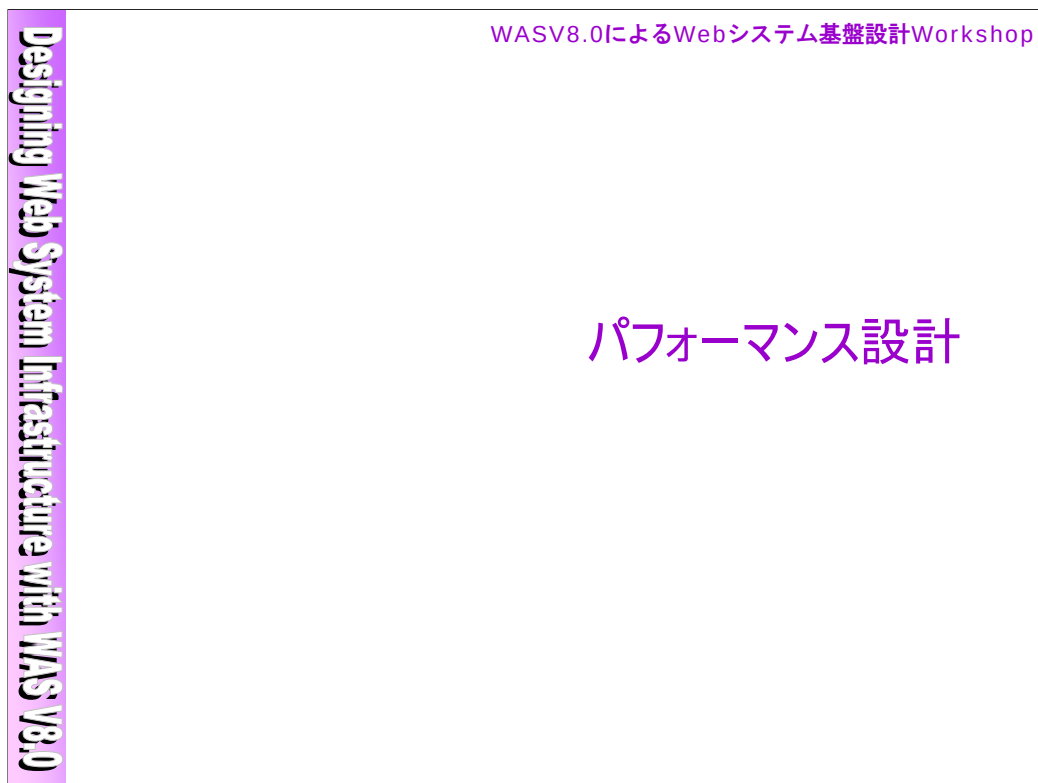




当セッションの目的

- ミッション・クリティカルなWebシステム環境におけるWAS V8.0のパフォーマンス設計に必要な知識の習得
 - ◆ WASが提供しているパフォーマンス・データ取得機能やチューニング項目を理解する。
 - ◆ パフォーマンス・テストに基づき、WASのチューニング・パラメータの設定基準、メリット・デメリットを理解する。

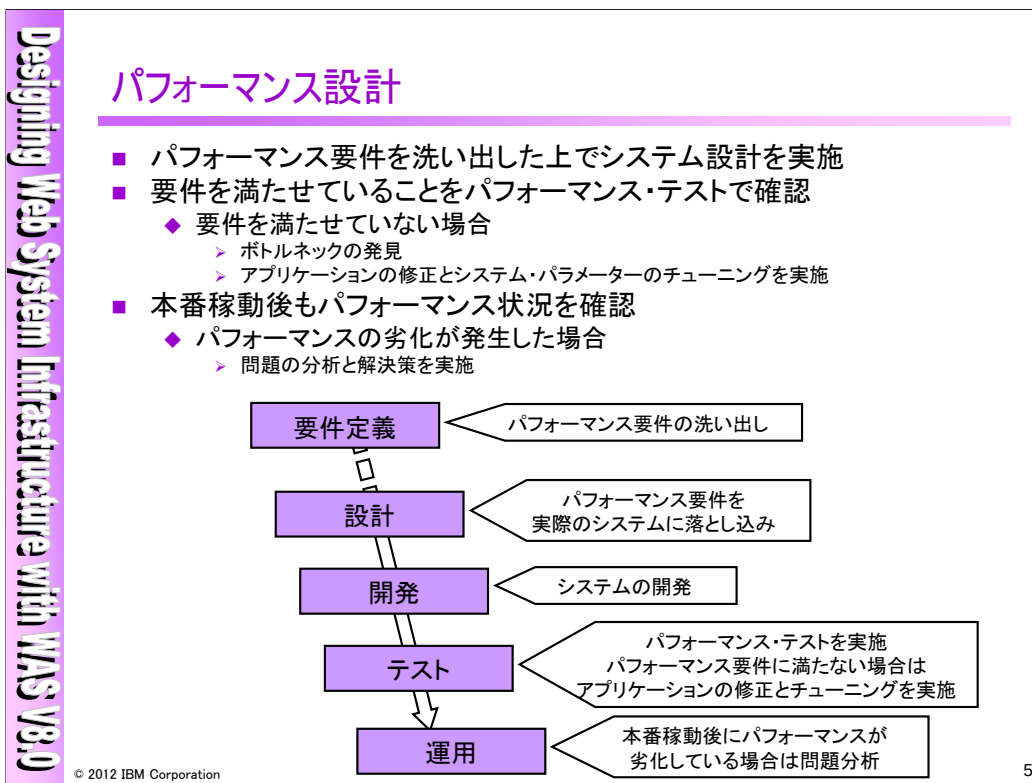
Agenda

1. パフォーマンス設計概説
 2. データの取得
 3. チューニング
 4. パフォーマンスを向上する追加コンポーネント
- まとめ・参考文献

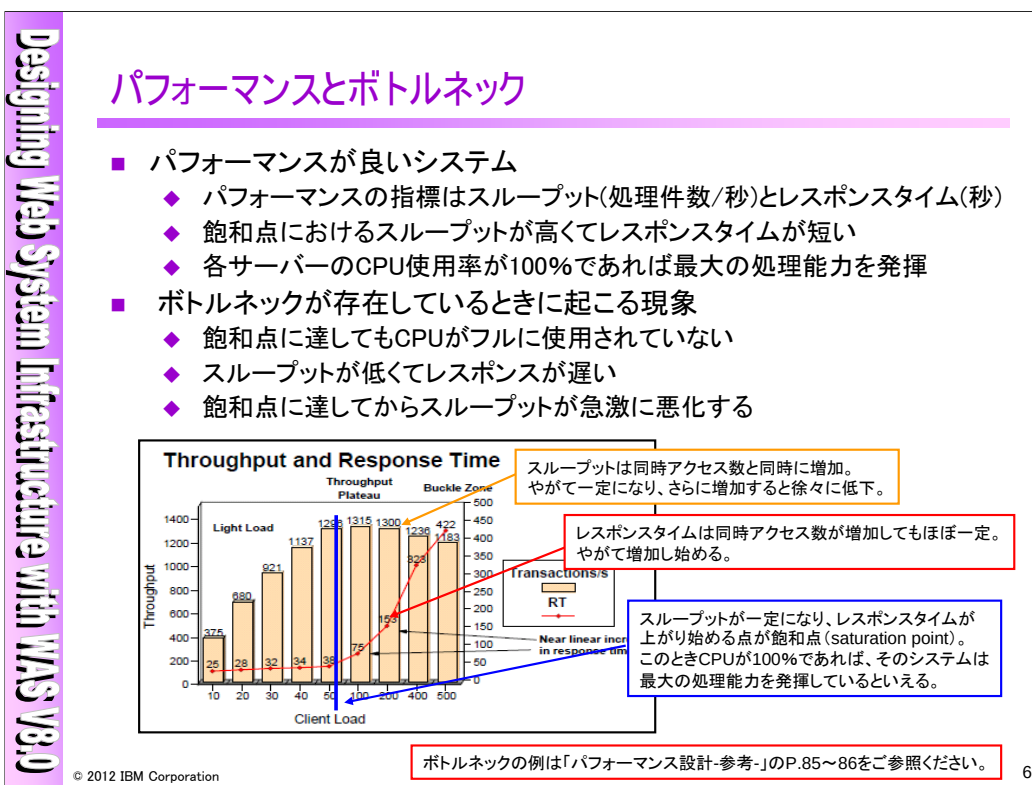
Designing Web System Infrastructure with WAS V8.0

1. パフォーマンス設計概説

© 2012 IBM Corporation 4



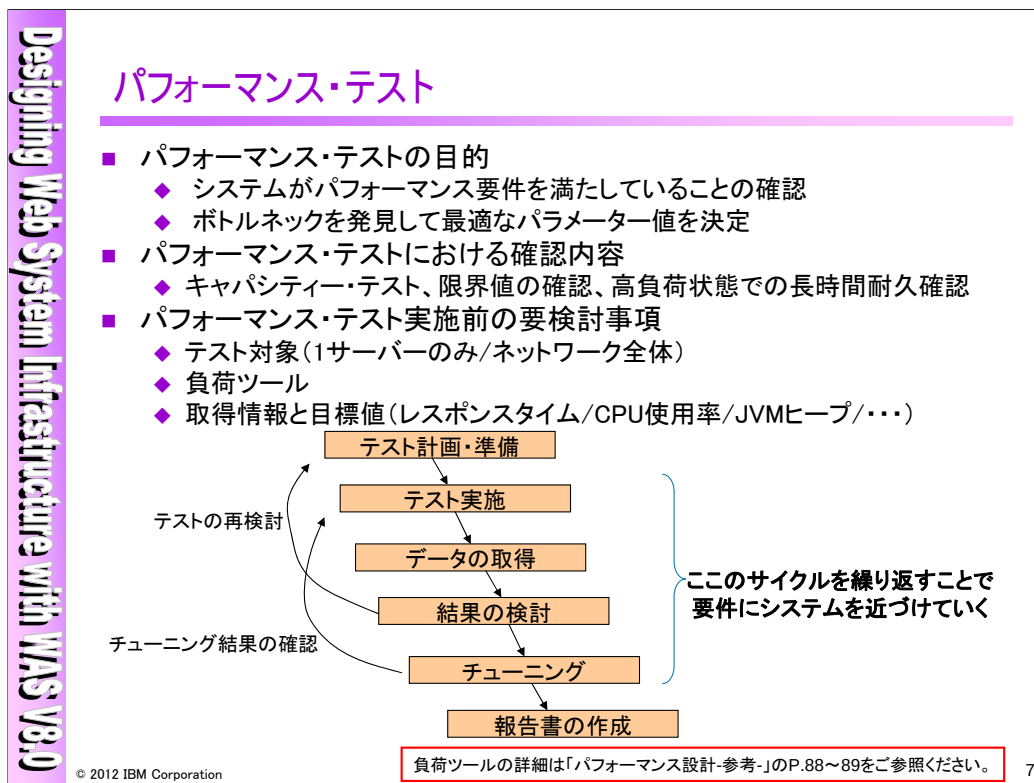
パフォーマンス設計は、まず、パフォーマンス要件を要件定義段階で洗い出すことから始まります。その要件に応じてシステム設計を実施し、実際にパフォーマンス要件を満たしていることを確認するためにパフォーマンス・テストを実施します。パフォーマンス・テストにて要件を満たしていない場合は、そのボトルネックを発見し、アプリケーションの修正とシステム・パラメーターのチューニングを繰り返し実施します。本番稼動後も、パフォーマンスの状況は常時あるいは定期的に確認し、パフォーマンスの劣化が発生した場合は、問題の分析と解決策を実施します。



一般的に、パフォーマンスの指標はスループット(処理件数/秒)とレスポンスタイム(秒)で表されます。システムに負荷をかけるにつれ、スループットは上昇、レスポンスタイムはほぼ一定になりますが、ある地点(飽和点)を超えるとスループットは一定になり、レスポンスタイムは上昇し始めます。この飽和点がそのシステムを構成するサーバーの最大処理能力であり、このときの同時ユーザー数が最大同時ユーザー数です。さらにこの飽和点において、各サーバーのCPUが100%であればシステムが最大の処理能力を発揮していることになります。

システムにボトルネックが存在する場合には、飽和点に達してもCPUがフルに使用されていない、スループットが低い、レスポンスが遅い、飽和点に達してからスループットが急激に悪化する、などといった現象が起こります。

ボトルネックの例に関しては、「パフォーマンス設計-参考-」のP.85～86をご参照ください。



パフォーマンス・テストの目的は、システムがおかれているフェーズによって異なります。内部・外部設計フェーズでは、プロトタイプを用いてキャパシティー・パフォーマンスの大まかな見積もりを行います。統合テストフェーズでは、実プログラムを使用して、システムがパフォーマンス要件を満たしているかの確認や高負荷時の振る舞いの確認、最終のパラメーター・チューニングを行います。サービスイン後の本番稼動時においてパフォーマンス・トラブルが発生した場合は、その解決に向けたチューニングのためのパフォーマンス・テストを実施します。

パフォーマンス・テストにおいて確認すべき内容は、要件やシステムによって異なりますが、一般的には、キャパシティー・テスト、限界値の確認、高負荷状態での長時間耐久確認、などが挙げられます。キャパシティー・テストでは、想定される最大ユーザー数でアクセスした場合に正常に動作すること、レスポンスタイムが要件を満たしていることを確認します。限界値の確認では、想定以上の負荷をかけ、最大使用可能ユーザー数の見極めやその際のボトルネックを見極めます。高負荷状態での長時間耐久確認では、長時間(想定される連続稼動時間)の負荷をかけ、動作に異常がないことを確認します。

パフォーマンス・テストを実施する際は、事前に何を確認するかを明確にしておくことが重要であり、それによって、テスト対象、負荷ツール、取得情報と目標値などを検討します。パフォーマンス・テストにおいて負荷ツールを使用することは、例えば人海戦術によるテストケースの実施に伴うコストや、テスト後に行うデータの集計作業の負荷などを軽減することができ、結果としてテスト期間の短縮とコストの削減につながります。

負荷ツールの詳細に関しては、「パフォーマンス設計-参考-」のP.88～89をご参照ください。

Designing Web System Infrastructure with WAS V8.0

2. データの取得

- 2-1 パフォーマンス・データ取得の概要
- 2-2 OSによるパフォーマンス・データ取得
- 2-3 IHSによるパフォーマンス・データ取得
- 2-4 WASによるパフォーマンス・データ取得
- 2-5 パフォーマンス・データ取得のポイント

© 2012 IBM Corporation

8

Designing Web System Infrastructure with WAS V8.0

パフォーマンス・データ取得の概要

- パフォーマンス・データはさまざまな種類をさまざまな方法で取得可能
 - ◆ OS
 - サーバーのリソース使用状況
 - CPU、メモリー、ネットワークI/O、ディスクI/O、など
 - ◆ IHS
 - ユーザーアクセス数
 - レスポンスタイム
 - ◆ WAS
 - レスポンスタイム
 - ヒープ使用状況(GC発生状況)
 - Webコンテナ・スレッド状況
 - DB接続状況
- パフォーマンス・データ取得のポイント
 - ◆ 取得すべきパフォーマンス・データの明確化
 - ◆ 各機能の特徴や出力形式を理解した上でデータ取得方法を選択

© 2012 IBM Corporation
9

パフォーマンス・データは、さまざまな種類があり、さまざまな方法で取得することができます。同じデータであっても、複数の機能やツールによって取得することができるため、どの機能でどのパフォーマンス・データを取得することができるかを理解した上で、それぞれの特徴や出力形式などを考慮し、最適な方法を選択することが重要です。

一般的には、OSの機能やツールを使用してサーバー自体のCPUやメモリーといったリソース使用状況を確認します。IHSでは、詳細なアクセス・ログが取得できるため、全アクセス数やサーバーに入ってきた時点からクライアントにレスポンスを返すまでのレスポンスタイム(サーバー処理時間)を取得します。WASでは、各コンポーネント毎のレスポンスタイムやヒープ使用状況(GC発生状況)、Webコンテナ・スレッド状況、DB接続状況など、さまざまなパフォーマンス・データを取得することができます。

パフォーマンス・データ取得のポイントとしては、まず取得すべきデータを明確にしておくことが重要です。そして、各機能の特徴や出力形式を理解した上でデータ取得方法を選択する必要があります。

Designing Web System Infrastructure with WAS V8.0

2. データの取得

- 2-1 パフォーマンス・データ取得の概要
- 2-2 OSによるパフォーマンス・データ取得**
- 2-3 IHSによるパフォーマンス・データ取得
- 2-4 WASによるパフォーマンス・データ取得
- 2-5 パフォーマンス・データ取得のポイント

© 2012 IBM Corporation

10

Designing Web System Infrastructure with WAS V8.0

リソース使用率の確認

- OSの機能でリソース使用率の確認
 - ◆ AIX/Linux
 - vmstatコマンド
 - CPUやメモリーの使用状況の変化を一定時間間隔で表示
 - CPUネック時はkthr(カーネル・スレッド)とCPU使用率に注目
 - メモリネック時はメモリー使用量とページング発生状況に注目
 - svmonコマンド(AIX only)
 - メモリーの統計やセグメントに関する情報を表示
 - nmon performance / nmon Analyzer
 - AIX/Linuxのパフォーマンス分析ツール
 - 1画面にCPU使用率、メモリー使用量、ネットワークI/O、ディスクI/Oなどを表示
 - 分析結果をファイルに出力させることやグラフ化することも可能
 - ◆ Windows
 - パフォーマンス・モニター(perfmon)
 - メモリーやプロセッサ、ネットワークなどの利用状況のリアルタイム・データを収集
 - グラフ、ヒストグラム、レポート形式で表示
 - 閾値による警告の設定も可能
 - PsList
 - Javaのスレッド毎にCPU使用率を表示

各コマンドやツールの詳細は「パフォーマンス設計-参考-」のP.91～97をご参照ください。

11

© 2012 IBM Corporation

システムのCPUやメモリーといったリソース使用率は、非常に重要なパフォーマンス・データです。ミドルウェアやツール等でリソース使用状況を確認できるものもありますが、ここでは、OSの機能でそれらを確認する方法をご紹介します。

AIX/Linuxでは、vmstatコマンドを使用することで、CPUやメモリーの使用状況を確認することができます。CPUネック時およびメモリネック時の調査のアプローチについては、次ページで解説します。また、AIXのみになりますが、svmonコマンドを使用することでメモリーの統計やセグメントに関する情報を取得することができます。nmon performance および nmon Analyzer は、AIX/Linuxにおけるパフォーマンス分析ツールで、1画面に見やすく、CPU使用率、メモリー使用量、ネットワークI/O、ディスクI/O、などの情報を表示させることができ、分析結果をファイルに出力させたりグラフ化させることも可能です。

Windowsでは、パフォーマンス・モニターを使用して、メモリーやプロセッサ、ネットワークといったシステム・リソース使用率を確認することができます。また、PsListを使用することで、Javaのスレッド毎にCPU使用率を確認することもできます。

PsListコマンドによるパフォーマンス・データの取得の詳細については、下記のTechnoteを参照ください。

WAS Support Page – Technote「Using PsList to troubleshoot high CPU usage in Windows」

<http://www.ibm.com/support/docview.wss?uid=swg21304776>

各コマンドやツールの詳細に関しては、「パフォーマンス設計-参考-」のP.91～97をご参照ください。

Designing Web System Infrastructure with WAS V8.0

AIX/LinuxにおけるCPUネック時の挙動

■ kthr(カーネル・スレッド)とCPU使用率に注目

r: 実行可能なカーネル・スレッド。実行中のスレッド + CPU 待ちのスレッド
b: 待機キュー内のカーネル・スレッド。入出力待ちのスレッド

us: ユーザー・モードで使用されたCPU時間の割合。プロセスはそのアプリケーション・コード内部で実行
sy: システムモードで使用されたCPU時間の割合。カーネルプロセスおよびカーネル・リソースへのアクセスが必要な他のプロセスによって消費されたCPU
id: CPUが入出力の保留せずにアイドル状態になっている時間の割合
wa: ローカルディスクおよびNFSマウントのディスクの入出力が保留中で、CPUがアイドルだった時間の割合

vmstat 2 の出力 (AIX)

# vmstat 2																
kthr		memory		page		faults		cpu								
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa
1	0	22478	1677	0	0	0	0	0	188	1380	157	57	32	0	10	
1	0	22506	1609	0	0	0	0	0	214	1476	186	48	37	0	16	
0	0	22498	1582	0	0	0	0	0	248	1470	226	55	36	0	9	
2	0	22534	1465	0	0	0	0	0	238	903	239	77	23	0	0	
2	0	22534	1445	0	0	0	0	0	209	1142	205	72	28	0	0	
2	0	22534	1426	0	0	0	0	0	189	1220	212	74	26	0	0	
3	0	22534	1410	0	0	0	0	0	255	1704	268	70	30	0	0	
2	1	22557	1365	0	0	0	0	0	383	977	216	72	28	0	0	

実行キューの数がCPUの数より多いときは、CPU待ちのスレッドが存在している。
CPU待ちのスレッドが多くなると、パフォーマンスへの影響も大きくなる

us+syの割合が80%を超えると、プロセスが実行キューで待っている間に時間を費やしてしまう可能性がある。
waが25%を超えている場合は、ディスクの入出力が過多になっている (→ iostat コマンドなどによる確認)

© 2012 IBM Corporation

CPUネックの時に注目すべき項目は、vmstatレポートの2つの「kthr」(カーネル・スレッド) 欄と4つの「cpu」欄です。

kthr(r): CPUの数より多いときは、少なくとも1つのスレッドがCPUを待っています。CPU待ちのスレッド数が多いほど、パフォーマンスへの影響は大きくなります。最適の使用方法ではCPUが100%となりますが、マルチユーザー・システムのus + syの時間が80%を超えると、プロセスが実行キューで待っている間に時間を費やしてしまう可能性があり、その結果、レスポンスタイムとスループットが低下が発生する可能性があります。

cpu(wa): waitの実行中にディスクに対する未解決の入出力がある場合、その時間は入出力待ちに分類されます。プロセスが非同期通信I/Oを使用している場合を除き、ディスクに対する入出力要求では、要求が完了するまで呼び出しプロセスはブロック(またはスリープ)されます。プロセスの入出力要求が完了すると、呼び出しプロセスは実行キューに置かれます。つまり入出力が速く完了していれば、CPU 時間を多く使用できた可能性があります。なおwa 値が25%を超えている場合は、ディスク・サブシステムの平衡が正しく保てないことを示しているか、またはディスク集中のワークロードの結果である可能性があります。

Designing Web System Infrastructure with WAS V8.0

AIX/Linuxにおけるメモリネック時の挙動

■ メモリー使用量とページング発生状況に注目

avm: vmstatサンプルが収集された時点でアクティブであった仮想メモリー・ページ数(1ページ=4kB)
fre: 空きメモリー・ページの平均数

pi: ページング・スペースからページインされるページの数。ページング・スペースは仮想メモリーの一部で、ディスクに常駐している部分
po: ページング・スペースにページアウトされるページの数

vmstat 2 の出力 (AIX)

# vmstat 2		memory		page		faults		cpu	
kthr									
r	b	avm	fre	re	pi	po	fr	sr	cy
0	3	113659	135	0	2	2	108	323	0
0	2	113661	122	0	3	2	120	375	0
0	3	113662	128	0	10	3	134	432	0
1	5	113858	238	0	35	1	146	422	0
0	3	113969	127	0	5	10	153	529	0
0	3	113983	125	0	33	5	153	424	0
0	3	113682	121	0	20	9	154	470	0
0	4	113701	124	0	3	29	228	635	0

pi, poが常にゼロ以外の場合は、メモリーのボトルネックが存在する可能性あり

ページインやページアウトが多発すると、入出力待ちが多くなる

© 2012 IBM Corporation

メモリネックの場合も、同じくvmstatコマンドの出力結果を確認します。

memory(avm): vmstatサンプルが収集された時点でアクティブであった仮想メモリー・ページ数(1ページ=4kB)

memory(fre): 空きメモリー・ページの平均数

page(pi): ページング・スペースからページインされるページの数。ページング・スペースは仮想メモリーの一部で、ディスクに常駐している部分

page(po): ページング・スペースにページアウトされるページの数

piとpoの値が常にゼロ以外の場合は、メモリーのボトルネックが存在する恐れがあります。仮想メモリーの基本原理を考えると、適度にページングが起こってゼロ以外の値になるのは問題ありませんが、常にそういった状態に陥る時には注意が必要です。ページング・スペースからのページインおよびページアウトにより、出力に入出力待ちが多くなったり、ブロックされたキューのスレッドの数が多くなるなどの現象が起こり、パフォーマンスが低下する可能性があります。

Designing Web System Infrastructure with WAS V8.0

2. データの取得

- 2-1 パフォーマンス・データ取得の概要
- 2-2 OSによるパフォーマンス・データ取得
- 2-3 IHSによるパフォーマンス・データ取得**
- 2-4 WASによるパフォーマンス・データ取得
- 2-5 パフォーマンス・データ取得のポイント

© 2012 IBM Corporation

14

Designing Web System Infrastructure with WAS V8.0

IHSアクセス・ログからアクセス数およびスループットの取得

- アクセス数
 - ◆ IHSのアクセス・ログからアクセス数を確認可能
 - ◆ IHSアクセス・ログの時間表示はリクエストを受けた時間
 - ◆ アクセス数からスループットを算出可能

access.log

リクエストを受けた時間

```

9.170.251.57 - - [14/Mar/2009:15:19:07 +0900] "GET /Sample/sleep HTTP/1.1" 503 424
9.170.251.98 - - [14/Mar/2009:15:18:57 +0900] "GET /Sample/sleep HTTP/1.1" 200 417
9.188.52.31 - - [14/Mar/2009:15:19:01 +0900] "GET /Sample/sleep HTTP/1.1" 200 417
9.170.251.57 - - [14/Mar/2009:15:19:04 +0900] "GET /Sample/sleep HTTP/1.1" 200 417
9.170.251.98 - - [14/Mar/2009:17:27:17 +0900] "GET /JVM/input.html HTTP/1.1" 200 1494
9.170.251.98 - - [14/Mar/2009:17:27:18 +0900] "GET /JVM/theme/Master.css HTTP/1.1" 200 731
9.170.251.98 - - [14/Mar/2009:17:28:51 +0900] "GET /JVM/input.html HTTP/1.1" 200 1494
9.170.251.98 - - [14/Mar/2009:17:28:51 +0900] "GET /JVM/theme/Master.css HTTP/1.1" 200 731
9.170.251.98 - - [14/Mar/2009:17:30:13 +0900] "GET /JVM/input.html HTTP/1.1" 200 1494
9.170.251.98 - - [14/Mar/2009:17:30:13 +0900] "GET /JVM/theme/Master.css HTTP/1.1" 200 731
9.170.251.98 - - [14/Mar/2009:17:30:46 +0900] "GET /JVM/input.html HTTP/1.1" 304 -
9.170.251.98 - - [14/Mar/2009:17:30:46 +0900] "GET /JVM/theme/Master.css HTTP/1.1"

