    <datasource-class>oracle.jdbc.pool.OracleDataSource</datasource-class>
    <max-connections>200</max-connections>
    <idle-timeout>0</idle-timeout>
    <wait-timeout>5</wait-timeout>
    <connection-validation>auto-commit</connection-validation>
    <property>
      <name>username</name>
      <value>  db_user  </value>
    </property>
    <property>
      <name>password</name>
      <value> db_password    </value>
    </property>
    <property>
      <name>url</name>
      <value>jdbc:oracle:thin:@db_host_name:1521:oracle_sid</value>
    </property>
 <property>
      <name>ImplicitCachingEnabled</name>
      <value>true</value>
    </property>
    <property>
      <name>MaxStatements</name>
      <value>200</value>
```

```
    </property>
  </jdbc-resource
```

JVM のチューニング:

```
-server -Xmx1500m -Xms1500m -Xss128k -XX:+DisableExplicitGC
```

電子商取引アプリケーションの説明

このテストでは、大量のインベントリからアイテムを販売する電子商取引 Web サイトをモデル化しました。その実装には、標準的な Web アプリケーションのモデル/表示/制御のデザインパターンを使用しています。ユーザーインタフェース (つまり、表示) は、1 つのマスター制御サーブレットとインタフェースする 16 種類の JSP ページで処理されます。このサーブレットは、モデルとして機能して 27 種類のクエリーを処理するデータベースへの JDBC 接続を維持しています。これらの JSP ページは JSP タグライブラリを広範囲に使用しており、ほぼ 2000 行のロジックで構成されます。

データベースのカーディナリティー

データベースには、1000 個の注文可能なアイテム (やはり 1000 のカーディナリティーを持つ 2 つの関連するテーブルがある)、72000 の顧客 (2 つの関連するテーブルがある)、および 190 万の注文 (2 つの関連するテーブルがある) が含まれています。標準の JDBC 接続が、準備済み文と次の標準の JDBC 設計原則を使用してデータベース接続を処理します。

作業負荷

ランダムに選択されたユーザーがオンラインショッピングを実行します。複合作業負荷で次の操作が使用されました。各操作は、操作の優先順位に従って実行されました。Home、AdminConfirm、AdminRequest、BestSellers、BuyConfirm、BuyRequest、CustomerRegistration、NewProducts、OrderDisplay、OrderInquiry、ProductDetail、SearchRequest、SearchResults、および ShoppingCart。

読み込みの起動には Faban ドライバを使用しました。思考時間を負の指数分布から選択しました。最小の思考時間は 7.5 秒、最大の思考時間は 75 秒でした。システムがサポートできる並行ユーザーの最大数を、次の合格条件に基づいて決定しました。

表 **6–20** パフォーマンステストの合格条件

トランザクション	90 パーセンタイルの応答時間 (秒)
HomeStart	3
AdminConfirm	20

表 6–20 パフォーマンステストの合格条件 (続き)

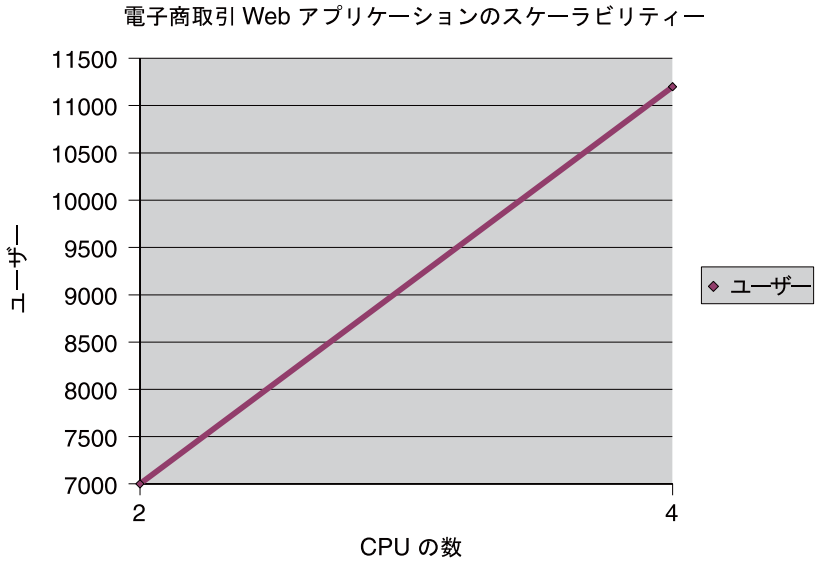
トランザクション	90 パーセンタイルの応答時間 (秒)
AdminRequest	3
BestSellers	5
BuyConfirm	5
BuyRequest	3
CustomerRegistration	3
Home	3
NewProducts	5
OrderDisplay	3
OrderInquiry	3
ProductDetail	3
SearchRequest	3
SearchResults	10
ShoppingCart	3

次の表は、電子商取引 Web アプリケーションテストの結果を示しています。

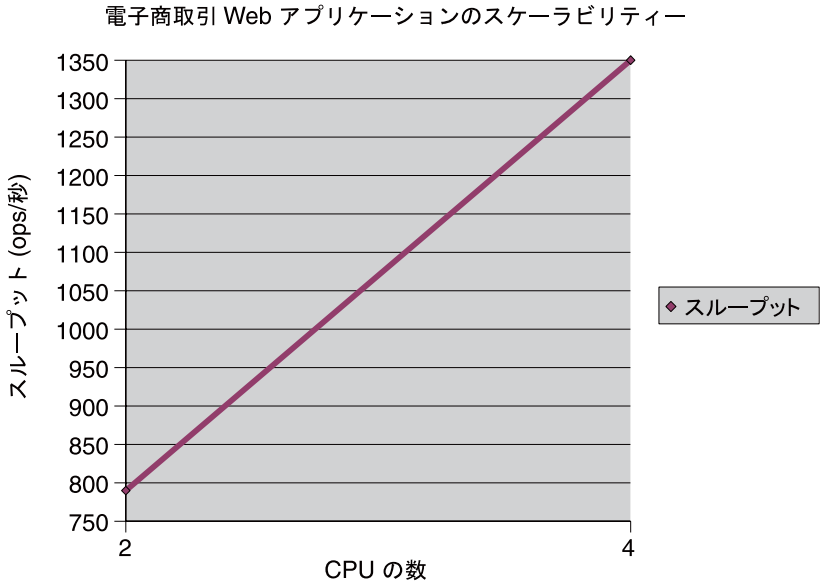
表 6–21 電子商取引 Web アプリケーションのスケーラビリティー

CPU の数	ユーザー	スループット (ops/秒)
2	7000	790
4	11200	1350

次の図は、電子商取引 Web アプリケーションのスケーラビリティーを示すグラフ表示です。



次の図は、電子商取引 Web アプリケーションのスケーラビリティーを示すグラフ表示です。



索引