```

IHSのアクセス・ログから処理したリクエスト数を時間間隔で割ると、スループット(アクセス件数/秒)が計算可能

© 2012 IBM Corporation
15

IHSのアクセス・ログから、アクセス数を確認することができます。また、アクセス・ログにはリクエストを受けた時間が記録されますので、処理したリクエストの数を時間間隔で割ることにより、スループット(アクセス件数/秒)を算出することができます。

Designing Web System Infrastructure with WAS V8.0

IHSアクセス・ログからレスポンスタイムの取得

- レスポンスタイム
 - ◆ IHSのアクセス・ログからレスポンスにかかっている時間を確認可能
 - ◆ デフォルトでは記載されないので手動で追加する必要あり
 - ◆ LogFormatディレクティブ
 - %D : レスポンスタイムをマイクロ秒単位で記録 (IHS V2.0、V6.0、V6.1、V7.0、V8.0)
 - %T : レスポンスタイムを秒単位で記録 (IHS V1.3、V2.0、V6.0、6.1、V7.0、V8.0)

httpd.conf

LogFormat "%h %l %u %t %>%r%" "%>s %b [%D]" common

access.log

レスポンスタイムをログに記録させる設定が必要

9.170.251.57 - - [20/Mar/2009:14:18:31 +0900] "GET /JVM/index.html HTTP/1.1" 404 61 [1234375]
 9.170.251.57 - - [20/Mar/2009:14:18:46 +0900] "GET /JVM/input.html HTTP/1.1" 200 1494 [62500]
 9.170.251.57 - - [20/Mar/2009:14:18:46 +0900] "GET /JVM/theme/Master.css HTTP/1.1" 200 731 [31250]
 9.170.251.57 - - [20/Mar/2009:14:18:52 +0900] "POST /JVM/TableView HTTP/1.1" 200 8924 [5718750]

アクセス・ログにレスポンスタイムが記録される

表計算ソフトなどを用いて、レスポンスの遅いリクエストを判断

A	B	C	D	E	F
クライアントIPアドレス	アクセス時間	HTTPリクエスト	HTTPコード	バイト数	レスポンスタイム
9.170.251.57	[20/Mar/2009:14:18:31 +0900]	GET /JVM/index.html HTTP/1.1	404	61	1234375
9.170.251.57	[20/Mar/2009:14:18:46 +0900]	GET /JVM/input.html HTTP/1.1	200	1494	62500
9.170.251.57	[20/Mar/2009:14:18:46 +0900]	GET /JVM/theme/Master.css HTTP/1.1	200	731	31250
9.170.251.57	[20/Mar/2009:14:18:52 +0900]	POST /JVM/TableView HTTP/1.1	200	8924	5718750
9.170.251.57	[20/Mar/2009:14:20:00 +0900]	GET /JVM/index.html HTTP/1.1	304	-	0
9.170.251.57	[20/Mar/2009:14:20:29 +0900]	GET /JVM/index.html HTTP/1.1	200	228	0

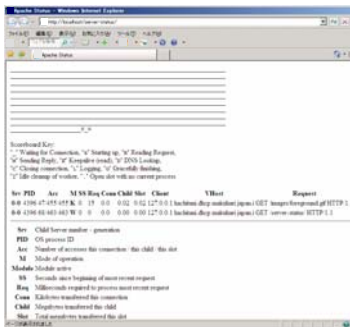
© 2012 IBM Corporation

IHSのアクセス・ログからレスポンスタイムを取得することができます。デフォルトではそのための設定がされていないため、レスポンスタイムは取得できません。レスポンスタイムを取得するには、LogFormatディレクティブでレスポンスタイムを表示させる設定が必要です。取得したアクセス・ログは表計算ソフトなどで解析し、レスポンスタイムが遅いリクエストを判断します。

Designing Web System Infrastructure with WAS V8.0

mod_statusから取得できるデータ

- mod_status
 - ◆ プロセスやスレッドの状態を確認可能
 - ◆ mod_statusのモジュールをアクティブ化して下記URLにアクセス
 - `http://<hostname>/server-status/`
 - ◆ 表示される項目
 - リクエストを扱っているワーカーの数
 - アイドル(リクエストを扱っていない)ワーカーの数
 - 各ワーカーの状態、ワーカーが扱ったリクエスト数、ワーカーが送った総バイト数
 - 総アクセス数と総バイト数
 - サーバーが起動もしくは再起動された時刻と動作している時間
 - 平均1秒あたりのリクエスト数、1秒あたりの送られたバイト数、リクエストあたりのバイト数
 - 現時点のホストと処理されているリクエスト



© 2012 IBM Corporation
17

IHSのmod_statusを使用すれば、プロセスやスレッドの状態を確認することができます。mod_statusを使用するには、IHS構成ファイルを修正し、モジュールのアクティブ化や適切なアクセス権限付与を行う必要があります。設定後に「`http://<hostname>/server-status/`」にアクセスすると、プロセスやスレッドに関する情報が表示されます。

Designing Web System Infrastructure with WAS V8.0

mod_mpmstats

- mod_mpmstatsの設定
 - ◆ 下記のように、サーバー活動状況の統計情報を取得することが可能

```
[Wed Jan 21 16:30:16 2009] [notice] mpmstats: rdy 598 bsy 2 rd 0 wr 0 ka 2 log 0 dns 0 cls 0
```

rdy	待機状態のスレッド数
bsy	処理中のスレッド数
rd	読み取り中のスレッド数
wr	書き込み中のスレッド数
ka	キープアライブ接続中のスレッド数
log	ログ出力中のスレッド数
dns	DNSルックアップ中のスレッド数
cls	終了処理中のスレッド数

- ◆ IHS v8.0(v7.0)ではデフォルトOnであり、600秒毎にerror.logに出力
- ◆ httpd.conf

```
LoadModule mpmstats_module modules/debug/mod_mpmstats.so
<IfModule mod_mpmstats.c>
ReportInterval 600
TrackModules On
</IfModule>
```

デフォルトON
(不要な場合は、ロードしない)

600秒毎にデータを取得

© 2012 IBM Corporation
18

IHSのmod_mpmstatsを使用すれば、スレッドのリクエストの処理状況を確認することができます。mod_mpmstatはデフォルトでONで、error.logに10分間隔で記録します。IHS構成ファイルを修正することで、設定をOFFにしたり、ロギング間隔を変更することができます。

mod_mpmstatsの情報を取得する要件がなく、(600秒ごとですが)パフォーマンスへの影響を考慮する場合には、mod_mpmstatsをロードしないでください。

Designing Web System Infrastructure with WAS V8.0

2. データの取得

- 2-1 パフォーマンス・データ取得の概要
- 2-2 OSによるパフォーマンス・データ取得
- 2-3 IHSによるパフォーマンス・データ取得
- 2-4 WASによるパフォーマンス・データ取得**
- 2-5 パフォーマンス・データ取得のポイント

© 2012 IBM Corporation

19

Designing Web System Infrastructure with WAS V8.0

要求メトリックからレスポンスタイムの取得

- 要求メトリック
 - ◆ 要求メトリックをONに設定すると各コンポーネントのレスポンスタイムがログに出力
 - ◆ どのコンポーネントの処理で遅延が発生しているかを判断

SystemOut.log

```
[09/03/15 14:19:07:266 JST] 00000030 PmiRmArmWrapp I PMRM0003I:
parent:ver=1,ip=9.170.251.57,time=1129785202984,pid=2720,reqid=1,event=1 -
current:ver=1,ip=9.170.251.57,time=1129785202984,pid=2720,reqid=2,event=1 type=JDBC
detail=SELECT * from staff elapsed=3125
[09/03/15 14:19:07:469 JST] 00000030 ServletWrapp A SRVE0242I: [JvmHandsOn] [JVM]
[/output.jsp]: 初期化が正常に行われました。
[09/03/15 14:19:08:172 JST] 00000030 PmiRmArmWrapp I PMRM0003I:
parent:ver=1,ip=9.170.251.57,time=1129785472812,pid=2968,reqid=0,event=1 -
current:ver=1,ip=9.170.251.57,time=1129785202984,pid=2720,reqid=1,event=1 type=URI
detail=/JVM/TableView elapsed=14313
```

DB接続のレスポンスタイム

Webコンテナでのレスポンスタイム

各コンポーネントのレスポンスタイムが記録されるので、その差に注目
差が大きいところに多くの時間を取られていることになる

A	B	C	D	E	F
日付/時刻	parent	current	type	detail	elapsed
[09/03/15 14:19:07:266 JST]	parent:ver=1,ip=9.170.251.57,time=1129785202984	current:ver=1,ip=9.170.251.57,time=1129785202984	type=JDBC	detail=SELECT * from staff	3125
[09/03/15 14:19:08:172 JST]	parent:ver=1,ip=9.170.251.57,time=1129785472812	current:ver=1,ip=9.170.251.57,time=1129785202984	type=URI	detail=/JVM/TableView	14313

リクエスト全てに書き込みを行うと、要求メトリックによる出力自体がパフォーマンスに影響を与える可能性があるため、フィルターにより対象となるリクエストのみ出力させる

© 2012 IBM Corporation

20

WASの要求メトリックの機能を使用すると、各々のコンポーネントでのレスポンスタイムがログに出力されるため、レスポンスタイムで多くの時間を取られているリクエストがどのコンポーネントで遅くなっているのかを判断することができます。出力先に関して、プラグインはhttp.plugin.logに出力され、その他はSystemOut.logに出力されます。

ここではレスポンスタイムの差に注目します。その差が大きいところで遅延が発生していることがわかります。ただし、リクエスト全てに書き込みを実行すると、ログへの書き込み自身がパフォーマンスに影響を与える可能性があるため、本番環境で取得が必要な場合は、必ずフィルターにより取得対象を絞って出力させるようにします。

要求メトリックを有効にしている場合、それ自体がパフォーマンスに影響を与える可能性があるため、基本的に、パフォーマンス・テストの際は使用せず、パフォーマンス・テストで問題があった場合にボトルネックを見つけるための手段として使用します。また、フィルターにより対象となるリクエストを絞ることもできます。

要求メトリックのログの見方

- 要求メトリックの出力
 - ◆ 出力は親 (parent: 呼び出し元)、子 (current: 呼び出された側) で構成
 - ◆ リクエストの流れはリクエストIDを見て判断
 - ◆ elapsedが処理にかかった時間 (ms) を示す

http_plugin.log

```
[Sun Mar 15 13:21:20 2009] 00000684 00001284 - PLUGIN:
parent:ver=1,ip=9.170.251.57,time=1125201907343,pid=1668,reqid=0,event=1 -
current:ver=1,ip=9.170.251.57,time=1125201907343,pid=1668,reqid=0,event=1 type=HTTP
detail=/JVM/TableView elapsed=23203 bytesIn=14 bytesOut=8871
```

SystemOut.log

```
[09/03/15 13:21:06:375 JST] 00000034 PmiRmArmWrapp I PMRM0003I:
parent:ver=1,ip=9.170.251.57,time=1125202727641,pid=6048,reqid=1,event=1 -
current:ver=1,ip=9.170.251.57,time=1125202727641,pid=6048,reqid=2,event=1 type=JDBC
detail=SELECT * from staff elapsed=359
[09/03/15 13:21:19:953 JST] 00000034 PmiRmArmWrapp I PMRM0003I:
parent:ver=1,ip=9.170.251.57,time=1125201907343,pid=1668,reqid=0,event=1 -
current:ver=1,ip=9.170.251.57,time=1125202727641,pid=6048,reqid=1,event=1 type=URI
detail=/JVM/TableView elapsed=21765
```

DB接続のレスポンスタイム

Webコンテナでのレスポンスタイム

elapsedが各コンポーネントのレスポンスタイムを示す

ID(reqid=0) とこれを親にもつ子が結びつく

同様にID(reqid=1)とこれを親にもつ子が結びつく

© 2012 IBM Corporation

21

要求メトリックにより出力されるデータの形式および見方について解説します。

レコードは親 (parent: メソッドの呼び出し元) と子 (current: 呼び出された側) で構成されています。データの中のreqidがリクエストIDを示し、リクエストIDが同じ親と子の流れからリクエストの流れを判断します。その処理がどのコンポーネントの処理かをtypeで判断します (HTTPはWebサーバーのプラグインにきたリクエスト、URIはWebコンポーネント、JDBCはDBに対しての処理を示します)。elapsedの欄がその処理にかかった経過時間を示します。1リクエストにおいて、この開きがもっとも大きいところに注目します。時間の開きが大きいところがもっとも処理に時間を取られている点であると判断できます。

詳細は下記のInformation Centerの記載も参照ください。

WAS V8.0 Information Center「要求メトリックのパフォーマンス・データ」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/rprf_tracerecord.html

Designing Web System Infrastructure with WAS V8.0

要求メトリックのフィルタリング機能

- 要求メトリックのフィルター
 - ◆ 要求メトリックの出力によるパフォーマンスダウンを抑えるために特定データのみを出力 (EJB/URI/SourceIP/WebService/JMS)

フィルターの種類を選択

/JVM/TableViewというURIに対してフィルターをかけた例。
他のURIをアクセスしても要求メトリックの結果は出力されない

フィルターの値を入力

使用可能にする、を選択

要求メトリックの設定などの詳細は「パフォーマンス設計-参考-」のP.99～101をご参照ください。

© 2012 IBM Corporation

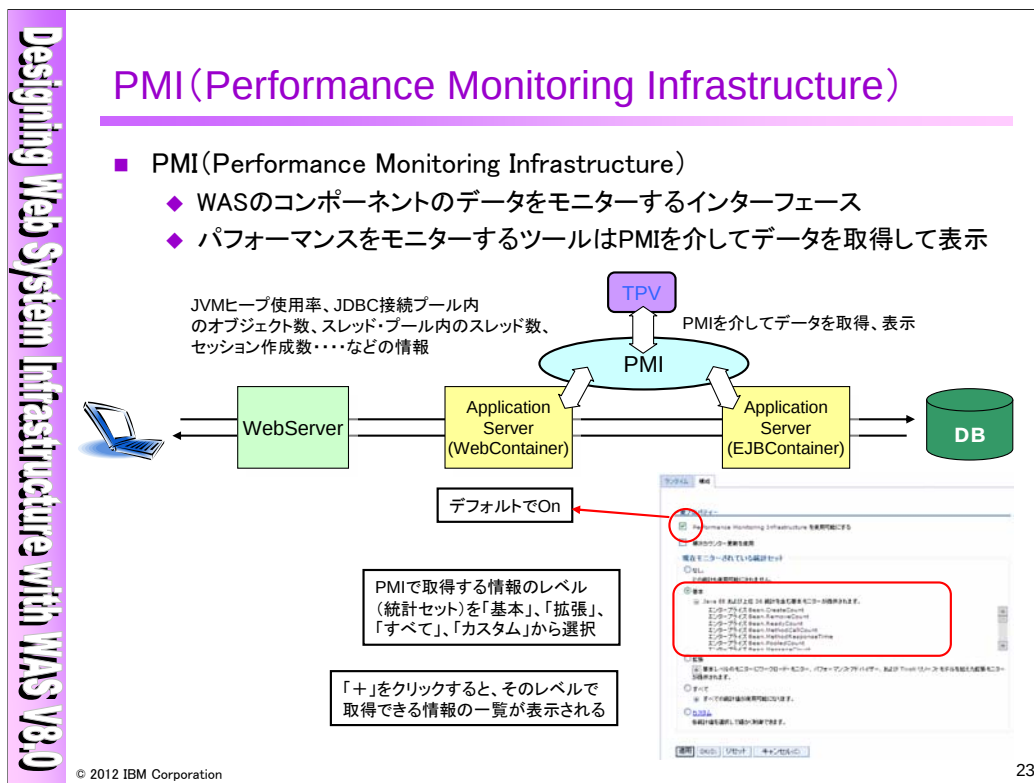
22

全てのリクエストに対して要求メトリックの機能でログに出力させていると、その出力がパフォーマンスに影響を与える可能性がありますので、フィルターを指定し、取得すべきデータを絞ります。設定できるフィルターは、EJB、URI、SourceIP、WebService、JMSの5種類です。

URIとSourceIPはHTTPリクエストに対するフィルターで、それぞれ要求するURI、発信元のIPアドレスを指定します。EJBはEJBメソッドの絶対パスを指定します。WebServiceとJMSはV6.0からの新機能で、WebServiceフィルターの値はWSDLのポート名、オペレーション名およびトランスポート名の組み合わせを指定します。JMSフィルターの値はバス名および宛先名の組み合わせを指定します。

上記の例は、/JVM/TableViewというURIでフィルターをかけた例です。この例では他のURIをアクセスしても要求メトリックの結果は出力されません。/JVM/TableViewというURIにアクセスしたときのみログに出力されます。

要求メトリックの設定などの詳細に関しては、「パフォーマンス設計-参考-」のP.99～101をご参照ください。



PMIはWASにおけるパフォーマンス・データ管理を行なう仕組みであり、デフォルトで使用可能になっています。PMIの設定画面からは、取得する情報のレベルを基本、拡張、全て、カスタムから選択します。そのレベルで取得できる情報の一覧を見ることも可能です。またカスタムについては、データ項目一つ一つに対してモニターの有無を指定することも可能です。

Designing Web System Infrastructure with WAS V8.0

Tivoli Performance Viewer

- Tivoli Performance Viewer
 - ◆ WASが提供しているパフォーマンス・モニター・ツール
 - ◆ 管理コンソールに統合
 - [モニターおよびチューニング]→[Performance Viewer]
 - ◆ リアルタイムでデータを表示
 - ◆ ログの取得と再生が可能

右側に取得値がグラフで表示
(表にすることも可能)

対象サーバーを選択して、
[モニターの開始]を押す

モニターする項目を選んで
[モジュールの表示]

24

TivoliPerformanceViewer (TPV) はWAS付属のモニタリング・ツールです。TPVはリアルタイムでデータを表示し、その記録をログに取得し、再生することも可能です。管理コンソールに統合されているため、ブラウザで管理コンソールに接続できる環境があればTPVを使用することができます。

Tivoli Performance Viewer で取得できるデータ	
カテゴリ	取得情報
エンタープライズBean	応答時間やライフサイクルのアクティビティ、Enterprise BeanメソッドおよびEnterprise Beanによって使用されるリソース・インターフェースの情報
JDBC接続プール	新規に作成された接続数など、データソースの接続プールについての使用情報
JCA接続プール	JCAについての使用情報
JVMランタイム	使用メモリなどJVMIに関する情報
サーブレット・セッション・マネージャー	アクティブなセッションの平均数など、HTTP セッションの使用状況
スレッド・プール	オブジェクト・リクエスト・ブローカー (ORB) スレッドのスレッド・プール、HTTP要求を処理するWebコンテナ・プールについての情報
トランザクション・マネージャー	アクティブ・トランザクションの平均数など、トランザクションマネージャーのパフォーマンス情報
Webアプリケーション	ロードされたサーブレットの数、サーブレット要求の応答時間など、選択されたサーバーの情報
ORB	オブジェクト参照ルックアップ時間などORBIに関する情報
ワークロード管理	IOP要求の数など、クライアントとサーバーのワークロード管理に関する情報
動的キャッシング	メモリ内のキャッシュ・サイズなど、動的キャッシュ・サービスに関する情報
Webサービス	ロードされたWebサービスの数など、Webサービスに関する情報
Webサービス・ゲートウェイ	同期要求の数など、Webサービス・ゲートウェイに関する情報
システム・データ	CPU使用率など、マシン(ノード)に関する情報(複数のサーバー・バージョンのノード・エージェントでのみ使用可能)
アラーム・マネージャー	アラームに関する情報
オブジェクト・プール	オブジェクト・プールに関する情報
スケジューラー	スケジューラー・サービスに関する情報
HA マネージャー	HAマネージャーに関する情報
DCS統計	DCSスタックを使用した送信メッセージ数など、DCSIに関する情報
SipContainerModule	アプリケーション・サーバーのSIPContainerに関する情報
ポートレット・アプリケーション	ポートレット・アプリケーションに関する情報
SIBサービス	SIBサービスに関する情報

© 2012 IBM Corporation

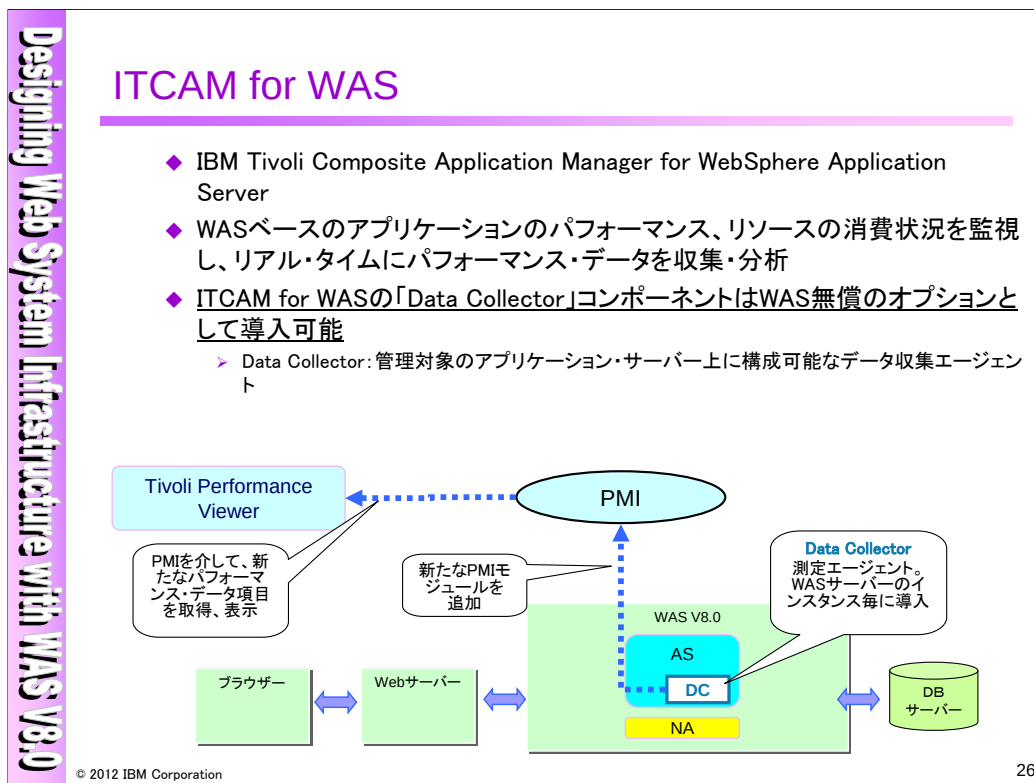
25

こちらはTPVで取得可能な情報の一覧です。PMIで取得可能なデータはカテゴリで分類されており、そのカテゴリと取得できる情報の例を示しています。またカテゴリは其中でさらにサブカテゴリに分類されるものもあります。取得データ一つ一つについての詳細な情報は、統計セットから「カスタム」を選択すると参照することが可能です。

カテゴリの詳細と各カテゴリのカウンター項目の詳細については、下記Information Centerのリンクとそのページ下部のサブトピックのリンクを参照ください。

WAS V8.0 Information Center「PMI データ編成」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/rprf_dataorg.html



ITCAM for WAS (IBM Tivoli Composite Application Manager for WebSphere Application Server) はWASに同梱されている無償ツールで、WASに追加インストールすることができます。

ITCAM for WAS ではTPVでモニタリングできる項目に加えてアプリケーション・サーバーおよびアプリケーション・サーバー上で実行されているアプリケーションのパフォーマンス、リソースの消費状況に関する情報の収集・分析に役立ちます。

ITCAM for WAS をインストールするとWASにData CollectorというITCAMのデータ収集エージェントが組み込まれPMI経由でデータを取得します。ITCAM for WASで取得したデータは他のTPVの項目と同様に管理コンソールから表示させることができます。

Designing Web System Infrastructure with WAS V8.0	ITCAM for WASによって追加されるパフォーマンス・データ項目	
	パフォーマンス・データ項目	説明
	90%CPUUsage	CPU 使用量を収集してから、90% メディアン の CPU 使用量をミリ秒単位で計算します。
	90%ResponseTime	応答時間を収集してから、90% メディアン の時間をミリ秒単位で計算します。
	AverageCPUUsage	CPU 使用量を収集してから、平均 CPU 使用量をミリ秒単位で計算します。
	AverageResponseTime	応答時間を収集してから、平均応答時間をミリ秒単位で計算します。
	LastMinuteAverageCPUUsage	最終段階の CPU 使用量を収集してから、平均 CPU 使用量をミリ秒単位で計算します。
	LastMinuteAverageResponseTime	最終段階の応答時間を収集してから、平均応答時間をミリ秒単位で計算します。
	MaximumCPUUsage	CPU 使用量を収集してから、最大 CPU 使用量をミリ秒単位で計算します。
	MaximumResponseTime	応答時間を収集してから、最大応答時間をミリ秒単位で計算します。
	MinimumCPUUsage	CPU 使用量を収集してから、最小 CPU 使用量をミリ秒単位で計算します。
	MinimumResponseTime	応答時間を収集してから、最小応答時間をミリ秒単位で計算します。
	RequestCount	要求の総数。
© 2012 IBM Corporation		27

ITCAM for WASを使用することで追加で取得できるデータ項目です。

Designing Web System Infrastructure with WAS V8.0

ITCAM for WASの利用手順(1/2)

- ITCAM for WASの開始
 - ◆ ITCAM for WAS(Data Collector)を導入、構成後、PMIコンソール画面上、対象となるアプリケーション・サーバー上に「追加プロパティ」が表示される
 - ◆ 該当プロパティをクリックし、ランタイム・タブにて、「Start Monitoring」ボタンを押下
- ITCAM for WASのパフォーマンス項目の有効化
 - ◆ PMIコンソール画面で、「カスタム」の統計セットを選択
 - ◆ 「ITCAM Application Performance」を展開し、モニターするデータ項目を選択し、使用可能にする



選択	カウンター	タイプ	説明	状況
☐	90%CPUUsage	CountStatistic	このメトリックは CPU 使用量を収集してから、90% メディアン CPU 使用量をミリ秒単位で計算します。	使用可能
☐	90%ResponseTime	CountStatistic	このメトリックは応答時間を収集してから、90% メディアンの時間をミリ秒単位で計算します。	使用可能
☐	AverageCPUUsage	AverageStatistic	このメトリックは CPU 使用量を収集してから、平均 CPU 使用量をミリ秒単位で計算します。	使用可能
☐	AverageResponseTime	AverageStatistic	このメトリックは応答時間を収集してから、平均応答時間をミリ秒単位で計算します。	使用可能
☐	LastMinuteAverageCPUUsage	AverageStatistic	このメトリックは最終60秒間の CPU 使用量を収集してから、平均 CPU 使用量をミリ秒単位で計算します。	使用可能
☐	LastMinuteAverageResponseTime	AverageStatistic	このメトリックは最終60秒間の応答時間を収集してから、平均応答時間をミリ秒単位で計算します。	使用可能
☐	MaximumCPUUsage	CountStatistic	このメトリックは CPU 使用量を収集してから、最大 CPU 使用量をミリ秒単位で計算します。	使用可能

© 2012 IBM Corporation

28

ITCAM for WAS を導入するとPMI コンソール上で追加プロパティに「ITCAM for WebSphere Application Server」という項目が追加されます。

「Start Monitoring」をクリックするとITCAM for WASが開始されます。

必要に応じてITCAM for WASのパフォーマンス項目を選択して使用可能にします。

Designing Web System Infrastructure with WAS V8.0

ITCAM for WASの利用手順(2/2)

- ITCAM for WASパフォーマンス・データのモニタリング
 - ◆ Tivoli Performance Viewを通して、事前選択されたパフォーマンス・データ項目をリアルタイムでデータを表示

ITCAM Application Performance		
☐	RequestCount (?)	90.0
☐	AverageResponseTime (?)	1.7333333
☐	LastMinuteAverageResponseTime (?)	0.0
☐	AverageCPUUsage (?)	1.6666666
☐	LastMinuteAverageCPUUsage (?)	0.0
/snoop		
☒	RequestCount (?)	90.0
	AverageResponseTime (?)	1.7333333
	LastMinuteAverageResponseTime (?)	16.0
	AverageCPUUsage (?)	0.0
	LastMinuteAverageCPUUsage (?)	0.0
	...	1.6666666
	...	15.0
	...	0.0
	...	0.0

有償のITCAM管理サーバーとの統合の詳細は「パフォーマンス設計-参考-」のP.105～106をご参照ください。

29

モニタリング・データの表示方法は、通常のTPVでモニタリングできる項目と同様です。

別途ライセンスが必要になりますが、パフォーマンスや問題判別の統合管理サーバーである ITCAM管理サーバーとの統合に関しては、「パフォーマンス設計-参考-」のP.105～106をご参照ください。

Designing Web System Infrastructure with WAS V8.0

2. データの取得

- 2-1 パフォーマンス・データ取得の概要
- 2-2 OSによるパフォーマンス・データ取得
- 2-3 IHSによるパフォーマンス・データ取得
- 2-4 WASによるパフォーマンス・データ取得
- 2-5 パフォーマンス・データ取得のポイント**

© 2012 IBM Corporation

30

Designing Web System Infrastructure with WAS V8.0

WAS V8.0パフォーマンス・データ取得のポイント

- 1. パフォーマンス・テスト内容の明確化
 - ◆ キャパシティー・テスト
 - ◆ 限界値の確認
 - ◆ 高負荷状態での長時間耐久確認

- 2. 取得すべきデータおよび目標値の明確化
 - ◆ レスポンスタイム
 - ◆ スループット
 - ◆ システム・リソース

- 3. データ取得方法およびデータ分析方法の検討
 - ◆ OS/ミドルウェアの機能を使用
 - ◆ 使用実績のあるツールを使用
 - ◆ できるだけシステムのパフォーマンスに影響を与えないツールを使用

© 2012 IBM Corporation
31

ここで、WAS V8.0におけるパフォーマンス・データ取得のポイントをまとめます。

まずは、ただパフォーマンス・テストといっても、その内容や目的は多種ありますので、事前に明確にしておくことが重要です。キャパシティーを確認するだけのテストなのか、限界値を確認するためのテストなのか、高負荷状態での長時間耐久確認を行うテストなのか、などです。

テスト内容が明確になったら、それに応じて取得すべきデータおよび目標値を明確にします。お客様要件にも依存しますが、あくまでもユーザーにとってのレスポンスタイムを重視したテストにするのか、スループットも確認するのか、どのシステム・リソースに関するデータ取得を行うのか、などです。

取得すべきデータが明確になったら、それをどのように取得するかを検討します。既述の通り、同じデータでも、OSやミドルウェア、ツールなどさまざまな方法で取得することができるため、テストの目的に応じた機能を提供している方法を選択します。製品の機能を使用するのか、あるいは、それ自体の負荷までシビアに考慮する必要があるのか他のツールを使用するのか、データ収集や分析を効率的に行うために過去に使用実績のあるツールを使用するのか、などです。また、ただデータを取得するだけでなく、その後に分析する必要があるのか、データ分析機能の有無なども考慮に入れて検討します。

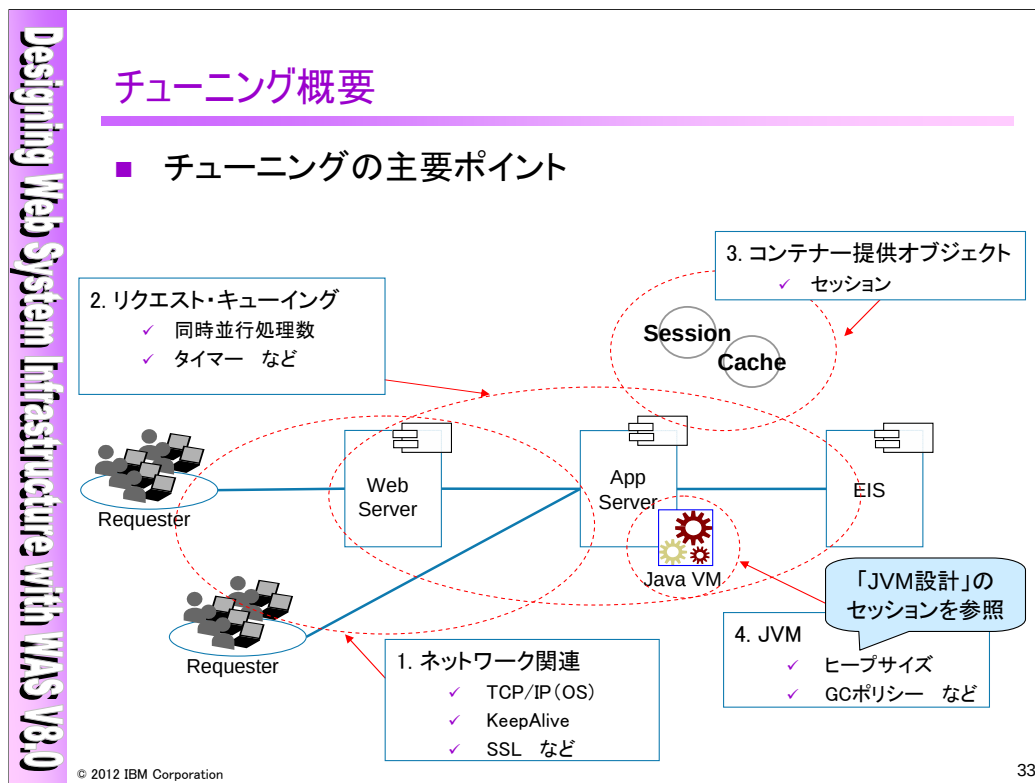
Designing Web System Infrastructure with WAS V8.0

3. チューニング

- 3-1 チューニング概要**
- 3-2 ネットワーク関連のチューニング
- 3-3 リクエスト・キューイング
- 3-4 コンポーネント別のチューニング
- 3-5 パフォーマンス・チューニングのポイント

© 2012 IBM Corporation

32



この章では、システム的设计を行なう上で必ず検討項目に入るような代表的なチューニング・パラメーターをネットワーク関連のチューニング、リクエスト・キューイング、コンテナ提供オブジェクトのような各コンポーネント別のチューニングについて解説します。JVMのチューニングについては、「JVM設計」のセッションを参照ください。

Designing Web System Infrastructure with WAS V8.0

チューニング・テンプレート V7.0.0.9～

- パフォーマンス・パラメーターを設定するwsadminスクリプト
 - ◆ アプリケーション・サーバーまたはクラスターに適用
 - ◆ チューニングの開始点として使用
 - ◆ 標準/ピーク/開発の3種類

設定されるパラメーターの例(抜粋)

パラメーター	デフォルト(標準)	ピーク	開発
JVMヒープサイズ(MB)	50 (最小) / 256 (最大)	512 (最小) / 512 (最大)	256 (最小) / 512 (最大)
Verbose GC	OFF	ON	ON
PMI統計セット	基本	なし	なし
データソース接続プール・サイズ(*)	1 (最小) / 10 (最大)	10 (最小) / 50 (最大)	
プリペア・ステートメント・キャッシュ・サイズ(*)	10	50	

(*) の項目は、スクリプト実行時点で定義されているもののみに適用

プロファイル作成時に表示されるのは、アプリケーション・サーバー・プロファイル作成時のみ



© 2012 IBM Corporation

34

WAS V8では、アプリケーション・プロファイルを新規作成するときに、サーバー・ランタイムのパフォーマンス・チューニング設定を「標準」、「ピーク」、「開発」から選択することができます。デプロイメント・マネージャー・プロファイルやカスタム・プロファイルの作成時には、この選択項目はありません。

この3種類のチューニング設定は、WAS V7.0.0.9で導入されたチューニング・テンプレートがベースになっています。このチューニング・テンプレートは、パフォーマンス・チューニングの開始点として使用するためのデフォルトのチューニング・パラメーターを設定するwsadminスクリプトです。JVMヒープ・サイズやデータソースの接続プール・サイズなどの値が自動的に設定されます。ただし、デプロイされているアプリケーションなどによらず固定値ですので、あくまでもパフォーマンス・チューニングの開始点として使用し、スクリプトの実行後、必ずアプリケーションやサーバー特性に合わせて、パフォーマンス・チューニングを実施してください。

設定されるパラメーターの詳細については、下記のInformation Centerを参照ください。

WAS V8.0 Information Center「事前定義のチューニング・テンプレートを使用したアプリケーション・サーバーのチューニング」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/tprf_tuneappserv_script.html

WAS V7 Information Center「定義済みの調整テンプレートを使用したアプリケーション・サーバーの調整」

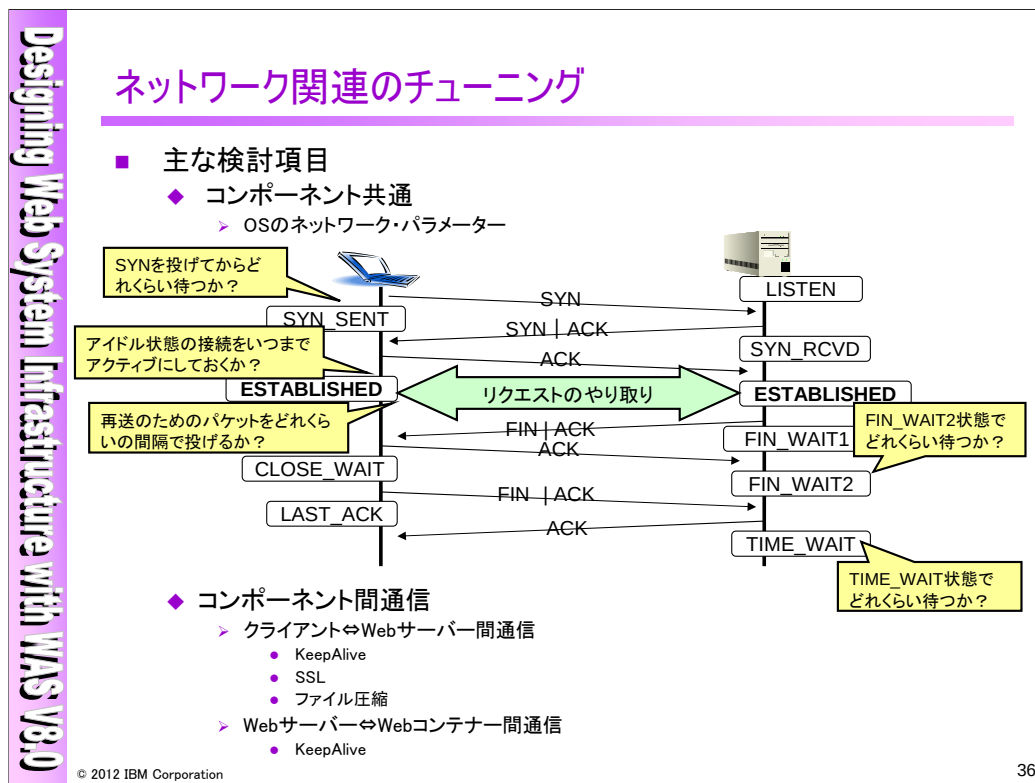
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/tprf_tuneappserv_script.html

Designing Web System Infrastructure with WAS V8.0

3. チューニング

- 3-1 チューニング概要
- 3-2 ネットワーク関連のチューニング**
- 3-3 リクエスト・キューイング
- 3-4 コンポーネント別のチューニング
- 3-5 パフォーマンス・チューニングのポイント

© 2012 IBM Corporation 35



ネットワーク関連のチューニングとして、大きく2種類に分けられます。1つ目は、コンポーネント共通に関係のあるOSのネットワークに関するパラメーターのチューニングです。TCP/IPのコネクションに関するパラメーターはミドルウェアでも設定できる場合がありますが、OSが持つパラメーターによるものも多くあります。2つ目は、各コンポーネント間の通信におけるパラメーターのチューニングです。クライアントとWebサーバー間の通信では、Keep AliveやSSL、ファイル圧縮などの機能に関する考慮が必要になります。WebサーバーとWebコンテナ間の通信では、KeepAliveの機能に関する考慮が必要になります。

Designing Web System Infrastructure with WAS V8.0

OSのネットワーク・パラメーター - TCP/IP接続タイムアウト

- AIXの例
 - ◆ tcp_keepinit (デフォルト150 /0.5秒単位: 推奨値 40 = 20秒)
 - TCPコネクション初期タイムアウト値

ネットワーク障害などでSYNIに対する返事がないとき、この値に従って接続が破棄される

 - ◆ tcp_keepidle (デフォルト14400 /0.5秒単位: 推奨値 600 = 5分)
 - 通信を行っていないTCPコネクションを確認する時間
 - ◆ tcp_keepintvl (デフォルト150 /0.5秒単位: 推奨値 10 = 5秒)
 - tcp_keepidle時に送信したパケットに返事がない際に、再度確認のために送るパケットの送信間隔
 - ◆ tcp_keepcnt (デフォルト8)
 - tcp_keepidle時に送信したパケットに返事がない際に再度確認のために送るパケットの数

$(tcp_keepidle) + (tcp_keepintvl) \times (tcp_keepcnt) = 2時間10分$
 デフォルトでは2時間10分経てば接続はOSが切断する

 - ◆ tcp_finwait2 (デフォルト1200 /0.5秒単位)
 - FIN_WAIT2状態で待つ時間
 - ◆ tcp_timewait (デフォルト1 /15秒単位)
 - TIME_WAIT状態で待つ時間

接続を終了する前にサーバー側は「TIME_WAIT」状態で15秒は待つ

© 2012 IBM Corporation
37

まずはOSのネットワークパラメーターに関するチューニングです。TCP/IPのコネクションに関するパラメーターはミドルウェアでも設定できる場合がありますが、OSが持つパラメーターによるものも多くあります。

AIXを例に挙げて、TCP/IPコネクションに関するパラメーター値を説明します。tcp_keepinitはTCPコネクションの初期タイムアウト値を決めるパラメーターです。つまり、ネットワーク障害などでSYNパケットを投げても、それに対するACKが返ってこないときは、この値に従って接続は破棄されます。AIXではこの値は75秒です。tcp_keepidleは通信を行っていないTCPコネクションに対して確認する時間、tcp_keepintvlは接続の確認のために送るパケットの送信間隔、tcp_keepcntは返事がないときに再度確認のために送るパケットの数をそれぞれ指定するパラメーター値です。AIXでは、これらのデフォルト値より、2時間10分経てば、そのコネクションはOSによって切断されることになります。サーバー側で出したFINに対し、クライアントからFINが帰ってくるまでの間、接続のステータスはFIN_WAIT2になりますが、この状態でいつまで待ち続けるかもパラメーターtcp_finwait2によって決められています。またクライアントからFINを受け取ってコネクションが完全に終了するまでの間、接続のステータスはTIME_WAITになりますが、TIME_WAITで待ち続ける時間もパラメーターtcp_timewaitによって決められています。tcp_timewaitはデフォルトで15秒ですので、コネクションが完全に終了するまで15秒はTIME_WAIT状態でソケットを持ち続けることになります。またこの値を0にすることはできません。

OSが持つパラメーターの調整については、下記のInformation Centerの各オペレーティング・システムの項目やTechnoteを参照ください。

WAS V8.0 Information Center「オペレーティング・システムの調整」

http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/tprf_tuneopsys.html

Technote「【注意事項】DB障害時におけるKeepAliveパラメーター設定のWASサービスへの影響について (WAS-09-048)」

<https://www-304.ibm.com/support/docview.wss?uid=jpn1J1004347>

Designing Web System Infrastructure with WAS V8.0

クライアント⇔Webサーバー間通信

- チューニング・ポイント
 - ◆ KeepAlive
 - ハンドシェイクを繰り返すことによるオーバーヘッドを抑えるためにTCPコネクションを再利用
 - 複数の画像やフレームを含むコンテンツに有効
 - KeepAlive (デフォルト On) ... KeepAlive接続の有無
 - MaxKeepAliveRequests (デフォルト100) ... 1つのKeepAlive接続で受け付けるリクエスト数(画像などの合計数程度)
 - KeepAliveTimeout (デフォルト10) ... 次のリクエストまでに接続を保持する時間(秒)
 - ◆ SSL通信
 - セキュリティ強度やパフォーマンスの観点からSSL通信の要否を検討
 - SSLセッションIDのキャッシュのタイムアウトをSSLV3TimeoutやSSLV2Timeoutディレクティブで設定可能
 - ◆ ファイル圧縮
 - ファイルを圧縮することにより通信帯域の負荷を軽減して通信時間を短縮
 - サーバー負荷になるので圧縮すべきものを見極めが必要、対応Webブラウザには制限あり

各機能の詳細は「パフォーマンス設計-参考-」のP.108～111をご参照ください。

© 2012 IBM Corporation
38

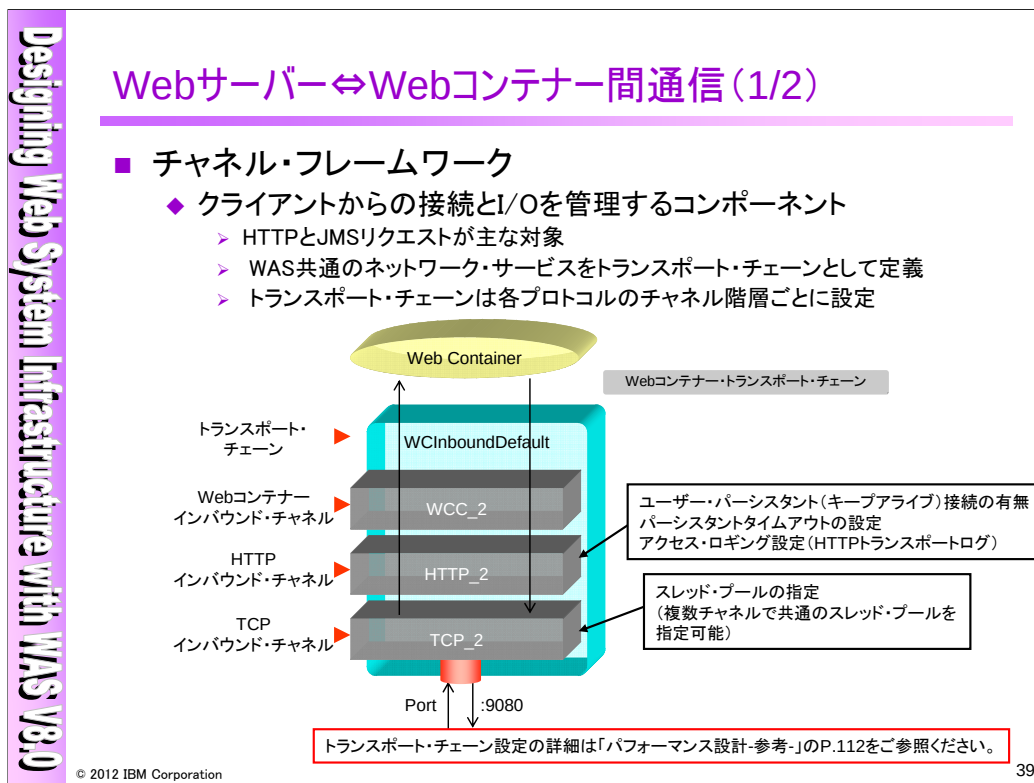
クライアントとWebサーバー間のネットワーク通信に関する設定のうち、主なチューニング・ポイントは、KeepAlive、SSL通信、ファイル圧縮の設定になります。

KeepAlive接続を行うことにより、TCPコネクションを再利用し、毎回ハンドシェイクを繰り返すことによるオーバーヘッドを抑えることが可能です。HTTP/1.1からサポートされています。KeepAliveに関するIHSの設定には、KeepAlive、MaxKeepAliveRequests、KeepAliveTimeoutがあります。MaxKeepAliveRequestsは1つのKeepAlive接続で受け付けるリクエスト数ですので、一画面の画像の合計数程度にしておきます。またKeepAlive接続では、KeepAliveTimeoutで指定された時間だけ次のリクエストが来るまでに接続を確保するため、1、2秒程度の短い値にしておくといふと考えられます。

SSL(Secure Socket Layer)はNetscape Communications Corporationによって開発された認証や暗号化により、盗聴や改ざん、なりすましを防ぎ、データの機密性を保証するプロトコルです。クライアントからSSLアクセスの要求が来ると、サーバーはクライアントとやり取りを行う暗号化アルゴリズムとSSLセッションIDを指定し、クライアントに返すとともにサーバー証明書を送信します。クライアントはそれを受けて暗号化アルゴリズムを宣言し、サーバーに返すことでSSL通信が確立します。またクライアント認証を行うときは、サーバーはクライアントに対しクライアント証明書を要求し、クライアントは適切なクライアント証明書をサーバーに返します。このようにSSLハンドシェイクは通常のHTTP接続よりも接続に負荷がかかるので、セキュリティ強度やパフォーマンスの観点から使用するかどうかを検討します。また、SSLハンドシェイクは通常のHTTP接続よりもさらに接続に負担がかかるので2回目からSSLのセッションIDをキャッシュすることで、パフォーマンスダウンを防いでいますが、SSLV3TimeoutやSSLV2TimeoutディレクティブでこのSSLセッションのタイムアウトを設定することができます。デフォルトでSSLV3Timeoutは120秒、SSLV2Timeoutは40秒ですので、SSLV3も120秒経つとSSLハンドシェイクをやり直します。ディレクティブの有効範囲はSSLV3Timeoutは0-86400秒、SSLV2Timeoutは0-100秒ですので、できる限りセッションIDを再利用し、新規のSSLハンドシェイクによるパフォーマンスダウンを防ぎたい場合には、この値を大きくします。

IHS V2.0からはmod_deflateモジュールによるファイルの圧縮が可能になりました。これにより、通信帯域にかかる負荷を軽減し、通信時間を短縮することが可能です。ただし圧縮処理を行う分だけサーバー側には負荷となりますし、圧縮に対応しているかどうかはクライアントのWebブラウザに依存しますので、圧縮すべきものを見極めて、圧縮すべきものだけを圧縮する必要があります。IHS構成ファイルにて圧縮率を指定することも可能です。

ここで解説した機能の詳細に関しては、「パフォーマンス設計-参考-」のP.108～111をご参照ください。



WebサーバーとWebコンテナー間のネットワーク通信に関する設定を考えると、チャンネル・フレームワークの仕組みを理解することが重要です。

チャンネル・フレームワークは、WebコンテナーやメッセージングなどWebSphereで稼動するコンポーネントに対しネットワークの共通基盤となるものであり、スレッド・プールなどのリソースを複数チャンネルで共有することにより同時接続性を向上させるものです。Webコンテナーのスレッド・プールもこのチャンネル・フレームワークを用いています。具体的には、Webコンテナーの設定は、アプリケーション・サーバーが持つ9080などのポートを保有するトランスポート・チェーンのTCPインバウンド・チャンネルに結び付けられたWebContainerというスレッド・プールに対して行います。チャンネル・フレームワークではネットワーク・サービスをトランスポート・チェーンとして定義しています。トランスポート・チェーンでは個々のI/Oプロトコルごとのチャンネルというものをプロトコル・スタックとして内包します。このようにトランスポート・チェーンをチャンネルごとに分けることで、TCPチャンネルではポートやスレッド・プールの指定、HTTPチャンネルではアクセス・ロギングやパーシスタントのタイムアウト設定などチャンネルごとに個々のレイヤーで設定を行うことが可能になりました。

トランスポート・チェーン設定の詳細に関しては、「パフォーマンス設計-参考-」のP.112をご参照ください。

Designing Web System Infrastructure with WAS V8.0

Webサーバー⇔Webコンテナ間通信(2/2)

- チューニング・ポイント
 - ◆ KeepAlive
 - Webコンテナ・スレッドとクライアントが独立
 - KeepAliveによりスレッドが待たされることなく次のリクエストの処理を実行可能
 - デフォルトで全てのスレッドがKeepAlive有効
 - 非KeepAlive接続の確保やタイムアウトの短縮が不要

© 2012 IBM Corporation

40

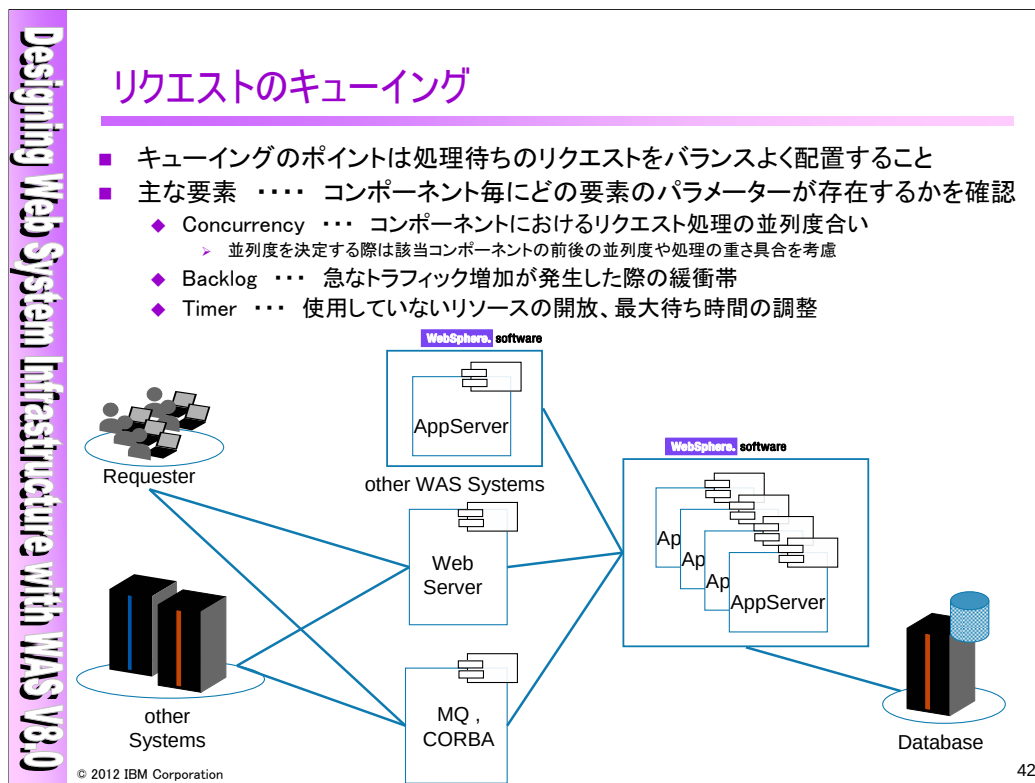
チャンネル・フレームワークの導入により、Webコンテナのスレッドとクライアントとは完全に独立しました。クライアントからのリクエストはTCPインバウンド・チャンネルのキューが一旦受け取り、そこからWebContainerへリクエストが送られます。これにより、クライアントとWebコンテナのスレッドが直接結びつかなくなり、リクエストをより有効に使用されるようになりました。具体的にはKeepAlive接続の切断を待たずに、次のリクエストの処理の実行が可能になりました。これにより、KeepAliveの設定を全スレッド数の90%に設定して、スレッドを新規のリクエストのために確保する必要はなくなり、Webコンテナのスレッドもデフォルトで全てKeepAlive対応となっています。

Designing Web System Infrastructure with WAS V8.0

3. チューニング

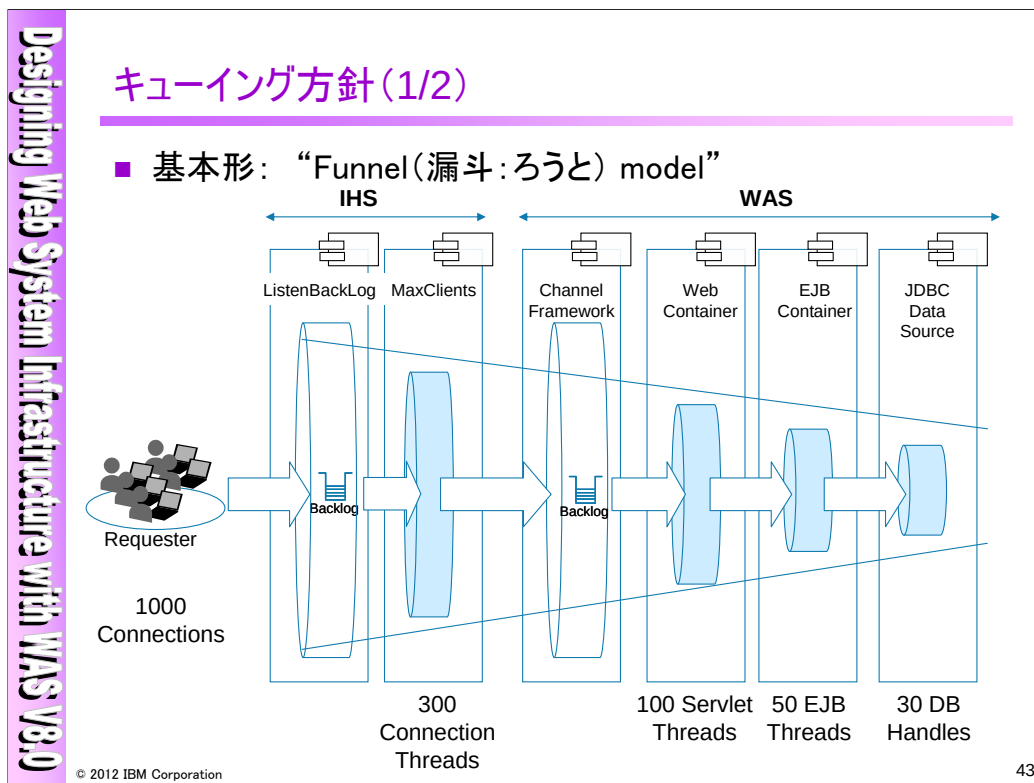
- 3-1 チューニング概要
- 3-2 ネットワーク関連のチューニング
- 3-3 リクエスト・キューイング**
- 3-4 コンポーネント別のチューニング
- 3-5 パフォーマンス・チューニングのポイント

© 2012 IBM Corporation 41

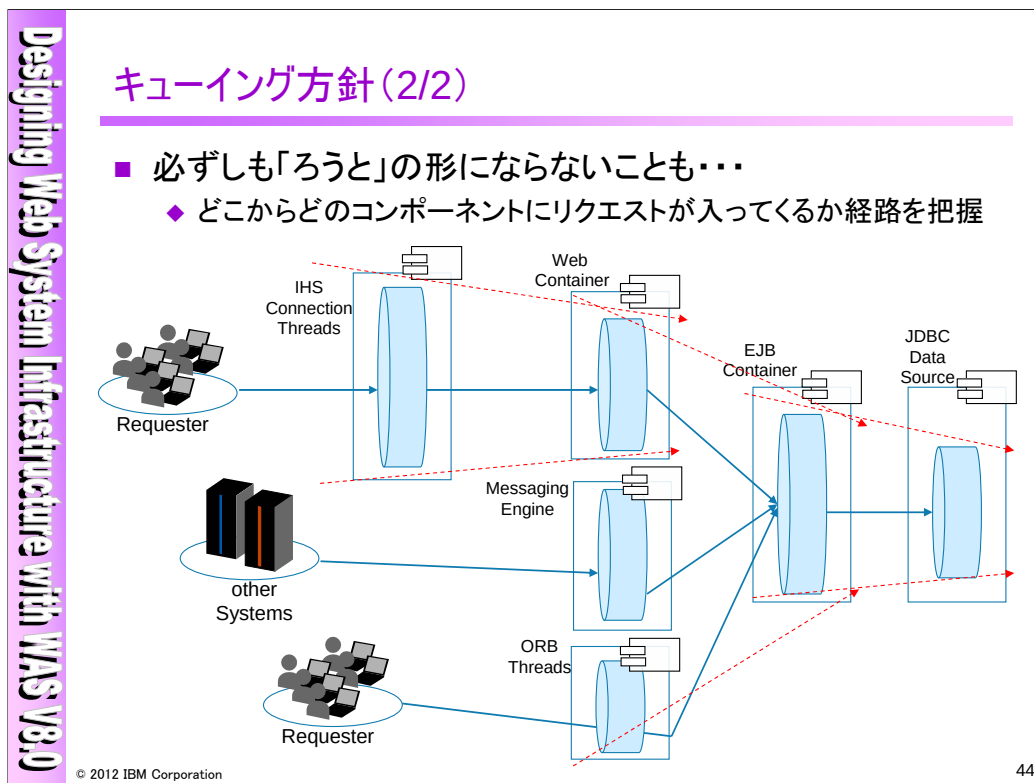


リクエストは通信で使用するプロトコルによって異なる経路をたどってターゲットとなるアプリケーション・サーバーに到達します。想定される最大のトラフィックのすべてを同時並行的に対処できるようなシステム・リソースを常に確保できるようなシステムは通常ありません。そのため、リクエストにはたいいどこかのタイミングで処理待ちが発生します。リクエストのキューイングとは、その「処理待ち」をどのレイヤーでどの程度行なわせるかを決めてハードウェア・リソースの各コンポーネントへの適正な配分を行なうための戦略です。

Concurrencyとは、あるコンポーネントにおいて一度に処理を行なうことのできる並列度を示します。コンポーネントとしてServletやJSPの実行エンジンであるWebコンテナを例にとると、通常のWeb (HTTP) リクエストひとつに対してWebコンテナは処理(worker)スレッドをひとつ割り当てて処理を行ないます。処理スレッドは使いまわしを行なうためにプールされていますので、プールに沢山のスレッドを用意しておけばその分同時に処理できる数が大きくなります(並列度が高くなる)。並列度を高くすると、その分CPUやメモリーなどハードウェアのリソースがそのコンポーネントだけで大量に消費されてしまい、他のコンポーネントの処理性能が低下してしまいます。ハードウェアリソースを共有しているコンポーネント同士の兼ね合いや、コンポーネントから見たリクエスト元とリクエストのディスパッチ先の処理能力などを考慮して並列度を決定します。Backlogは、急なトラフィック増加が発生した際の緩衝帯です。コンポーネント間の接続は常にブロッキング(1:1の紐付け)がなされており、同期処理が行なわれるような状態とは限りません。コンポーネントによってはバックログ(リクエストを入れるキュー)を用意することで非ブロッキング接続を実現しているものもあります。これによりリクエストが一旦キューに格納されることで、リクエストはコンポーネントの並列度に関わらず受け入れを行なえるようになります。そのため、同時処理性能を超えるトラフィックが来た場合もキューが一時的な緩衝帯(バッファ)となってシステムのハングなどを回避できるようになります。Timerは、使用していないリソースの開放や最大待ち時間の調整です。リクエスト・キューイングによるリソースの適切な配分に絞って考えると、タイマー(タイムアウト)の主な役割は使用していない(アイドル状態の)ミドルウェア・リソース(スレッドや接続オブジェクトなど)の開放にあります。ネットワーク通信のチューニングのところでご紹介したKeepAlive接続のタイムアウト戦略も同じ考え方に基づいています。未使用のリソースを開放することで、それに紐づくハードウェア・リソースも開放され、それを他に必要なコンポーネントに配分することでパフォーマンスの向上が望めます。



並列度設定にあたっての考慮点のところでもご紹介しましたが、このチャートにあるように各Webサイトのコンポーネントでの並列度パラメーターを前段から絞り込んでいくと、一般にリソースの有効活用とボトルネックの解消に効果があると言われています。最前列の口を一番大きく開けて、最後尾のバックエンドのDB接続の部分で並列度が最も小さくなる(口が絞り込まれている)ことから、この手法は漏斗モデル(Funnel model)メソッドとも呼ばれています。



リクエストは必ずWebサーバーから入ってくるわけではありません。アプリケーション・サーバーは基幹システムや他のWebシステムからメッセージという形でリクエストを受け付ける場合もありますし、IIOP接続で直接EJBのビジネス・ロジックを要求する場合があります。従って必ずしも前のページで紹介したようなきれいな「漏斗」型をとるとは限りません。しかし、その場合も、基本的にアプリケーションの観点で最も後方に配置されているコンポーネントから前段のコンポーネント(の集合)に向かって徐々に口をあけていくイメージで設定を行なっていくと良い結果が得られる場合が多いです。

Designing Web System Infrastructure with WAS V8.0

3. チューニング

- 3-1 チューニング概要
- 3-2 ネットワーク関連のチューニング
- 3-3 リクエスト・キューイング
- 3-4 コンポーネント別のチューニング**
- 3-5 パフォーマンス・チューニングのポイント

© 2012 IBM Corporation 45

Designing Web System Infrastructure with WAS V8.0

コンポーネント別チューニング

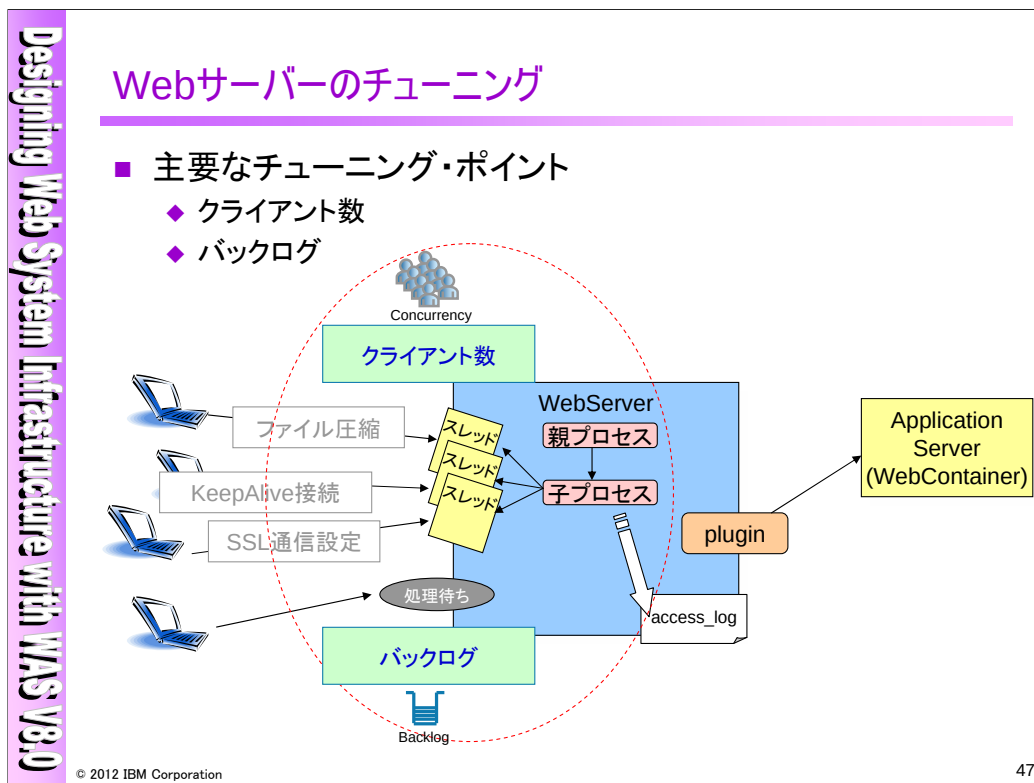
- 主な検討項目
 - ◆ Webサーバー
 - クライアント数
 - バックログ
 - ◆ Webサーバー・プラグインとWebコンテナ
 - Webコンテナのスレッド・プール
 - アプリケーション・サーバーが処理できる接続の最大数
 - 構成ファイルの再ロード間隔
 - セッション・オブジェクト
 - ◆ EJBコンテナ
 - 参照渡し
 - Object Request Broker(ORB)スレッド・プール
 - タイムアウト
 - ◆ JDBCデータソース
 - 接続プール
 - preparedStatementCacheサイズ
 - ◆ メッセージング
 - 接続プール
 - MDBの同時並行数
 - パーシスタント・メッセージ

46

© 2012 IBM Corporation

チューニング項目はコンポーネント毎に異なっており、ここでは、一般的にチューニングの際に検討する項目をご紹介します。

Webサーバーでは、クライアント数やバックログによるチューニングを行います。Webサーバー・プラグインとWebコンテナでは、Webコンテナのスレッド・プールやアプリケーション・サーバーが処理できる接続の最大数、構成ファイルの再ロード間隔、セッション・オブジェクト、によるチューニングを行います。Webコンテナには、Webサービス呼び出しのアウトバウンド接続プールもありますが、Webサービスのセッションで詳細を説明します。EJBコンテナでは、Message Driven Bean (MDB) やObject Request Broker (ORB) によるチューニングを行います。JDBCデータソースでは、接続プールやPrepared Statement Cache によるチューニングを行います。メッセージングでは、接続プールによるチューニングを行います。



Webサーバーの主なチューニング項目は、クライアント数とバックログの設定です。

Designing Web System Infrastructure with WAS V8.0

Webサーバー:クライアント数(1/3)

- AIX、HPUX、Solaris、Linuxはworkerモジュールをサポート
- 複数の子プロセスが各々複数のスレッドでリクエスト処理
- <IfModule worker.c>内で定義
 - ◆ ThreadLimit 25
 - ThreadsPerChildに設定可能な上限値
 - ◆ ServerLimit 64
 - 設定可能なサーバー・プロセスの上限値
 - ◆ StartServers 1
 - 起動時の子プロセス数の初期値
 - ◆ MaxClients 600
 - 同時に処理できるリクエストの最大数
 - ◆ MinSpereThreads 25
 - 待機状態にあるスレッドの最小値
 - ◆ MaxSpareThreads 75
 - 待機状態にあるスレッドの最大値
 - ◆ ThreadsPerChild 25
 - 各プロセスが生成するスレッド数
 - ◆ MaxRequestsPerChild 0
 - 各プロセスが処理できるリクエスト数の最大値

```

graph TD
    A[親プロセス] --> B[子プロセス]
    A --> C[子プロセス]
    A --> D[子プロセス]
    B --> B1[スレッド]
    B --> B2[スレッド]
    B --> B3[スレッド]
    C --> C1[スレッド]
    C --> C2[スレッド]
    C --> C3[スレッド]
    D --> D1[スレッド]
    D --> D2[スレッド]
    D --> D3[スレッド]
  
```

© 2012 IBM Corporation
48

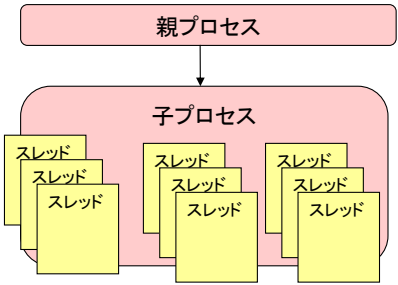
AIX、HPUX、Solaris、Linuxではworkerモジュールにより、複数プロセス・複数スレッドによりリクエスト処理を行うことが可能です。つまり、一つの親プロセスが複数の子プロセスを生成し、それらの子プロセスがスレッドにより複数の同時リクエストを処理できる仕組みになっています。

クライアント数に関するチューニング項目として、上記のようなものがあります。起動時にはStartServers(起動時の子プロセス数) × ThreadsPerChild(1プロセスのスレッド数)のリクエストを処理できる状態にあります。リクエストは最大でMaxClientsで指定されている数だけ同時に処理することが可能です。MaxClientsに相当するリクエストが来たとき、 $\text{MaxClients } 600 \div \text{ThreadsPerChild } 25$ で プロセス数は24個存在することになります。

Designing Web System Infrastructure with WAS V8.0

Webサーバー:クライアント数(2/3)

- Windowsはmpm_winntモジュールをサポート
- 子プロセスは1つのみ
- 子プロセスが複数のスレッドでリクエスト処理
- <IfModule mpm_winnt.c>内で定義
 - ◆ ThreadLimit 600
 - ◆ ThreadsPerChild 600
 - ◆ MaxRequestsPerChild 0
- 同時に処理できるリクエストの最大数はThreadsPerChildで設定



© 2012 IBM Corporation
49

Windowsはmpm_winntモジュールによりリクエスト処理を実行します。Windowsでは親プロセスが一つの子プロセスを作成し、その子プロセスのスレッドにより、リクエストを処理します。従いまして、同時に処理できるリクエストの最大数は、ThreadsPerChildで設定されている値になります。

Designing Web System Infrastructure with WAS V8.0

Webサーバー:クライアント数(3/3)

- $\text{ServerLimit} \times \text{ThreadsPerChild} \geq \text{MaxClients}$
 - ◆ 親プロセスが生成する子プロセスの数と子プロセスの生成する数の積が実際にリクエストを処理する上限
 - ◆ 上記を満たさないときはerror.logに以下のWarningが出力
 - MaxClients150に対し、ThreadsPerChild25、ServerLimit1に設定すると、結局最大で25のリクエストしか扱えない。

WARNING: MaxClients of 150 would require 6 servers, and would exceed the ServerLimit value of 1. Automatically lowering MaxClients to 1. To increase, please see the ServerLimit directive.

- 処理可能な最大リクエスト数を越えたときのerror.log
 - ◆ 使用率に余裕がある場合はWebサーバーのボトルネックが考えられるので増加を検討

[Sun Mar 15 13:13:04 2009] [error] server reached MaxClients setting, consider raising the MaxClients setting

[Sun Mar 15 16:01:14 2009] [warn] Server ran out of threads to serve requests. Consider raising the ThreadsPerChild setting

50

複数プロセス・複数スレッドで動作するWebサーバーでは、生成できる子プロセスの数と1プロセスが生成するスレッドの積が、実際に処理可能なリクエスト数に相当します。ServerLimitが生成される最大の子プロセス数。ThreadsPerChildが各プロセスが作るスレッド数ですので、これらの積が実際に処理可能なリクエスト数です。これらの積がMaxClientsに満たない場合には、当然MaxClientsに相当するリクエストは処理できないことになります。上の例は、MaxClients150に対し、ThreadsPerChild25、ServerLimit1に設定したときにエラーログに出力される警告です。MaxClients150に相当するリクエストを処理するには6プロセスが必要です、という内容の警告が出力されます。

また、MaxClients (WindowsならThreadsPerChild) を越えたリクエストが来るとerror.logに警告として出力されます。CPUなどマシンのスペックに余裕がある場合には、Webサーバーがリクエストを絞りすぎていることになりますので、同時リクエスト数を上げることを検討します。

Designing Web System Infrastructure with WAS V8.0

Webサーバー: バックログ

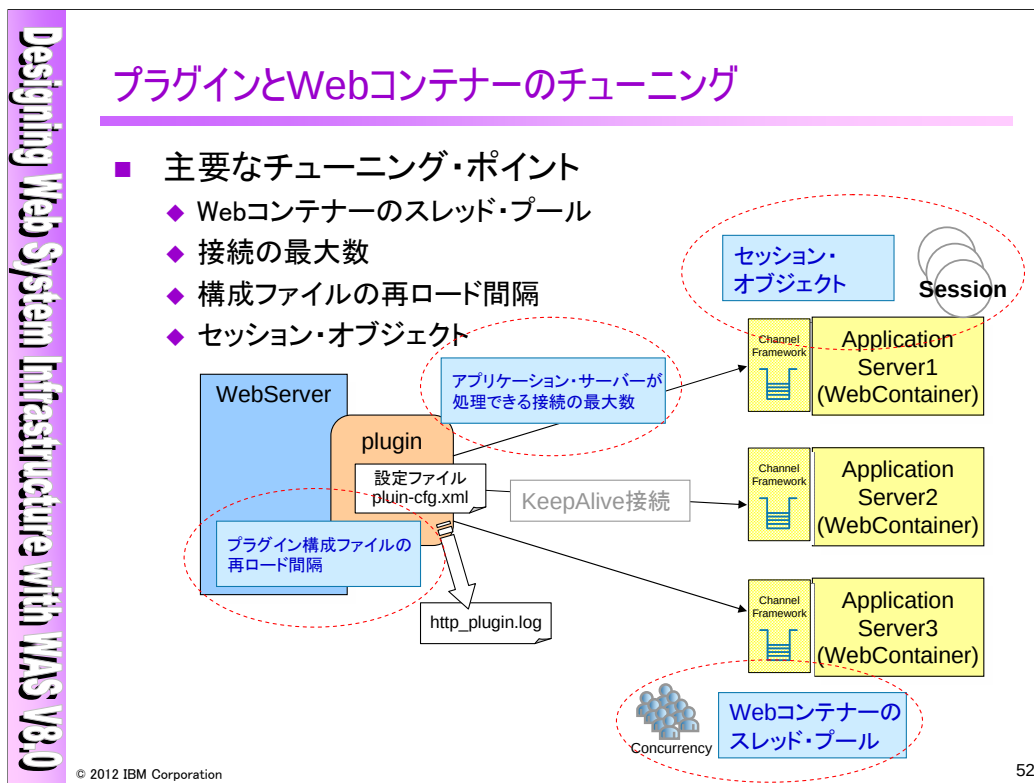
- 同時に処理可能な数を越えるリクエストが来た場合にあふれたリクエストは処理待ちの状態に待機
 - ◆ OSのパラメーターでも設定
 - AIXではsomaxconn（デフォルト1024）
 - ◆ ListenBackLog
 - 処理待ちのキューの数を指定
 - httpd.confには記載がないがデフォルトは511
 - ◆ 処理を待たせてもエラーを返さないようにする場合はListenBackLogを増やす
 - ◆ IHSの処理中とListenBackLogの合計接続数は、netstatコマンドで確認できる

OS側でListenBackLogの値を確認する方法の詳細は「パフォーマンス設計-参考-」のP.114をご参照ください。

© 2012 IBM Corporation
51

同時に処理可能な数を越えるリクエストが来た場合、あふれたリクエストは処理待ちの状態に処理ができるようになるまで待たされます。リクエストがあふれた際、いくらまでのアクセスを処理待ちの状態に保持されるかの指定はOSのパラメーターでも変更することが可能（AIXではso_maxconnパラメーターがこれに相当し、デフォルトでは1024です）ですが、Webサーバーでも処理待ちのキューの数を指定することが可能です。IHSの場合、この値はListenBackLogディレクティブにより指定します。処理を待たせてもクライアントにエラーを返したくない場合には、このListenBackLog（デフォルト511）を指定し、多くのリクエストを処理待ちの状態に待たせておくようにします。この接続待ちの状態は、ESTABLISHEDもしくはSYN_RCVDの状態になります。なおListenBackLogをOSのパラメーター値より高く設定するとOSの持つ値が優先されます。つまり、OSが持つ値とIHSが持つ値で低いほうが優先されます。

OS側でListenBackLogの値を確認する方法の詳細は「パフォーマンス設計-参考-」のP.114をご参照ください。



Webサーバーとアプリケーション・サーバー間すなわちWebサーバー・プラグインとWebコンテナ間のチューニング項目は、アプリケーション・サーバー側で行う設定と、プラグイン側で行う設定との2種類があります。アプリケーション・サーバー側で検討する必要があるのは、Webコンテナのスレッド・プールとKeepAlive接続、セッション・オブジェクトです。プラグイン側で検討する必要があるのは、アプリケーション・サーバーが処理できる接続の最大数、プラグイン構成ファイルの再ロード間隔です。

Designing Web System Infrastructure with WAS V8.0

プラグインとWebコンテナ:スレッド・プール

- スレッド・プール内の設定
 - ◆ 最大サイズ
 - ◆ 最小サイズ
 - ◆ スレッド非活動タイムアウト
 - ◆ 最大スレッド数を超えた割り振りの許可の有無
- TPVによるモニタリングと分析
 - ◆ 「スレッド・プール」カテゴリでWASの各スレッド・プールのデータを取得可能
 - ◆ プール・サイズ、アクティブ・スレッド数、全てのスレッドが使用中である時間の比率を確認
 - すべてのスレッドが使用中である状態が長く続くようであればスレッドの最大数の増加を検討

全てのスレッドが使用中

プール・サイズが最大スレッド数に達しており、その全てがアクティブになっている

© 2012 IBM Corporation
53

Webコンテナのチューニングは、WebContainerというスレッド・プールに対して行います。スレッド・プールの設定項目としては、スレッドの最大・最小サイズ、スレッドの非活動タイムアウト、最大スレッド数を超えた時の割り振り許可の有無があります。

実際のチューニングには、パフォーマンス・テストを実施し、TPVでスレッドの使用率や活動中のスレッド数を確認しながら調整を行います。TPVでスレッド・プールの状況を確認するには「スレッド・プール」のカテゴリを確認します。Webコンテナにおいてスレッドの使用状況を確認し、プール・サイズ、アクティブ・スレッド数、全てのスレッドが使用中である時間の比率（「カスタム」を設定）の3項目を確認します。上記の例では、プール・サイズがそのまま最大スレッド数に張り付いている状態が長く続いているので、スレッドが不足していることが確認できます。このような場合は、最大スレッド数の増加を検討します。

一般的には、スレッドの生成や消滅の負荷を軽減するために、スレッド・プールの最大サイズと最小サイズを同じにした方がパフォーマンスが良いことが多く、WAS V8のデフォルト値も同じ値になっています。

Designing Web System Infrastructure with WAS V8.0

プラグインとWebコンテナー: 接続の最大数

- MaxConnections
 - ◆ プラグインが各サーバーに同時に割り振る最大数
 - IHSの子プロセス単位でカウントされるので注意
 - ◆ デフォルトは「-1」(無制限)
 - ◆ MaxConnectionsに達すると別の割り振り可能なサーバーに割り振り
 - ◆ クラスター内に使用可能なアプリケーション・サーバーが存在しない場合に503 (Service Temporarily Unavailable) がクライアントに返される
 - 待ちには入らず、即時にクライアントに返される

```
<Server ConnectTimeout="10" ExtendedHandshake="false" MaxConnections="10"
Name="mercuryNode01_server1" ServerIOTimeout="0" Wait-forContinue="false">
  <Transport Hostname="mercury" Port="9080" Protocol="http"/>
  <Transport Hostname="mercury" Port="9443" Protocol="https">
    ... (略)
  </Server>
```

プラグインの生成時にプラグイン構成ファイルに反映される。

アプリケーション・サーバーが処理できる接続の最大数

☒ 接続の最大数の使用
接続の最大数 (10) 増減

この場合はTCPインバウンド・チャンネルにはストックされず、即時にクライアントにエラーが返る。

54

プラグイン側でも各サーバーへの接続数を制限することができます。デフォルトは-1に設定されており、プラグイン側では各サーバーに割り振る数は制限していません。例えば、あるクラスターメンバーに対しこの値を10に設定すると、プラグインはそのサーバーに対し10のリクエストを割り振り、それが10に達すると、別のクラスターメンバーに割り振りを行うようになります。そしてどのクラスターメンバーにも割り振りを行えなくなると、プラグインはクライアントに503 (Service Temporary Unavailable) のエラーを返します。この値はプラグイン構成ファイルで設定されていますが、IHSがデプロイメントマネージャーの配下にあれば、管理コンソールからでも設定を行うことが可能です。管理コンソールからアプリケーション・サーバーを選択し、[Webサーバープラグインプロパティ]を選択すると設定画面になります。設定を保管し、プラグインを生成、伝播すると反映されます。ただし、この設定を行った場合、割り振られるものがなくなれば、TCPインバウンド・チャンネルのキューにはストックされず、プラグインから即時にステータスコード503のエラーが返されます。MaxConnectionsが少なすぎるとエラーが出やすくなりますので注意が必要です。

WAS V8.0 によるWebシステム基盤設計ワークショップ

54

Designing Web System Infrastructure with WAS V8.0

プラグインとWebコンテナ：構成ファイルの再ロード間隔

- RefreshInterval
 - ◆ 構成ファイルplugin-cfg.xmlの情報が読み込まれる間隔
 - ◆ デフォルト60秒
 - ◆ 情報の更新から反映までの遅延として許容できる値に設定

Webサーバー選択し、[追加プロパティ]から [プラグイン・プロパティ]選択

plugin-cfg.xml

```
<Config ASDisableNagle="false" AcceptAllContent="false" AppServerPortPreference="HostHeader"
ChunkedResponse="false" FIPSEnable="false" IISDisableNagle="false" IISPluginPriority="High"
IgnoreDNSFailures="false" RefreshInterval="60" ResponseChunkSize="64" VHostMatchingCompat="false">
```

© 2012 IBM Corporation
55

Webサーバー・プラグインは、RefreshIntervalの間隔ごとにプラグイン構成ファイルをロードします。この間隔はデフォルトでは60秒になっています。不必要なロードはプラグインにとっても負荷となりますので、プラグイン構成ファイルの再ロード間隔は情報の更新から反映までの遅延として許容できる値に設定しておくようにします。

Designing Web System Infrastructure with WAS V8.0

プラグインとWebコンテナー: セッション・オブジェクト(1/2)

- セッション管理の設定
 - ◆ メモリー内の最大セッション・カウント(デフォルト: 1000セッション)
 - ◆ セッション・タイムアウト(デフォルト: 30分)
 - ◆ オーバーフローの許可(セッション情報が複数サーバー間で共有されていない場合のみ有効)
- TPVによるモニタリングと分析
 - ◆ 「サーブレット・セッション・マネージャー」カテゴリでセッション・データ取得可能
 - ◆ チューニング
 - 「LiveCount」が非常に大きく、長時間とどまっている場合は、セッション・オブジェクトがヒープに残り続けているので、「ActiveCount」との差が大きいのであれば、セッション・タイムアウト値を小さくする
 - 「NoRoomForNewSessionCount」がカウントされている場合、無効なセッションが作成されている。メモリー内の最大セッション・カウント値を大きくする
 - 分散セッションで、「CacheDiscardCount」がカウントされている場合、セッションのスペース不足により、セッション・オブジェクトが破棄が発生。最大セッション・カウント値を大きくする
 - 「ActivateNonExistSessionCount」がカウントされている場合、タイムアウトにより無効化されたセッションへのアクセスが起きているので、セッション・タイムアウト値を大きくする

ActiveCountに比べ、LiveCountの差が大きい例。(アクセスのないセッションが多く残る)

セッション・オブジェクトの破棄が発生した例

セッション管理の設定の詳細は「パフォーマンス設計-参考-」のP.115～116をご参照ください。

56

セッション・オブジェクトがメモリーの大部分を占めているような場合は、セッション・オブジェクトの設定を見直す必要があります。セッション・オブジェクトの設定は管理コンソールから行うことが可能で、最大セッション・カウントとセッションのタイムアウトを設定します。また、DBによるパーシスタンスやメモリー間での複製を行っていない環境では、「オーバーフローの許可」の有無も設定できます。デフォルトは1000セッション、タイムアウトは30分ですので、メモリーには最大1000セッションが30分残ることになります。

これらの値はTPVでセッション・オブジェクトのサイズや状況を確認しながらチューニングを行います。TPVでセッションの状態を確認するには「サーブレット・セッション・マネージャー」のカテゴリを見ます。上記はセッションのチューニングを行う例を示しています。「LiveCount」が非常に大きく、長時間とどまっている場合には、セッション・オブジェクトがヒープに残り続けているので、「ActiveCount」との差が大きいのであれば、セッション・タイムアウト値の減少を検討します。「NoRoomForNewSessionCount」がカウントされている場合、最大セッション・カウントを超えるセッションが必要にも関わらず、その確保ができていけませんので、メモリー内の最大セッション・カウント値の増加を検討します。分散セッションで「CacheDiscardCount」がカウントされている場合、セッションのスペース不足により、セッション・オブジェクトが破棄が発生していますので、最大セッション・カウント値の増加を検討します。「ActivateNonExistSessionCount」がカウントされている場合、タイムアウトにより無効化されたセッションへのアクセスが起きているので、セッション・タイムアウト値の増加を検討します。

セッション管理の設定の詳細に関しては、「パフォーマンス設計-参考-」のP.115～116をご参照ください。

Designing Web System Infrastructure with WAS V8.0

プラグインとWebコンテナー:セッション・オブジェクト(2/2)

- セッション書き込み(セッションDB/メモリー間複製共に)
 - ◆ 設定
 - 書き込み頻度(serviceメソッド終了、手動、時間基準) デフォルト:10秒
 - 書き込みの内容(更新された属性のみ、全てのセッション属性) デフォルト:更新属性のみ
 - セッション・クリーンアップのスケジュール(時間基準でセッションをクリーンアップ) デフォルト:なし
 - ◆ 検討
 - 書き込み頻度を下げるとパフォーマンスは良くなるが可用性は下がる
- セッション・データベース
 - ◆ 設定
 - 複数行スキーマの使用(デフォルトでは1セッションにつき単一行にデータ書き込み)
 - DB2行サイズ(DB2 UDB Only) (デフォルトは4KB)
 - ◆ 分析
 - 複数行スキーマはセッション・データのサイズが大きい場合(2MB以上は必須)に有効
 - DB2はページ単位で読み込み、書き込みを行う
 - セッション・オブジェクトサイズ≤ページ・サイズとすることでパフォーマンスの向上が見込める

セッション・オブジェクトのサイズが8kBであれば、ページ・サイズも4kBより8kBのほうがパフォーマンスは上がる

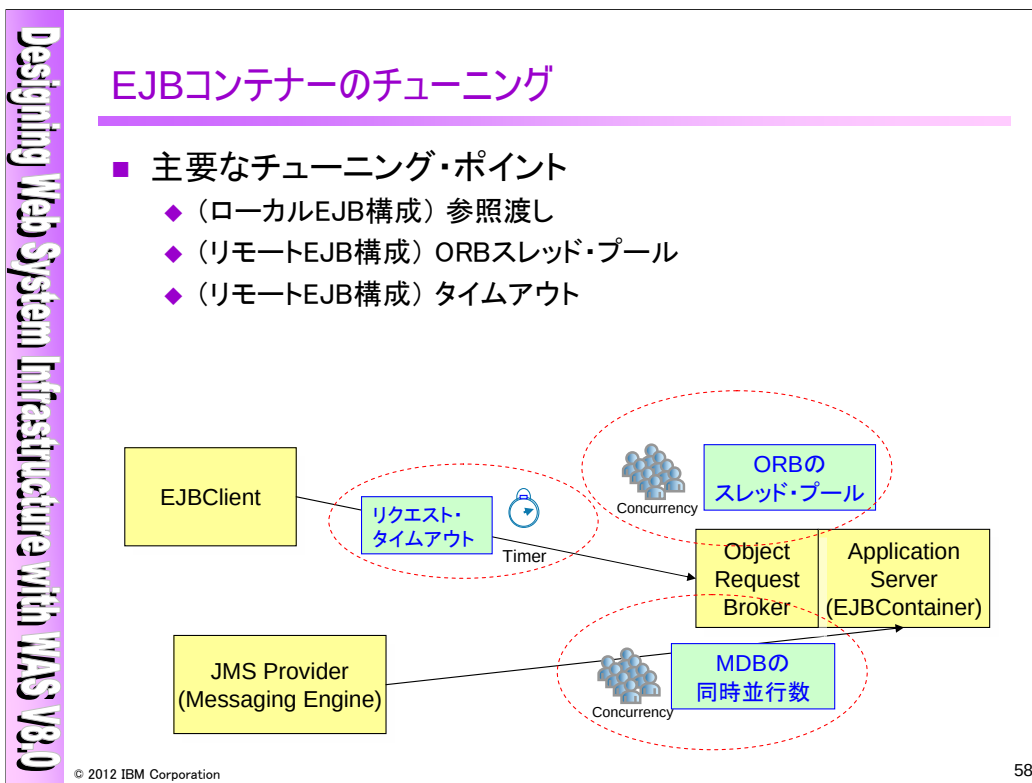
それぞれの設定の詳細は「パフォーマンス設計-参考-」のP.117～118をご参照ください。

© 2012 IBM Corporation
57

セッションパーシスタントを用いる場合、セッションの書き込み頻度もパフォーマンスに影響します。この場合はデータベースに限らずメモリー間コピーでも対象になります。セッションの書き込み頻度、書き込みの内容、セッション・クリーンアップの設定ができます。デフォルトではカスタム設定(書き込み頻度10秒、内容は更新された属性のみ、セッション・クリーンアップはしない)に設定されています。チューニングレベルでは、それらの設定を自動的に設定することが可能です。チューニングレベルを高くすると、書き込み頻度が抑えられるためにパフォーマンスは上がりますが、可用性は下がります。

セッションの書き込み頻度では、セッション・パーシスタンスを行う際に、メモリーやデータベースへと実際にデータの書き込みを行なうタイミングについての設定をします。書き込みの内容では、更新された属性(つまり差分)のみを書き込むか、セッション・データすべてを書き込みなおすかを指定します。セッション・クリーンアップのスケジュールでは、設定周期ごとにメモリーもしくはDBに格納されているオブジェクトの最終更新時刻のチェックを行ない、一定時間アクセスされていないセッションの無効化をします。これをONにする場合は、夜間や毎日定期的にサービスを停止する時間帯がある場合など、サーバーがビジーでない時間にクリーンアップをスケジューリングして無効化します。セッションパーシスタントとしてデータベースを用いる場合、セッション・データベースの設定を管理コンソールから行うことが可能です。デフォルトでは単一行スキーマを使用し、一つのセッション・データに対し単一行を使用しますが、「複数行スキーマを使用する」にチェックを入れると、セッション・データを複数行に分けてデータを書き込みます。セッション・データが小さいときは単一行でかまいませんが、セッション・データが非常に大きいときは複数行にわけて書き込むことでデータのシリアライズ化が抑えられ、パフォーマンスが向上します(ただし、書き込み内容が「更新された属性のみ変更」に設定されていること)。またセッション・データが2MB以上あるときは複数行スキーマが必須です。データベースがDB2 UDBの場合はDB2行サイズが設定できます。DB2はデータベースにつき最低1つのバッファプールをメモリー内に確保し、ページ(デフォルト4kB)単位で読み込みや書き込みを行います。セッション・オブジェクトサイズ≤ページサイズにすることでパフォーマンスを向上させることができます。その単位は4kB、8kB、16kB、32kBの4種です。

それぞれの設定の詳細に関しては、「パフォーマンス設計-参考-」のP.117～118をご参照ください。



EJBコンテナのチューニングについて解説します。主要なチューニング・ポイントとして、参照渡し、ORBスレッド・プール、EJBリクエストのタイムアウト、MDBの同時並行数が挙げられます。EJBクライアントからリモートでEJBにアクセスする際にはORBスレッド・プールの設定が必要です。

Designing Web System Infrastructure with WAS V8.0

EJBコンテナー: (ローカルEJB構成) 参照渡し

- WebコンテナーとEJBコンテナーが同じJVM上にある場合は同一のスレッドで処理
 - ◆ 並列度は前段のコンポーネント(つまりWebコンテナー)で調整
- EJBクライアントとEJBが同一のクラス・ローダーから読み込まれる場合(つまり同一EAR内にパッケージされている場合)は参照渡しを行なうことでパフォーマンスを向上
 - ◆ アプリケーションサーバーを選択し、[コンテナー・サービス]→[ORBサービス]で「参照による受け渡し」にチェック(次ページを参照)
 - ◆ もしくはシステム・プロパティにてcom.ibm.CORBA.iiop.noLocalCopies=true

© 2012 IBM Corporation

EJBクライアントとEJBコンテナーが同一アプリケーションサーバー上に配置されている場合には、同一アプリケーションサーバー上で処理されますので、この場合にはEJBコンテナーが受けるリクエスト数について設定することはできません。なお、EJBクライアントとEJBコンテナーは、同一アプリケーション・サーバー上に配置したほうがパフォーマンスは良くなります。

Designing Web System Infrastructure with WAS V8.0

EJBコンテナ：(リモートEJB構成)ORBスレッド・プール

- アプリケーション・サーバーの[コンテナ・サービス]→[ORBサービス]
 - ◆ 接続キャッシュの設定やPass by reference (参照渡し) の設定
- リモートのEJBクライアントからアクセスがある場合にはスレッド・プールを設定
 - ◆ 設定画面へはORBサービス画面または追加プロパティの[スレッド・プール]
 - ◆ スレッド・プールのデフォルトは最大50スレッド、最小10スレッド
 - ◆ 非活動タイムアウトを使用してアイドル状態にあるスレッドを開放

リモートのEJBクライアントからのアクセスに対しては最大・最小スレッド数を設定することが可能です。設定は管理コンソールからアプリケーション・サーバー選択し、[コンテナサービス]→[ORBサービス]→[スレッド・プール]の画面から最大・最小サイズを設定します。デフォルトは最大サイズが50スレッド、最小サイズが10スレッドです。

Designing Web System Infrastructure with WAS V8.0

EJBコンテナー: (リモートEJB構成)タイムアウト

- アプリケーション・サーバーの[サーバー・インフラストラクチャー]→[プロセス定義]→[Java仮想マシン]の汎用JVM引数で定義
- com.ibm.CORBA.requestTimeout
 - ◆ メソッド・コール中のネットワーク障害、処理滞留によりタイムアウト (org.omg.CORBA.NO_RESPONSE)とみなすまでの時間
 - ◆ デフォルト180秒
- com.ibm.websphere.wlm.unusable.interval
 - ◆ ダウンとみなされたサーバーに対し、再度リクエストを割り振るまでの間隔
 - ◆ デフォルト300秒

初期ヒープ・サイズ

最大ヒープ・サイズ

☐ HProf の実行

HProf 引数

☐ デバッグ・モード

デバッグ引数

☐ [xv.suspend=n,address=7777]

汎用 JVM 引数
-Dcom.ibm.websphere.wlm.unusable.interval=300

実行可能 JAR ファイル名

© 2012 IBM Corporation
61

EJB WLMのチューニング・ポイントとしては、requestTimeoutとintervalの設定があります。com.ibm.CORBA.requestTimeoutは、メソッド・コール中のネットワーク障害、処理滞留によりタイムアウト(org.omg.CORBA.NO_RESPONSE)とみなす時間です。com.ibm.websphere.wlm.unusable.intervalは、EJBコンテナーがダウンと判断された際に再度割り振りを行う間隔をそれぞれ指定します。これらの設定はアプリケーション・サーバーの[Java仮想マシン]画面の汎用JVM引数の欄で指定します。

Designing Web System Infrastructure with WAS V8.0

JDBCデータソースのチューニング

- 主要なチューニング・ポイント
 - ◆ JDBCドライバー・タイプの選択
 - 通常はType4を選択(詳細はデータベース接続設計のセッション参照)
 - ◆ 接続プール
 - ◆ PreparedStatementCacheサイズ

Concurency

Application Server

Connection Pool

Connection object

DB

Timer

接続プールの最大・最小接続数

PreparedStatement Cacheサイズ

・接続タイムアウト
・未使用タイムアウト
・経過タイムアウト

© 2012 IBM Corporation

62

アプリケーション・サーバーからデータベースへの接続に関するチューニングについて解説します。ここでのチューニング・ポイントは、接続プールの最大・最小接続数、PreparedStatementCacheサイズです。

JDBCデータソース: 接続プール

- データソースの接続数の設定
 - ◆ 通常時は最小接続数で対応してピーク時に最大接続数で対応するように設定
 - ◆ プール内のコネクション・オブジェクトの使用率が100%の状態が続き、接続待ちや、接続タイムアウトによるエラーとなったリクエストが多い場合は接続プールの最大接続数の増加を検討
- 接続タイムアウトの設定
 - ◆ 接続要求が待機する時間(デフォルト180秒)
 - ◆ タイムアウトになるとConnectionWaitTimeoutExceptionが発生
- 未使用タイムアウト、経過時間タイムアウト
 - ◆ アイドル状態にある接続を開放
 - 詳細は、「データベース接続設計」を参照

© 2012 IBM Corporation

63

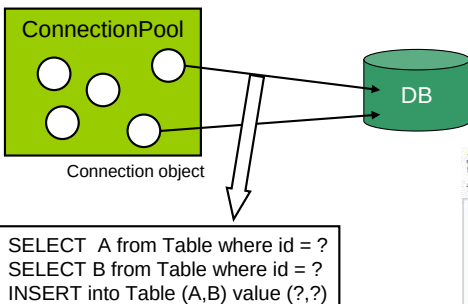
接続プールの設定は、データベースに接続するデータソースから設定します。管理コンソールからJDBCプロバイダーとデータソースを選択し[接続プール・プロパティ]を選択、一般プロパティの最大接続数、最小接続数で指定します。実際のチューニングは、TPVを用いて、コネクション・オブジェクトの使用率、接続待ち数、接続タイムアウトによるエラー数などを確認しながらチューニングを行います。また接続数を常に同じにしておくと、使用されない接続が確立されたままになる可能性がありますので、通常時は最小接続数で対応し、ピーク時に最大接続数で対応するように設定します。接続タイムアウトはdataSource.getConnection()を発行してから、接続オブジェクトを取得するまでの待ち時間になります。この時間だけ待っても接続プールに空きがなければ、ConnectionWaitTimeoutExceptionが返されます。デフォルトは180秒です。

接続プールのチューニングを行う例をご紹介します。「PercentMaxed」が100%の状態が続き、「WaitThreadCount」が多い場合は、接続プール内のオブジェクトが不足し、接続待ちが多く発生していますので、最大接続数の増加を検討します。さらに接続待ちのリクエストが接続タイムアウトによりエラー(ConnectionWaitTimeoutException)となると、「FaultCount」が大きくなります。この場合は最大接続数の増加とともに接続タイムアウトの増加も検討してください。「ManagedConnectionCount」の変動が多い場合は、物理接続の生成と破棄が繰り返されていますので、プールに存在するオブジェクトが少ないことが考えられます。未使用タイムアウト値、または最小接続数の増加を検討します。「PrepStmtCacheDiscardCount」が多い場合は、PreparedStatementのキャッシュが廃棄されており、ステートメント・キャッシュ・サイズが不足しているので増加を検討します。

Designing Web System Infrastructure with WAS V8.0

JDBCデータソース:PreparedStatementCacheサイズ

- 1コネクションごとにPreparedStatementをキャッシュ
 - ◆ JDBC→データソースを選択し、[WebSphere Application Serverデータ・ソース・プロパティ]の「ステートメント・キャッシュ・サイズ」で設定
 - ◆ PreparedStatementの破棄数が多いようであれば増加を検討
 - ◆ デフォルトでは10ステートメント

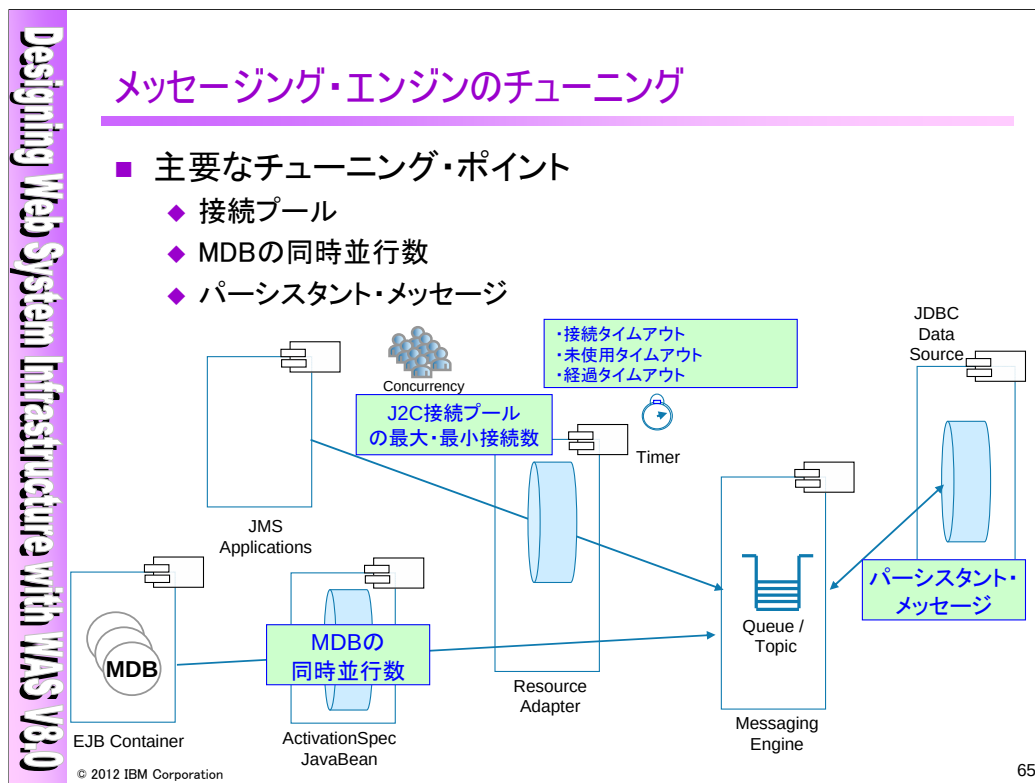




大きすぎるキャッシュ・サイズはヒープに悪影響を及ぼす可能性もあるので注意

© 2012 IBM Corporation
64

PreparedStatementを利用することで、Prepare済みのステートメントを再利用することができます。アプリケーションからprepareを要求すると、ステートメントキャッシュに同じSQL文が存在するか検索し、存在した場合はキャッシュ内のPreparedStatementを再利用します。キャッシュしておくステートメントの数もデータソースから設定が可能です。デフォルトは1コネクションあたり10ステートメントになっていますが、実際にはTPVを用いて、破棄されるPreparedStatementの数が多いようであれば増加を検討してください。



メッセージング・エンジンへの接続に関するチューニングについて解説します。ここでのチューニング・ポイントは、メッセージをPUTするために使用される接続プールの最大・最小接続数のチューニングとメッセージをGETするために使用されるMDBの同時並行数、メッセージのパーシスタンス方式を検討する必要があります。

Designing Web System Infrastructure with WAS V8.0

メッセージング・エンジン: 接続プール

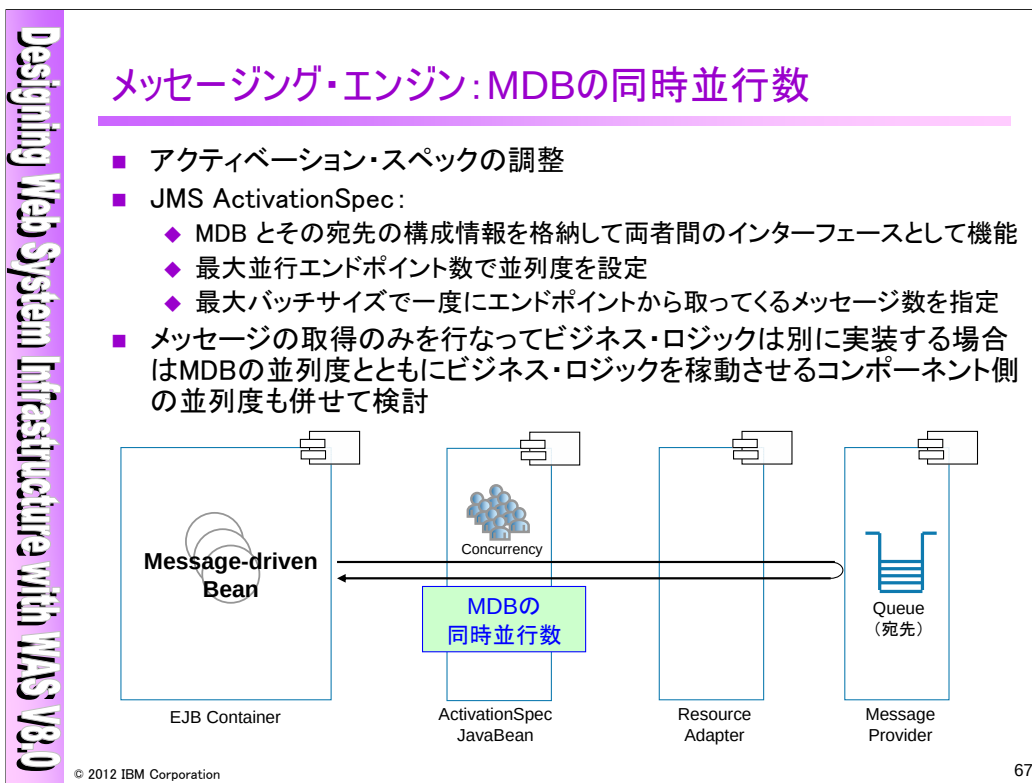
- メッセージング・エンジン (SIBus) への接続をプールすることで接続処理のオーバーヘッドを緩和
- JMSクライアントからの接続要求ではJ2C接続プールを使用
 - ◆ 使用する接続ファクトリーの追加プロパティ[接続プール]を選択

[接続ファクトリー](#) > [JMSConnectionFactory](#) > 接続プール
 このページを使用して、アプリケーションのパフォーマンスに影響を与える可能性のある、接続管理タスクのタイミングに影響するプロパティを設定します。デフォルト値は慎重に考慮してください。アプリケーション要件によっては、これらの値の変更が必要な場合があります。

構成
 一般プロパティ
 有効範囲: cells:mecell:clusters:cluster01
 接続タイムアウト: 180 秒間
 最大接続数: 10 接続
 最小接続数: 1 接続
 リープ時間: 180 秒間
 未使用タイムアウト: 1800 秒間
 経過時間タイムアウト: 0 秒間
 パージ・ポリシー: EntirePool
 追加プロパティ
 ■ [拡張接続プール: プロパティ](#)
 ■ [接続プール: カスタム・プロパティ](#)
 設定パラメーターはDBの接続プールと同じ
 適用 OK リセット 取り消し

66

JMSクライアントからの接続要求では、J2C接続プールを使用します。接続プールのパラメーターはDBの接続プールと同様です。



Message-driven Bean (MDB) もリモートORBと同様に、EJBコンテナの外側で並列度の調整を行います。なお、アクティベーション・スペック (ActivationSpec JavaBean) とはレガシーやEnterprise Information System (EIS) から、アプリケーション・サーバーのEJBを呼び出す為の仕組みとしてJava Connector Architecture Specification (JCA) 1.5から採用されたものです。最大並行エンドポイント数や最大バッチサイズなどのアクティベーション・スペック設定は、使用するJMSプロバイダーの追加プロパティから行なうことができます。

Designing Web System Infrastructure with WAS V8.0

メッセージング・エンジン: パーシスタント・メッセージ

- メッセージの永続化(メッセージ・ストア)にDBを利用する場合はJDBCの接続プールのチューニングが必要
 - ◆ パージポリシーをデフォルト(EntirePool)から変更しない
 - メッセージング・エンジンの停止と共に全ての接続の開放が目的
- メッセージ・ストアとしてファイル・ストアを選択するという方法もあり
 - ◆ パフォーマンスが良くなる可能性もあるがディスク性能に依存

© 2012 IBM Corporation

68

WAS V6.1からメッセージング・エンジンの利用に際してDBの準備は必要条件ではなくなり、ファイルベースの永続化も選択ができるようになっています。DBMSを使用する必要がなくなったため、運用効率や構成の観点からファイル・ストアは便利な面があります。ただし、ファイル・ストアの方がパフォーマンスが良くなる可能性はありますが、永続化を行なうディスクの性能にもよるため、一概には言えません。永続化にDBMSを利用する(データ・ストアの場合)、JDBCの設定が別途必要になります。

Designing Web System Infrastructure with WAS V8.0

3. チューニング

- 3-1 チューニング概要
- 3-2 ネットワーク関連のチューニング
- 3-3 リクエスト・キューイング
- 3-4 コンポーネント別のチューニング
- 3-5 パフォーマンス・チューニングのポイント**

© 2012 IBM Corporation 69

Designing Web System Infrastructure with WAS V8.0

WAS V8.0パフォーマンス・チューニングのポイント

- 事前確認
 - ◆ コンポーネント毎にどの要素のパラメーターが存在するかを確認
- パフォーマンス・テスト時のチューニング
 - ◆ システム全体を考慮
 - ◆ 1度に 1つのパラメーターを変更
 - ◆ チューニングを始める前にフォールバック手順を確認
 - ◆ ハードウェアをアップグレードする前に問題を理解

ボトルネック発見のアプローチは「パフォーマンス設計-参考-」のP.120～130をご参照ください。

70

© 2012 IBM Corporation

ここで、WAS V8.0におけるパフォーマンス・チューニングのポイントを解説します。

まず、事前確認として、各コンポーネントでさまざまなチューニング項目が存在するため、それらを整理しておくことで、チューニングを効率的に実施することができます。

パフォーマンス・テスト時のチューニングとして、いくつかのポイントをご紹介します。まず、システム全体を考慮する必要があります。他に影響を与えることなく1つのパラメーターまたはシステムのチューニングを行うことはできないので、チューニング前に、システム全体がどのような影響を受けるかを考慮します。また、チューニングの際は、1度に複数のパフォーマンス・チューニング・パラメーターを変更すると、それぞれの変更の効果を評価する事ができなくなります。1つのパラメーターを調整して1つの分野を改善すると、考慮に入れていなかった別の分野が影響を受けることもあるので、1度には1つのパラメーターを変更するようにします。また、チューニングを行うと予想外のパフォーマンス結果が生じることがあるため、パフォーマンスが低下した場合は、そのチューニングを元に戻して別のチューニングを行う必要がありますので、事前にフォールバック手順を確認しておきます。また、最終手段としてハードウェアのアップグレードも検討しますが、その前に問題を理解することが重要です。それは、例えば、ディスク装置を追加しても、それを活用するだけの処理能力やチャネルがない場合があるためです。

ボトルネック発見のアプローチに関しては、「パフォーマンス設計-参考-」のP.120～130をご参照ください。

Designing Web System Infrastructure with WAS V8.0

4. パフォーマンスを向上する追加コンポーネント

© 2012 IBM Corporation

71

Designing Web System Infrastructure with WAS V8.0

高パフォーマンスを考慮したアーキテクチャ設計

- 高パフォーマンス実現のための検討項目
 - ◆ キャッシング
 - ◆ 負荷分散と流量制御
 - ◆ 非同期処理/並列処理

WAS/IHSだけではなく、
他のコンポーネントも組み合わせてのパフォーマンス設計

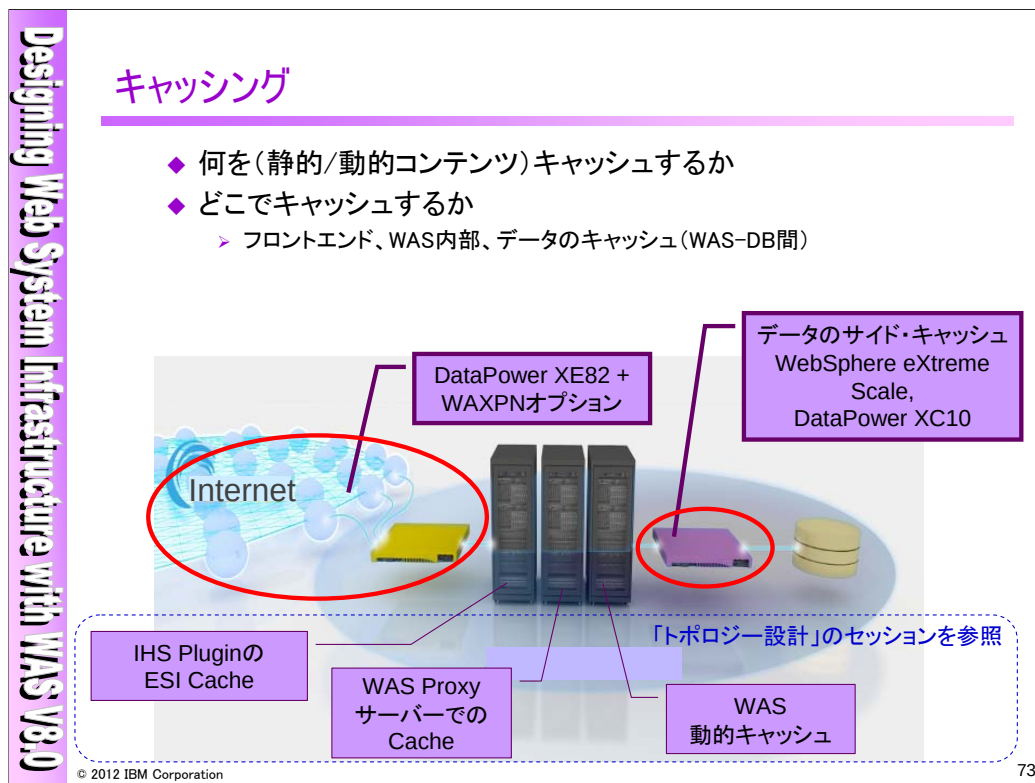
© 2012 IBM Corporation

72

ここでは、WAS単体ではなく、他のコンポーネントと連携することによるパフォーマンスの向上を紹介します。

WAS内部の処理を高速に処理するだけではなく、WAS内部で同一の処理を繰り返さないようにするキャッシングや、優先度に応じた流量制御、非同期処理や並列処理を組み合わせることにより、エンドユーザーが体感するレスポンスを向上させる方法を紹介します。

WAS/IHSだけでなく、他のコンポーネントも組み合わせてのパフォーマンス設計となりますので、システム構成にも影響しますのでご注意ください。



73

トポロジー設計のセッションで紹介したように、WASやWAS Proxyサーバー、WebサーバーのPluginでWASの動的コンテンツをキャッシングすることができます。

一般的に、キャッシングは、より前段でキャッシュすることで、よりパフォーマンスの向上が実現できますし、後段では、より細かい粒度でキャッシュを行うことができます。

キャッシングを検討するときには、何をどこでキャッシュするか、また、動的コンテンツをキャッシュする場合には、どのように、キャッシュされたデータを削除するかを考える必要があります。

Designing Web System Infrastructure with WAS V8.0

インターネットのエッジにキャッシュ・コンテンツを配信

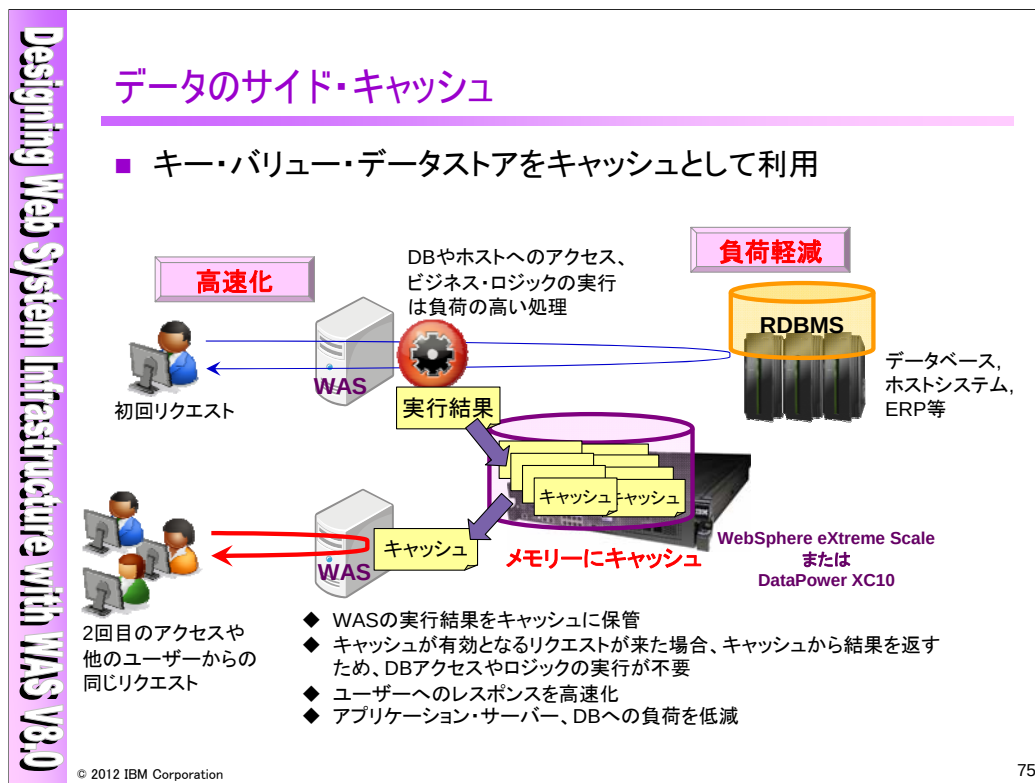
- WAXPN (WebSphere Application Accelerator for Public Networks)
 - ◆ WAXPNはアカマイ・サービスを含んだDataPowerとの連携オプション
 - ◆ インターネットのフロントエンドに企業内のキャッシュ・コンテンツを配信
 - DataPowerとの連携により、動的コンテンツのキャッシュも可能に
 - ◆ アカマイ・ネットワークは、キャッシュだけでなく、インターネットのルートの最適化などにより、オリジナル(WASなど)からのコンテンツの配信も高速化

The diagram illustrates the WAXPN architecture. On the left, a group of users is shown. Red arrows labeled 'Cache Hit !!' point from the users to a cluster of Akamai edge servers. A purple arrow labeled 'キャッシュ・データをAkamaiエッジ・サーバーに配信' (Distribute cache data to Akamai edge servers) points from the Akamai servers to the DataPower XE82 appliance. The DataPower XE82 is connected to a WAS (WebSphere Application Server) represented by purple cubes. The entire setup is labeled 'WAXPNオプション' (WAXPN option). The Akamai logo is visible within the edge server cluster.

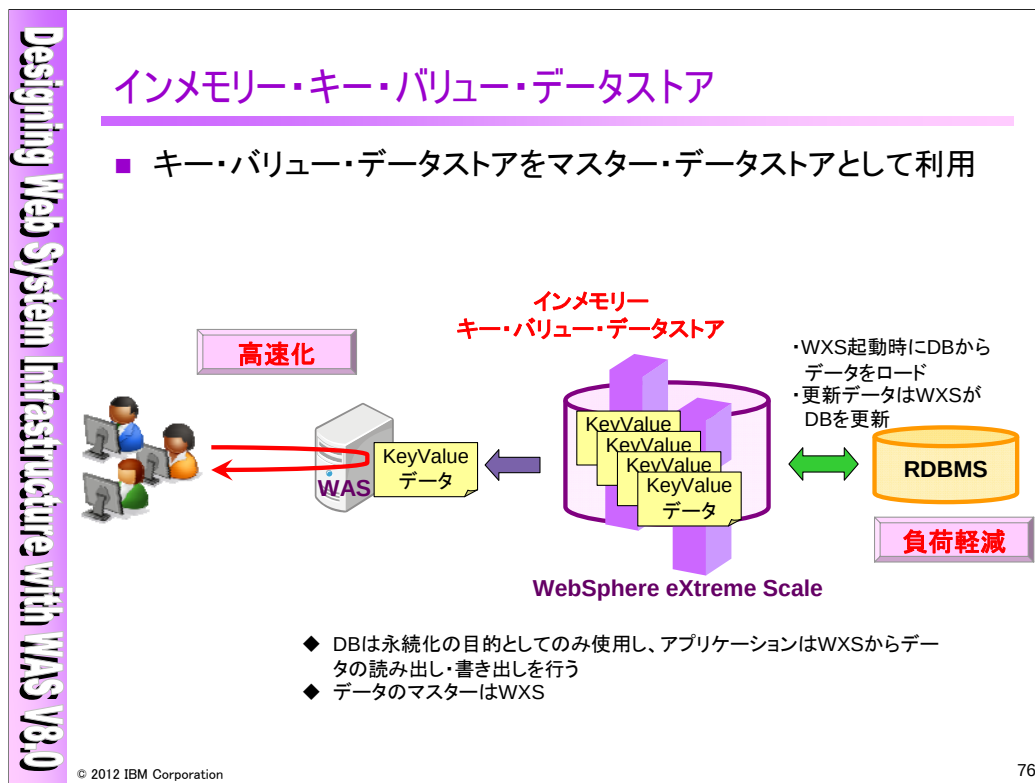
© 2012 IBM Corporation

74

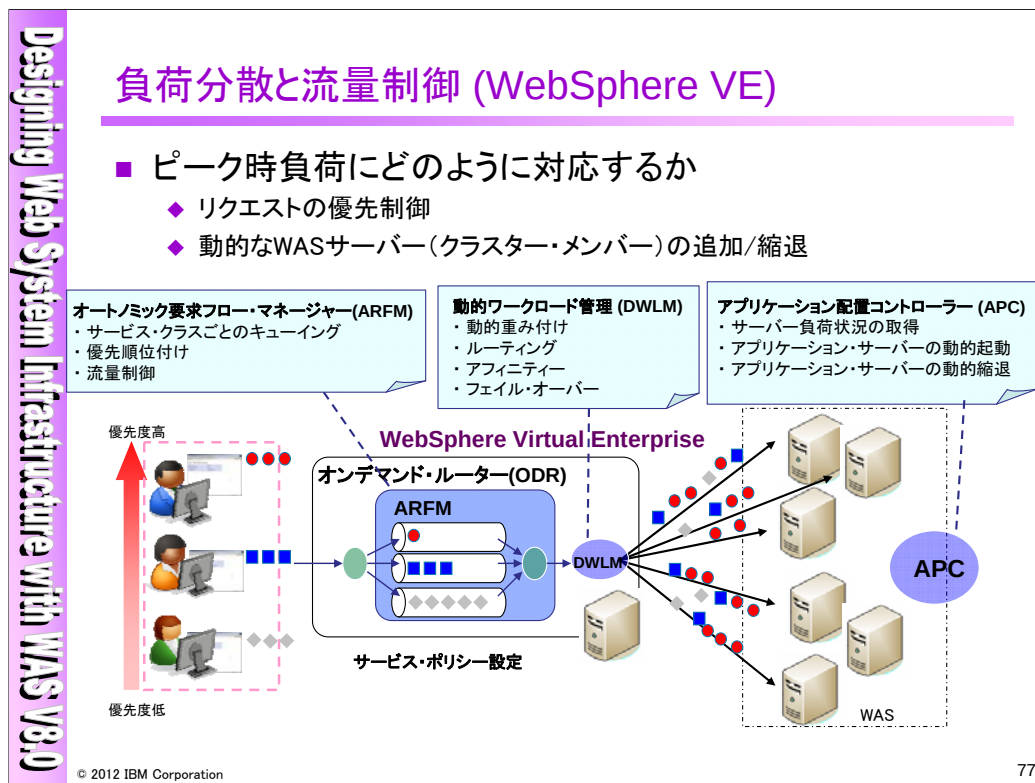
DataPower XE82は、Edge Applianceと呼ばれ、HTTP(s)の処理や負荷分散に特化した、DMZに配置されることを意図したアプライアンス製品です。DataPower XE82とWAXPNオプションを使用することで、インターネット上にオーバーレイ・ネットワークを構築するアカマイ・サービスを使用することによるWASなどのアプリケーション・サーバーから提供されるコンテンツの配信を高速化することができます。高速化のメカニズムの一つが、キャッシュ・コンテンツを、インターネット上に配置されているアカマイ・エッジ・サーバーへの配信です。これにより、エンド・ユーザーからのアクセスが、キャッシュにヒットした場合には、アカマイ・エッジ・サーバーから直接応答が返り、レスポンス・タイムが向上します。



WebSphere eXtreme Scale (WXS)は、分散インメモリー・キー・バリュー・データストアを実装した製品です。また、DataPower XC10は、WXSのアプライアンス版です。どちらの製品を使用しても、データベースやホストシステムなどデータの取得に時間やリソースが必要な場合に、その結果をインメモリー・キー・バリュー・データストアにキャッシュすることで、パフォーマンスを向上することができます。WXSやXC10を使用することで、個々のWASのアプリケーション・サーバーごとに動的キャッシュで管理するのと比較して、クラスター全体でキャッシュ・データを一元管理することができます。



WebSphere eXtreme Scale (WXS)を使用した場合には、DBからデータを取得するのではなく、WXSをマスターのデータストアとして使用することもできます。WXSは起動時にDBからデータをロードし、WXSのデータに追加/更新/削除が行われた場合には、その変更を同期または非同期でDBに反映することができます。この構成ではすべてのデータがメモリー上に配置されていますので、DBからデータを取得するのと比較して高速に応答を返すことができます。また、一般的なDBがデータ量の増加に対応するためには、スケール・アップで対応する必要があるのに対して、WXSでは、サーバーの追加によるスケール・アウトによりデータ量の増加に対応することができ、比較的少ないコストでデータ量の増加に対応することができます。



ピーク時の負荷への対応方法にもいろいろな考え方ができます。ピーク時にも十分なサーバー・リソースを準備するという考え方もありますが、リクエストの優先度を考慮して、優先度の高いリクエストを優先的に応答し、優先度の低いリクエストについては、少し待たせて応答を遅らせて対応するという考え方もあります。

WebSphere Virtual Enterprise (WVE)はWASの拡張製品です。WVEを使用することで、URLなどで優先度を定義し、優先度ごとにキューイングを行うことで、優先度に応じたリクエストの負荷分散を行うことができます。また、WVEでは、リクエストの割り振り先のリソース状況をモニタリングすることで、高負荷なクラスター・メンバーの数を動的に増加させ、負荷の低いクラスター・メンバーの数を減少させることで、サーバー・リソースを有効に活用して、応答をかせすこともできます。

Designing Web System Infrastructure with WAS V8.0

非同期処理/並列処理

- 非同期処理
 - ◆ 長時間かかる処理を別スレッドで処理し、エンドユーザーに処理完了前に応答
 - WAS内部で別スレッドで実行 (Common-J WorkManager, TimerManager)
 - WASメッセージングやWMQを使用した非同期処理
- 並列処理
 - ◆ 1リクエストを複数スレッドで並列処理することによる応答の高速化
 - WAS内部で別スレッドで実行 (CommonJ WorkManager)
 - オンライン処理: WXSのGridAgentを使用したMapReduce処理
 - Batch (長時間) 処理: WCGの平行レル・ジョブ

WXS: WebSphere eXtreme Scale
WCG: WebSphere Compute Grid

© 2012 IBM Corporation

78

非同期処理や並列処理を組み合わせることにより、エンドユーザーへの応答時間を向上させることができます。

長時間かかる処理を別スレッドで処理し、エンドユーザーには処理完了前に応答を返す方法として、WAS内部で別スレッドで処理を実行するプログラミング・モデルとして、WebSphere拡張プログラミング・モデルであるCommon-JのWorkManager (作業マネージャー) やTimerManagerを利用することができます。また、WASのメッセージング (SIBus) やWebSphere MQを使用することにより、異なるWASプロセスで処理を実行することもできます。

1リクエストを複数スレッドで並列処理することによる応答を高速化する方法として、非同期処理と同様にCommon-J WorkManagerを使用することで、別スレッドを起動して並列に処理を実行し、処理完了後に各スレッドの処理結果を取得、マージすることができます。ただ、Common-J WorkManagerを使用した並列処理では、同一WASプロセスでの複数スレッドでの処理になります。WebSphere eXtreme Scale (WXS)やWebSphere Compute Grid (WCG)を使用することで、リクエストを受け付けたWASプロセスとは異なるプロセスやホストでの並列処理を実現することができます。

Designing Web System Infrastructure with WAS V8.0

WXSでのMapReduce処理

- 並列分散データ処理: DataGrid Agent
 - ◆ アプリケーション・ロジックをデータ上で実行する
 - 真にリアル・タイムの応答性能を求められるシステムではネットワークにアクセスする時間も回避したい
 - データをアプリケーションに持ってくるのではなく、「アプリケーションをデータに持っていく」「データ上で、アプリケーションを実行する」
 - ◆ Map Reduce/パターンによる大規模データ並行処理

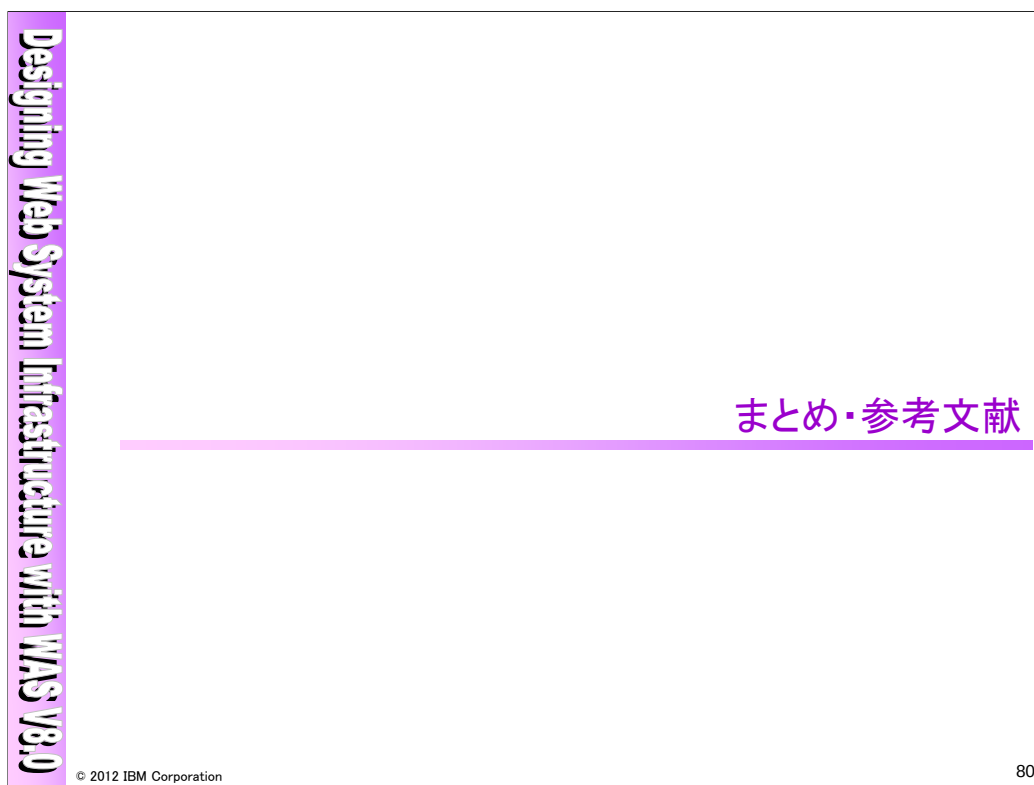
WebSphere eXtreme Scale

© 2012 IBM Corporation

79

インメモリー・キー・バリュー・データストアであるWebSphere eXtreme Scale (WXS)を使用して、MapReduce処理を実行することができます。通常、WXSのデータにアクセスする場合には、ObjectMapインターフェースのgetやputメソッドを使用して、データの取得や更新を行います。

WXSのDataGrid Agent (MapGridAgentまたはReduceGridAgentインターフェース) のAPIを使用することで、WASなどのWXSのクライアントにデータを持ってくるのではなく、WXSのデータ上で処理を並列に実行することができます。また、処理した結果をクライアント側でマージすることができます。



Designing Web System Infrastructure with WAS V8.0

まとめ

- パフォーマンス設計概説
 - ◆ パフォーマンス設計の基本
- データの取得
 - ◆ 取得すべきデータと目標値の明確化
 - レスポンスタイム、スループット、システム・リソース
 - ◆ データ取得方法の検討
 - OS、IHS、WAS、ITCAM
- チューニング
 - ◆ 主要なチューニング・ポイント
 - ネットワーク関連、リクエスト・キューイング、コンポーネント別、JVM
 - ◆ パフォーマンス・チューニングの基本
- パフォーマンスを向上する追加コンポーネント
 - ◆ キャッシング、負荷分散と流量制御、非同期処理/並列処理によるパフォーマンス向上

© 2012 IBM Corporation
81

当セッションで説明した内容をまとめます。

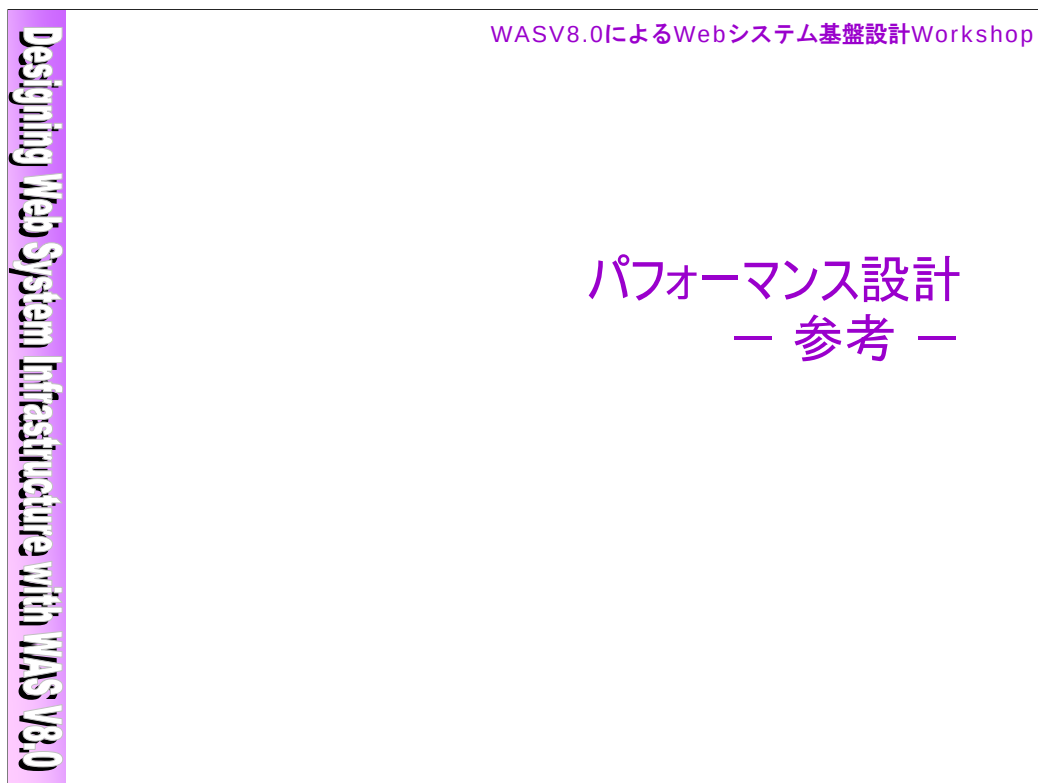
パフォーマンス設計概説では、当セッションの前提知識となる、一般的なパフォーマンスやボトルネック、パフォーマンス・テストの基本について説明しました。

データの取得に関しては、まずは取得すべきデータと目標値を明確にすることが重要であり、取得すべきデータとして、レスポンスタイムやスループット、システム・リソースなどがあるということを説明しました。また、取得すべきデータが明確になったら、どのような方法あるいはツールでそのデータを取得するのかを検討する必要があり、それにはOSの機能やIHS/WASの機能などさまざまな方法があるということを説明しました。

チューニングに関しては、主要なチューニング・ポイントとして、ネットワーク関連のチューニング、リクエスト・キューイングのチューニング、コンポーネント別のチューニング、JVMのチューニングを挙げました。そして、パフォーマンス・チューニングを実施する際のポイントについても説明しました。

参考文献

- Information Center
 - ◆ WebSphere Application Server V8.0 Information Center
 - <http://publib.boulder.ibm.com/infocenter/wasinfo/v8r0/index.jsp>
- ワークショップ資料
 - ◆ WebSphere Application Server V8 アナウンスメント・ワークショップ
 - http://www.ibm.com/developerworks/jp/websphere/library/was/was8_ws/
 - ◆ WebSphere Application Server V7.0によるWebシステム基盤設計ワークショップ資料
 - http://www.ibm.com/developerworks/jp/websphere/library/was/was7_guide/index.html
- ガイド
 - ◆ IBM HTTP Server 7.0 ガイド
 - http://www.ibm.com/developerworks/jp/websphere/library/was/ihs7_guide/index.html
- DeveloperWorks記事
 - ◆ Case study: Tuning WebSphere Application Server V7 and V8 for performance
 - http://www.ibm.com/developerworks/websphere/techjournal/0909_blythe/0909_blythe.html

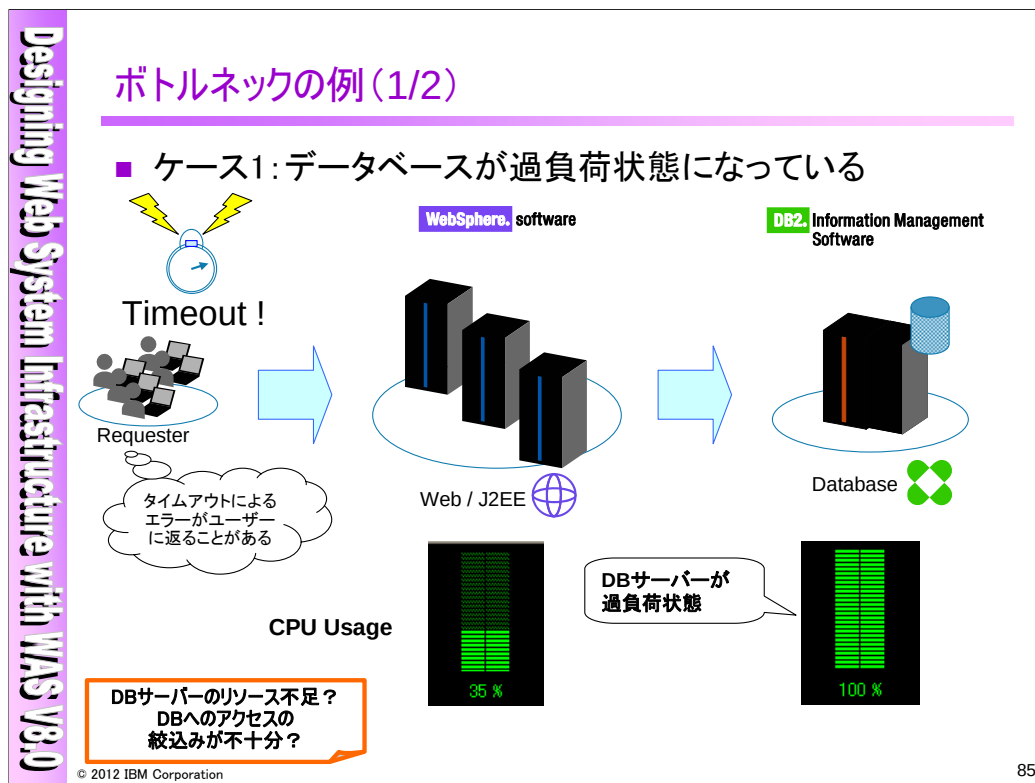


Designing Web System Infrastructure with WAS V8.0

1. パフォーマンス設計概説

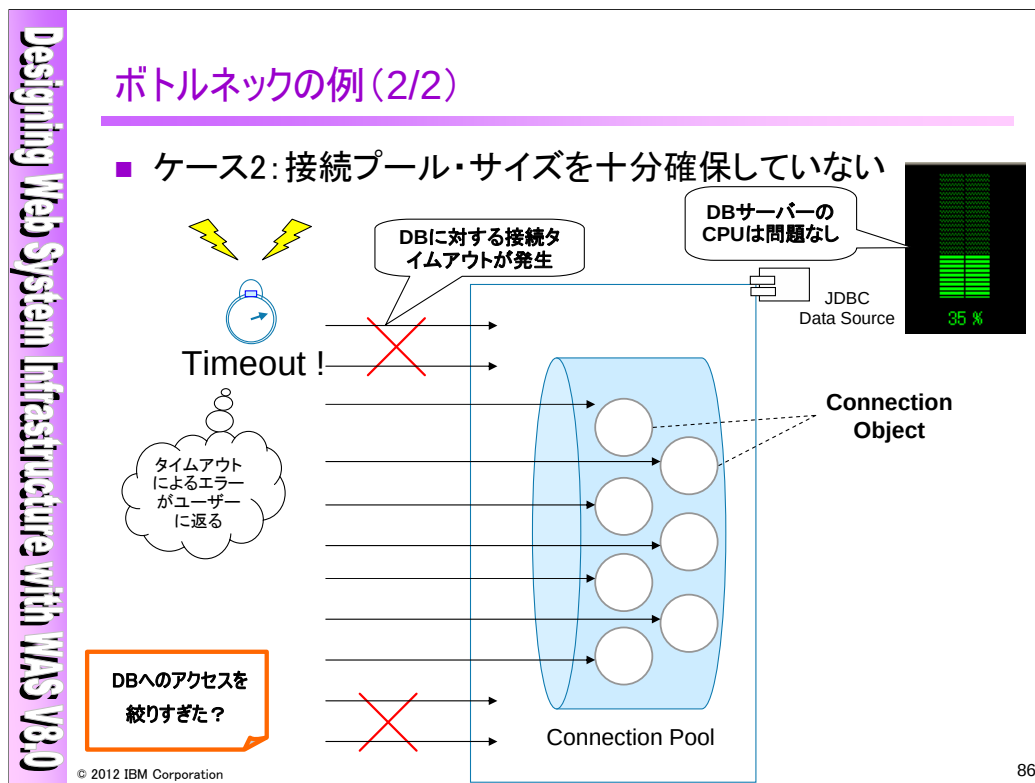
© 2012 IBM Corporation

84



85

ボトルネックの例の1ケース目で、ある特定のサーバー/コンポーネントに負荷が集中してしまう例です。チャートではデータベースが過負荷状態に陥ったためレスポンスが返せなくなってしまい、結果としてクライアント側でタイムアウトとなってしまっています。一般にボトルネックがリクエスト処理の負荷に依存している場合は、設定由来の場合と異なり、まず何よりも先にアプリケーション側での改善を検討します。アプリケーション側での対策をとった上で、ミドルウェア側の対策を検討します。この例の場合はデータベース側のリソースの増強を行なうか、もしくはデータベースから見たクライアントにあたる、前段のアプリケーション・サーバーでリクエストの絞込みを行なって単位時間当たりのデータベース要求を減らすことを検討します。



ボトルネックの例の2ケース目で、あるコンポーネント(チャートではDBの接続プール)のところで必要以上にリクエストを絞り込んでいる例です。DB処理待ちを余儀なくされるリクエストがコンポーネントの前で待たされ、タイムアウトになってしまうというケースです。プール・サイズを拡大するだけのハードウェア・リソースがある場合はプール・サイズの拡張を検討することになります。

Designing Web System Infrastructure with WAS V8.0

2. データの取得

- 2-1 パフォーマンス・データ取得の概要**
- 2-2 OSによるパフォーマンス・データ取得
- 2-3 IHSによるパフォーマンス・データ取得
- 2-4 WASによるパフォーマンス・データ取得
- 2-5 パフォーマンス・データ取得のポイント

© 2012 IBM Corporation

87

Designing Web System Infrastructure with WAS V8.0	負荷ツールの紹介 (1/2)		
		IBM Rational Performance Tester	HP LoadRunner
	有償？無償？	有償	有償
	仮想ユーザー	製品とは別に仮想ユーザーを使うためのライセンスが必要	製品とは別に仮想ユーザーを使うためのライセンスが必要
	複数台構成	○ (1台のLocalに対し複数のAgentを配置可能)	○ (1台のControllerに対し複数のAgentを配置)
	スクリプト	GUI、VUの2種の独自スクリプト	Cが標準のスクリプト。その他VisualBasic、VBScript、JavaScript
	スクリプトの変更	手動、GUIによる変更	手動、GUIによる変更
	複数シナリオ実行	○	○
	SSL	○	○
	Cookie	○	○
	入力データのパラメーター化	○ (データプールの使用)	○ (データファイルの作成)
	HTTPヘッダ調整	スクリプトの編集により可能	スクリプトの編集、GUIにより可能。スクリプト作成時に特定ヘッダーだけ取得することも可能
	レポート	テストログやパフォーマンス・レポート、コマンドデータなど多彩なテストレポート。さらにクライアント・マシン、サーバー・マシンのシステム・リソース・データも取得可能	仮想ユーザーの状態やテストのサマリー、エラー、パフォーマンスとレスポンスタイム、Webリソースの情報、Webコンポーネントごとの情報、サーバー・マシンのシステム・リソースなど非常に多岐なデータを取得
	レポートの出力	独自フォーマット。CSVファイルとしてエクスポート可能	htmlやcsv、xls、Word文書の作成も可能
	開発元	IBM Rational	日本ヒューレット・パッカード

© 2012 IBM Corporation

88

ここでは、有償で利用できる負荷ツールをご紹介します。

一般的に、有償版の方が機能が豊富にあり、さまざまなテストケースに対応できます。また、有償版ではテストを効率的に行なう上での鍵ともいえる結果の整理を効率よく行なう仕組みも豊富に備えています。ただし、高機能ゆえに、使いこなすためにはそれ相応の製品スキルが必要となります。有償版のツールを使う場合には、テストツール要員の教育、もしくは外部からテストツール専門の要員・サービスを確保することによるコストも見積もりに入れておく必要があります。

Designing Web System Infrastructure with WAS V8.0

負荷ツールの紹介 (2/2)

Apache JMeter	
有償？無償？	無償
仮想ユーザー	ライセンスは不要
複数台構成	○(1台のControllerに対し複数のAgentを配置可能)
スクリプト	GUI操作によるテスト計画の作成のみ
スクリプトの変更	N/A
複数シナリオ実行	○
SSL	○
Cookie	○
入力データのパラメーター化	○(データファイルの作成)
HTTPヘッダ調整	スクリプトの編集、GUIにより可能。スクリプト作成時に特定ヘッダだけ取得することも可能
レポート	クライアント側のレポートのみ。サーバ側のデータは取得不可能
レポートの出力	テキスト形式。XML、もしくはCSVファイルとしてエクスポート
開発元	Apache Foundation

© 2012 IBM Corporation
89

ここでは、無償で利用できる負荷ツールをご紹介します。

一般的には、ここでご紹介したツールに限らず、無償で利用可能な負荷ツールはどれも同様の機能を保有しています。Apache JMeterや、今回ご紹介はしていませんがEclipse TPTPはWeb上の文章が公式・非公式に多数存在しています。そのため、これらのツールは、プロジェクトで初めて負荷テストツールを利用するような場合、利用するにあたり敷居が比較的低いツールになると考えられます。

ここでご紹介したJMeterは以下のサイトからダウンロードすることができます。

<http://jakarta.apache.org/jmeter/>

Designing Web System Infrastructure with WAS V8.0

2. データの取得

- 2-1 パフォーマンス・データ取得の概要
- 2-2 OSによるパフォーマンス・データ取得**
- 2-3 IHSによるパフォーマンス・データ取得
- 2-4 WASによるパフォーマンス・データ取得
- 2-5 パフォーマンス・データ取得のポイント

© 2012 IBM Corporation

90

Designing Web System Infrastructure with WAS V8.0

vmstatコマンド

- vmstatコマンド
 - ◆ CPUやメモリーの使用状況の変化を一定時間間隔で表示

10秒間隔で利用状況をレポート

-tオプションで時刻を表示

vmstat -t 10 の出力 (AIX)

1回目は起動時からの平均

指定間隔で平均を取る

#vmstat -t 10																	
kthr		memory		page				faults				cpu					
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa	hr mi se
1	1	115652	95387	0	0	0	0	0	0	159	1693	127	1	3	95	1	14:18:24
0	0	115662	95377	0	0	0	0	0	0	153	1275	107	1	1	98	0	14:18:34
0	0	115660	95379	0	0	0	0	0	0	152	1819	107	1	3	96	0	14:18:44
2	0	115659	95380	0	0	0	0	0	0	162	2779	130	2	5	89	4	14:18:54
1	0	122849	88190	0	0	0	0	0	0	155	3968	104	83	6	10	0	14:19:04
1	0	133614	77425	0	0	0	0	0	0	155	7772	107	92	8	0	0	14:19:14
1	0	143252	67786	0	0	0	0	0	0	156	4619	114	92	7	0	0	14:19:24
1	0	146578	64460	0	0	0	0	0	0	155	2350	102	96	4	0	0	14:19:34

© 2012 IBM Corporation

91

CPU使用率やメモリー使用状況を順次確認するコマンドとしてAIXやLinuxではvmstatがあります。こちらはAIXで実行したときの例です。例では-tオプションで時間を表示させ、秒間隔(10秒)を指定し、指定された秒間隔でデータを表示させています。また、1回目のデータはサーバー起動時からの平均を示しています。

Designing Web System Infrastructure with WAS V8.0

vmstatコマンドで取得できる内容

- AIXとLinuxでは取得できる情報が一部異なるので注意
 - ◆ AIX
 - オプションで時刻を表示。表示間隔と回数を指定
 - kthr: 実行可能(r)、待機待ち(b)のカーネル・スレッド数
 - メモリー: アクティブな仮想ページ(avm)、空きメモリーページ(fre)
 - ページ: ページャーの入出力リスト(re)、ページイン数(pi)、ページアウト数(po)、フリーページ数(fr)、ページ置換アルゴリズムによりスキャンされたページ数(sr)、クロックサイクル数(cy)、
 - フォルト: デバイスの割り込み数(in)、システムコールの数(sy)、コンテキスト切替数(cs)
 - CPU: ユーザー・モードのCPU使用率(us)、システム・モードのCPU使用率(sy)、アイドル状態のCPU使用割合(id)、ローカルディスク入出力が保留中のCPUの割合(wa)
 - ◆ Linux
 - 表示間隔と回数を指定
 - Procs: 実行待ち状態のプロセス数(r)、割り込み不可能なスリープ状態にあるプロセス数(b)、
 - Memory: 使用されている仮想メモリー量(swpd)、空きメモリー量(free)、バッファとして使われているメモリー量(buff)、ページ・キャッシュとして使われているメモリー量(cache)
 - Swap: ディスクからスワップインされたメモリー量(si)、ディスクにスワップアウトされたメモリー量(so)
 - IO: ブロック・デバイスに送られたブロック(bi)、ブロック・デバイスから受け取ったブロック(bo)
 - System: 毎秒での割り込み数(in)、毎秒のコンテキスト・スイッチ数(cs)
 - CPU: ユーザー・モードのCPU使用率(us)、システム・モードのCPU使用率(sy)、アイドル状態のCPU使用割合(id)、入出力が待機中のCPUの割合(wa)

© 2012 IBM Corporation
92

vmstatコマンドはAIXでもLinuxでも使用できますが、取得情報は一部異なりますのでご注意ください。

Designing Web System Infrastructure with WAS V8.0

複数CPUマシンの各CPU使用率の確認

■ sarコマンドを用いて各CPUごとの使用率を確認可能

sar -P ALL 2 3 の出力 (AIX)

```
# sar -P ALL 2 3
AIX aixsmphost 2 5 00049FDF4D01 02/22/04
```

17:30:50	cpu	%usr	%sys	%wio	%idle
17:30:52	0	8	92	0	0
	1	0	4	0	96
	2	0	1	0	99
	3	0	0	0	100
-	2	24	0	74	
17:30:54	0	12	88	0	0
	1	0	3	0	97
	2	0	1	0	99
	3	0	0	0	100
-	3	23	0	74	
17:30:56	0	11	89	0	0
	1	0	3	0	97
	2	0	0	0	100
	3	0	0	0	100
-	3	23	0	74	
Average	0	10	90	0	0
	1	0	4	0	96
	2	0	1	0	99
	3	0	0	0	100
-	3	24	0	74	

各CPUごとの使用率

上記4CPUの平均

vmstatで表示されるのは4CPUの平均になる

kthr		memory		page		faults		cpu	
r	b	avm	fre	re	pi	po	fr	sr	cy
in	sy	cs	us	sy	id	wa			
1	1	255733	15930	0	0	0	0	0	476
49437	27	2	24	74	0				
1	1	255733	15930	0	0	0	0	0	473
48923	31	3	23	74	0				
1	1	255733	15930	0	0	0	0	0	466
49383	27	3	23	74	0				

表示回数を指定すると、最後に各CPUの平均を表示

© 2012 IBM Corporation

93

sar -Pコマンドは、指定されたプロセッサの統計情報を報告します。ALLを指定すると、個々のプロセッサごとの統計情報と全プロセッサの平均が報告されますので、複数CPUマシンの場合に有効です。なお、通常のvmstatで表示されているデータは全CPUの平均値になっています。

Designing Web System Infrastructure with WAS V8.0

svmonコマンド

- svmonコマンド (AIX Only)
 - ◆ Pid(プロセスID)、Command(コマンド)、Inuse(実メモリー内のページ数)、Pin(ページアウトされないページ数)、Pgsp(ページング・スペースのページ数)、Virtual(仮想アドレスのページ数)、64-bit(プロセスが64ビットか否か)、Mthrd(マルチスレッドか否か)、Lpage(大規模ページがあるか否か)
 - ◆ 4KB単位で表記

svmon -P <process id> の出力 (AIX)

プロセスIDを指定

使用しているメモリーの情報
(4KB/ページ)

```
# svmon -P 22556
```

Pid	Command	Inuse	Pin	Pgsp	Virtual	64-bit	Mthrd	LPage
22556	java	61244	4065	0	50404	N	Y	N

Vsid	Esid	Type	Description	LPage	Inuse	Pin	Pgsp	Virtual
21010	4	work	working storage	-	19718	0	0	19718
3affd	3	work	working storage	-	12247	0	0	12247
32019	d	work	text or shared-lib code seg	-	11985	0	0	11985
0	0	work	kernel seg	-	6052	3998	0	6052
3c5de	-	pers	/dev/hd2:315563	-	1389	0	-	-

© 2012 IBM Corporation
94

svmonは-PオプションでプロセスIDを指定します。svmonコマンドにより、そのプロセスが用いているメモリーの統計とセグメントに関する情報を見ることが可能です。情報はすべて4kB単位で表示されており、実メモリー量を算出するには4kBを乗じる必要があります。

Designing Web System Infrastructure with WAS V8.0

nmon performance

- AIX /Linuxのパフォーマンス分析ツール
- 以下から自由にダウンロード可能
 - ◆ http://www.ibm.com/developerworks/eserver/articles/analyze_aix/
- 1画面にCPU使用率、メモリー使用量、ネットワークI/O、ディスクI/Oの情報などを出力
- 分析結果をファイルに出力させることも可能
 - ◆ `./nmon -f -s 10 -c 20` (10秒ごとに20回データを出力)
 - ◆ `<host名>_<date>_<time>.nmon`というファイル名で保存される

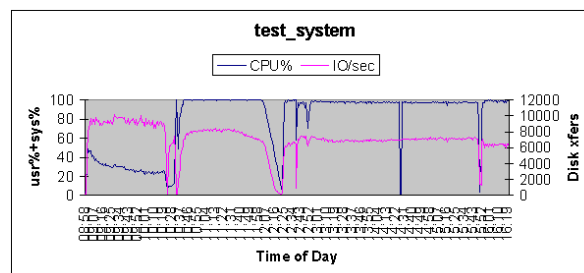
95



AIXやLinuxのパフォーマンス分析ツールとしてnmonがあります。nmonは自由にダウンロードしてマシンに配置するだけで用いることが可能です。起動にはnmonコマンドを実行します。nmonでは、CPU使用率やメモリー使用量、ネットワークI/O、ディスクI/Oといったマシンリソースの情報を1画面で表示させることが可能です。また、分析結果を<host名>_<date>_<time>.nmonというファイルに出力することも可能です。さらにnmonファイルはnmon Analyzerを用いてEXCEL形式にすることもできます。

nmon Analyzer

- nmonの出力をグラフ化するツール
- 以下から自由にダウンロード可能
 - ◆ http://www.ibm.com/developerworks/aix/library/au-nmon_analyser/
- 利用方法
 1. nmonにてパフォーマンス・データ取得
 2. データファイルをPCに転送
 3. nmon Analyserを実行し、「Analyse nmon data」ボタンを押してマクロを実行
 4. xlsファイルで出力
- ◆ 出力例



© 2012 IBM Corporation

96

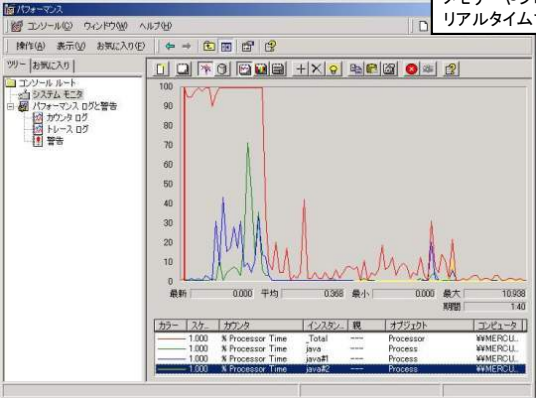
nmon Analyserはnmonの出力をグラフ化するツールで、自由にダウンロードして用いることが可能です。nmonにて取得したパフォーマンス・データをsortコマンドを用いてソート(-A はASCIIの順序配列でソート)します。それらのファイルをPCに転送し、nmon Analyserを実行すると、xlsファイルでパフォーマンス・データがグラフ化されて表示されます。上は横軸を時間にとってCPUの使用率とI/Oを示したグラフの出力例です。

Designing Web System Infrastructure with WAS V8.0

Windowsにおけるパフォーマンス・モニター

- パフォーマンス・モニター: perfmon
 - ◆ システム・モニター
 - メモリーやプロセッサ、ネットワークなどの利用状況のリアルタイム・データを収集
 - グラフやヒストグラム、レポート形式で表示
 - ◆ パフォーマンス・ログと警告
 - データを記録するためのログを構成
 - 閾値による警告の設定
 - カウンター・ログはCSV形式で出力することも可能

メモリーやプロセッサの利用状況をリアルタイムで表示



カラー	スケール	カウンタ	インスタンス	親	オブジェクト	プロパティ
1.000	% Processor Time	Total	Processor		% Processor Time	
1.000	% Processor Time	java	Process		% Processor Time	
1.000	% Processor Time	java2	Process		% Processor Time	

© 2012 IBM Corporation

97

Windowsにはパフォーマンス・モニタリング・ツールとして、perfmonがあります。このツールは、メモリーやプロセッサ、ネットワークの利用状況をリアルタイムで収集して表示する「システムモニター」と記録をログとして保管したり決められた閾値により警告を出す「パフォーマンス・ログと警告」の2つから構成されています。「パフォーマンス・ログと警告」のカウンター・ログは、CSV形式で記録し表計算ソフトで開くことも可能です。

Designing Web System Infrastructure with WAS V8.0

2. データの取得

- 2-1 パフォーマンス・データ取得の概要
- 2-2 OSによるパフォーマンス・データ取得
- 2-3 IHSによるパフォーマンス・データ取得
- 2-4 WASによるパフォーマンス・データ取得**
- 2-5 パフォーマンス・データ取得のポイント

© 2012 IBM Corporation

98

Designing Web System Infrastructure with WAS V8.0

要求メトリックの設定方法

- 管理コンソール左メニューの[モニターおよびチューニング]→[要求メトリック]
- デフォルトはOFF
- どのコンポーネントのデータを取得するかを設定

要求メトリックの使用の有無
(デフォルトはOFF)

フィルターの設定
フィルターをONにしたもののみ出力される

対象コンポーネントの指定

結果をログに出力させる

© 2012 IBM Corporation

99

要求メトリックは管理コンソールから設定可能です。デフォルトはOFFですので取得するときはONに設定する必要があります。対象コンポーネントの設定では、どのコンポーネントに対して要求メトリックを有効にするかを指定(複数指定可能)できます。コンポーネントはServlet、EJB、JDBC、WebService、JMS、AsynchronousBeanの6種類です。

Designing Web System Infrastructure with WAS V8.0	要求メトリックで出力される項目	
	フィールド	説明
	parent	呼び出し側(親)の識別子
	current	呼び出された側(子)の識別子
	ver	親と子の相関関係のバージョン、親子の両者は同じ数字になります
	ip	アプリケーション・サーバーがあるノードのIPアドレス
	pid	アプリケーション・サーバーのプロセスID
	time	アプリケーション・サーバーのプロセス開始時刻
	reqid	要求メトリックがリクエストに割り当てるID
	event	実際のトレースを区別するために割り当てられるイベントID
	type	リクエストのタイプ HTTP、URI、EJB、JDBC、JMS、非同期ビーン(COMMONJ_WORK_POOLED、COMMONJ_TIMER)、Webサービスがサポート
	detail	リクエストの詳細データで、URIのフルパス、SQLステートメント、EJBメソッド名など
	elapsed	トータルの経過時間(ミリ秒)で、子の経過時間を全て含んだ時間
	bytesIn	Webサーバー・プラグインが受け取ったリクエストのバイト数
	bytesOut	Webサーバー・プラグインがクライアントに送信されたレスポンスのバイト数
© 2012 IBM Corporation		100

要求メトリックの機能でログに出力される項目です。

Designing Web System Infrastructure with WAS V8.0

ARM(Application Response Measurement)

- ARM(Application Response Measurement)
 - ◆ レスポンスタイムを取得して外部のシステム管理ツールに報告するためのAPIを提供
 - Tivoli製品との連携
 - Tivoli Composite Application Manager for Response Time Tracking に要求メトリックで取得した情報を送信
 - ◆ アプリケーション・コードに測定の開始点と終了点を記述してその間のレスポンスタイムを取得

© 2012 IBM Corporation
101

ARMはレスポンスタイムを取得し、取得したデータを外部のシステム管理ツール(具体的にはTivoli Composite Application Manager for Response Time Tracking などのTivoli製品。)に報告するためのAPIを提供するものです。アプリケーションのコードに測定の開始点と終了点を記述し、その間のレスポンスタイムをデータとして取得しますので、より詳しくレスポンスタイムのデータを取得したい際や、Tivoli製品と連携を行いたい場合には有効です。

Designing Web System Infrastructure with WAS V8.0

アドバイザー

- アドバイザー
 - ◆ アドバイザーを利用することでアプリケーション・サーバーのパフォーマンスを改善するための様々なヒントを取得可能
 - ◆ 二つのアドバイザー機能は共にPMIから元データを収集
 - ランタイム・パフォーマンス・アドバイザー (Performance and Diagnostic Advisor)
 - Tivoli Performance Viewer アドバイザー
 - ◆ 目的に応じてどちらのアドバイザーを使用するかを選択



これまでのアドバイザーより有用なアドバイスをこなしてくれます。ただし、“大きなお世話”と思うようなアドバイスもあるかもしれません。

© 2012 IBM Corporation

102

PMIという情報収集基盤から取得するデータを元にして、WASの運用に関するアドバイスをこなう機能が二つあります。以前からアドバイザー機能はありましたが、バージョンがあがってくるにつれて、製品としてこれまで培ったベスト・プラクティスを反映してきたことで段々との的確なアドバイスを提供できるようになって来たと言われています。

Designing Web System Infrastructure with WAS V8.0	アドバイザー比較	
	ランタイム・パフォーマンス・アドバイザー	Tivoli Performance Viewer アドバイザー
アドバイスの出力方法	・SystemOut.log ・管理コンソール ・JMX 通知	管理コンソール上の Tivoli Performance Viewer画面
オペレーションの頻度	事前に調整した結果に従う	管理コンソールから最新表示をかけたとき
主なアドバイスの種類	パフォーマンス系のアドバイス: ・ORBサービスのスレッド・プール ・Webコンテナのスレッド・プール ・JDBC接続プールのサイズ ・永続化セッション(サイズ・保持時間) ・Prepared statement キャッシュのサイズ ・セッション・キャッシュのサイズ ・簡易メモリ・リーク検知 診断系のアドバイス: ・接続ファクトリー ・データ・ソース	パフォーマンス系のアドバイス: ・ORBサービスのスレッド・プール ・Webコンテナのスレッド・プール ・JDBC接続プールのサイズ ・永続化セッション(サイズ・保持時間) ・Prepared statement キャッシュのサイズ ・セッション・キャッシュのサイズ ・動的キャッシュのサイズ ・JVMのJavaヒープ・サイズ ・DB2 パフォーマンス構成ウィザード
© 2012 IBM Corporation		103

こちらは、二つのアドバイザーの比較表です。目的に応じてどちらのアドバイザーを使用してみるかを検討する際に参考にしてください。

Designing Web System Infrastructure with WAS V8.0

アドバイザ設定方法

■ ランタイム・パフォーマンス・アドバイザ

◆ アプリケーション・サーバーのパフォーマンスおよび診断アドバイザ構成をクリック

パフォーマンス

- Performance Monitoring Infrastructure (PMI)
- パフォーマンスおよび診断アドバイザ構成

➡

☒ パフォーマンスおよび診断アドバイザ・フレームワーク (ランタイム・パフォーマンス・アドバイザ) を使用可能にする

チェックを入れる

■ Tivoli Performance Viewer アドバイザ

◆ TPVにて[アドバイザ]をクリック

tomTestServer01

- アドバイザ
- 設定
- サマリー・レポート
- パフォーマンス・モジュール

➡

TPVの画面

名前	タイプ	ステータス	バージョン
Application Performance Monitor	APM	インストール済み	8.0.0
Business Process Monitor	BPM	インストール済み	8.0.0
Database Performance Monitor	DBPM	インストール済み	8.0.0
Enterprise Performance Monitor	EPM	インストール済み	8.0.0
Health Management	HM	インストール済み	8.0.0
Infrastructure Performance Monitor	IPM	インストール済み	8.0.0
Log Management	LM	インストール済み	8.0.0
Network Performance Monitor	NPM	インストール済み	8.0.0
Security Performance Monitor	SPM	インストール済み	8.0.0
System Performance Monitor	SPM	インストール済み	8.0.0
Web Performance Monitor	WPM	インストール済み	8.0.0

© 2012 IBM Corporation

104

ランタイム・パフォーマンス・アドバイザとTivoli Performance Viewerアドバイザの設定方法です。

Designing Web System Infrastructure with WAS V8.0

ITCAM for WAS(有償製品)

- 開発から運用までアプリケーションのライフサイクル全体をひとつのツールでサポート
 - ◆ 応答時間、リソース状況など詳細データを収集し、運用時の監視・データ保存のみならず、開発や問題発生時の切り分け・解析などあらゆるフェーズで活躍

Three scenarios are illustrated with starburst callouts and screenshots of the ITCAM for WAS interface:

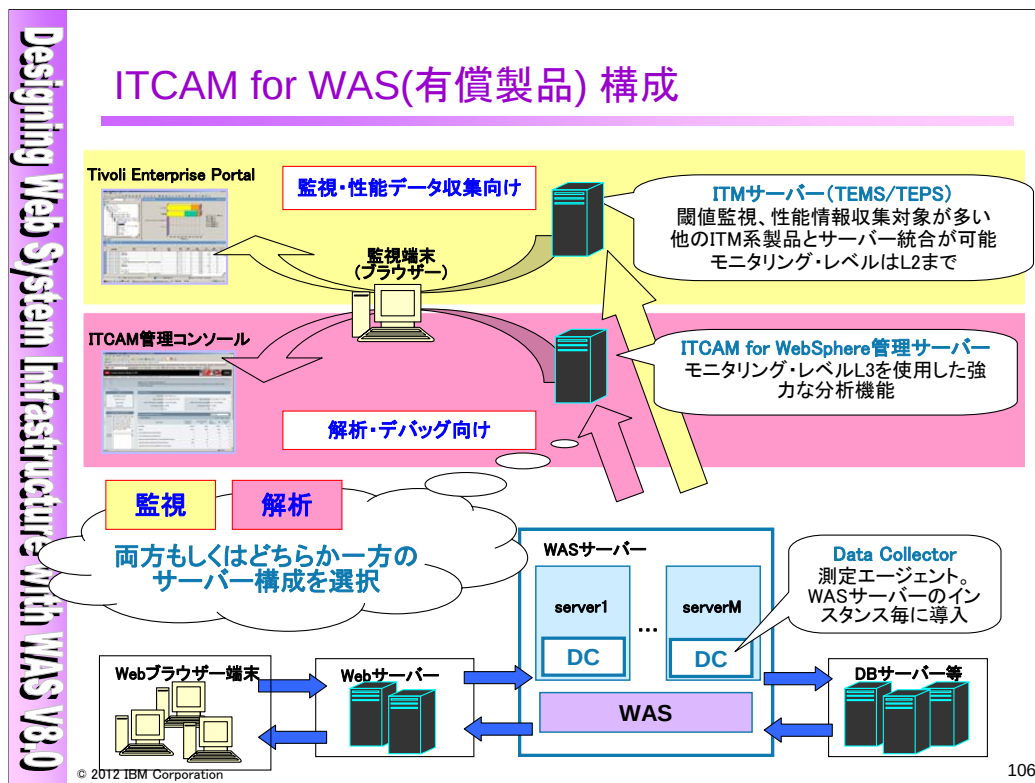
- 応答時間がわからない** (Response time unknown): リクエスト分析など (Request analysis, etc.)
- システム負荷がわからない** (System load unknown): リソース情報分析など (Resource information analysis, etc.)
- 問題解析に詳細情報が必要** (Detailed information needed for problem analysis): メソッド・トレース履歴データ参照、メモリー・リーク分析、ソフトウェア整合性検査、など (Method/trace history data reference, memory leak analysis, software integrity check, etc.)

© 2012 IBM Corporation

105

ITCAM for WAS は以下の特徴を持ち、WASサーバー上で稼動するアプリケーションの開発フェーズから運用フェーズまでをサポートします。

- WASサーバーのシステム・リソース消費状況の監視
- アプリケーションのパフォーマンスを測定
- 収集データの加工・レポートニング
- 障害発生時の問題判別・分析機能



ITCAM for WAS用のサーバーは2種類のサーバー構成から選択できます。1つはITCAM専用のサーバーを構築する構成で、モニタリング・レベルL3による高度な分析機能を使用できる開発向けの構成です。もう1つはITM(Tivoli Monitoring)のサーバーにDataCollectorを接続する構成で、閾値の設定や性能レポート用のデータ項目が豊富です。またITMやITM for DB等のITCAM以外のTivoli製品とサーバーを統一できる利点もあります。こちらは運用監視向けの構成と言えます。両方のサーバーを立てる構成も可能ですので、お客様の要件により適切なサーバー構成を選択してください。

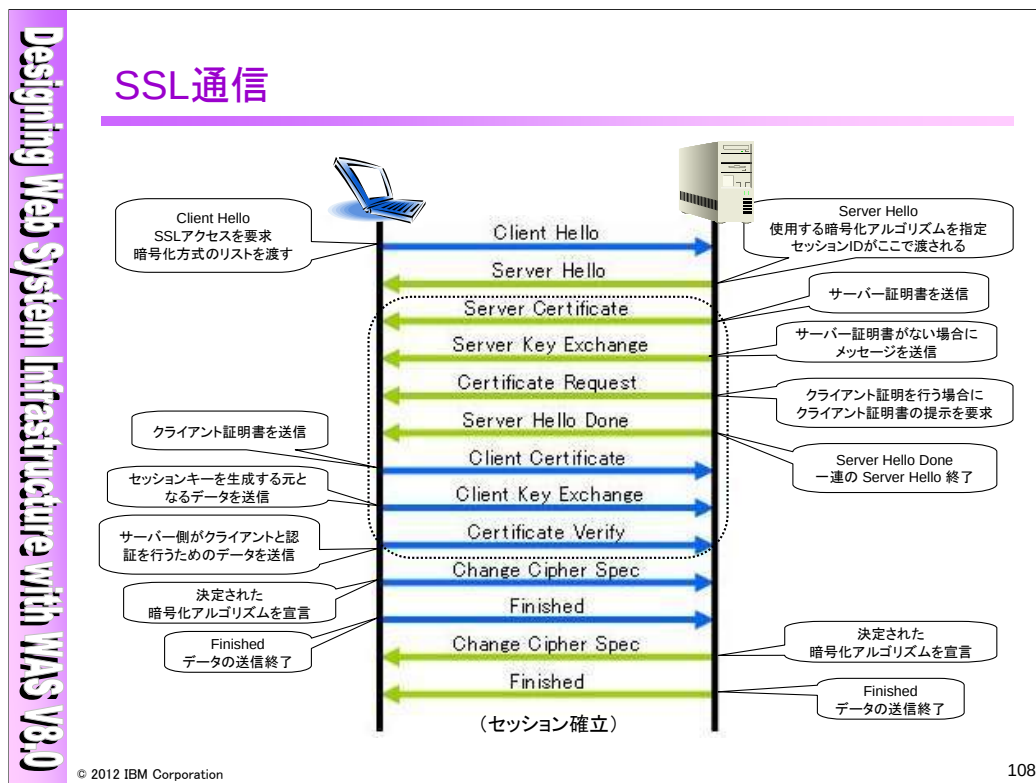
Designing Web System Infrastructure with WAS V8.0

3. チューニング

- 3-1 チューニング概要
- 3-2 ネットワーク関連のチューニング**
- 3-3 リクエスト・キューイング
- 3-4 コンポーネント別のチューニング
- 3-5 パフォーマンス・チューニングのポイント

© 2012 IBM Corporation

107



SSL(Secure Socket Layer)はNetscape Communications Corporationによって開発された認証や暗号化により、盗聴や改ざん、なりすましを防ぎ、データの機密性を保証するプロトコルです。SSLは上図のようなSSLハンドシェイクによりセキュアな接続を開始します。クライアントからSSLアクセスの要求が来ると、サーバーはクライアントとやり取りを行う暗号化アルゴリズムとSSLセッションIDを指定し、クライアントに返すとともにサーバー証明書を送信します。クライアントはそれを受けて暗号化アルゴリズムを宣言し、サーバーに返すことでSSL通信が確立します。またクライアント認証を行うときは、サーバーはクライアントに対しクライアント証明書を要求し、クライアントは適切なクライアント証明書をサーバーに返します。

Designing Web System Infrastructure with WAS V8.0

SSLハンドシェイクのキャッシュ

- SSLハンドシェイクは接続に負担がかかるので二回目からはセッションIDをキャッシュ
 - ◆ ErrorLogのレベルをInfo以上にするとハンドシェイクのキャッシュが確認可能

error.log (LogLevel info)

```
[Wed Sep 21 13:44:00 2005] [info] Session ID: AABjqsqzy6X02W9ygvVu4Sg9/CpYWFhYQzDIEAAAAA=
[Wed Sep 21 13:44:02 2005] [info] New Session ID: 1
[Wed Sep 21 13:44:02 2005] [info] Session ID: AABjqsqzy6X02W9ygvVu4Sg9/CpYWFhYQzDIEAAAAA=
[Wed Sep 21 13:44:02 2005] [info] New Session ID: 0
[Wed Sep 21 13:45:16 2005] [info] Session ID: AABjqsqzy6X02W9ygvVu4Sg9/CpYWFhYQzDIEAAAAA=
[Wed Sep 21 13:45:16 2005] [info] New Session ID: 0
```

1はSSLの新規接続

0はSSLセッションを再利用

- ◆ SSLV3Timeout、SSLV2Timeoutを設定すると、SSLセッションIDのキャッシュにタイムアウトを設定できる
- ◆ SSLV3はデフォルト120秒、SSLV2はデフォルト40秒

error.log (LogLevel info)

```
[Sun Nov 13 19:37:08 2005] [info] Session ID: AABcXpR4hkJDp2yf9Jln0jQV7sVYWFhYQ3cXVAAAAA=
[Sun Nov 13 19:37:08 2005] [info] New Session ID: 1
[Sun Nov 13 19:37:10 2005] [info] Session ID: AABcXpR4hkJDp2yf9Jln0jQV7sVYWFhYQ3cXVAAAAA=
[Sun Nov 13 19:37:10 2005] [info] New Session ID: 0
[Sun Nov 13 19:38:16 2005] [info] Session ID: AABcXpR4hkJDp2yf9Jln0jQV7sVYWFhYQ3cXVAAAAA=
[Sun Nov 13 19:38:16 2005] [info] New Session ID: 0
[Sun Nov 13 19:38:39 2005] [info] Session ID: AABcXpR4hkJDp2yf9Jln0jQV7sVYWFhYQ3cXVAAAAA=
[Sun Nov 13 19:38:39 2005] [info] New Session ID: 0
[Sun Nov 13 19:39:23 2005] [info] Session ID: AABcXpR4hkJDp2yf9Jln0jQV7sVYWFhYQ3cXVAAAAA=
[Sun Nov 13 19:39:23 2005] [info] New Session ID: 1
```

タイムアウトが来るとSSLハンドシェイクをやり直し、新しいセッションIDになる。

109

SSLハンドシェイクは通常のHTTP接続よりもさらに接続に負担がかかるので2回目からSSLのセッションIDをキャッシュすることで、パフォーマンスダウンを防いでいます。SSLセッションIDがキャッシュされたかどうかの確認はerror.logのLogLevelをinfo以上にすると確認することができます。info以上に設定すると、上の例のように「New Session ID:」という出力がされます。この数字が1は新規接続を示しており、0はセッションの再利用がされたことを示しています。またセッションIDもログに出力されるようになります。さらに詳しく見たいときはSSLCacheTraceLogを設定してください。

また、SSLV3TimeoutとSSLV2TimeoutディレクティブはこのSSLセッションのタイムアウトを設定します。デフォルトでSSLV3Timeoutは120秒、SSLV2Timeoutは40秒ですので、SSLV3も120秒経つとSSLハンドシェイクをやり直します。ディレクティブの有効範囲はSSLV3Timeoutは0-86400秒、SSLV2Timeoutは0-100秒ですので、できる限りセッションIDを再利用し、新規のSSLハンドシェイクによるパフォーマンスダウンを防ぎたい場合には、この値を大きくします。

WAS V8.0 によるWebシステム基盤設計ワークショップ

109

Designing Web System Infrastructure with WAS V8.0

SSLCipherSpec

- SSLCipherSpecによる暗号化方式の指定
 - ◆ Client Helloでクライアントが対応している暗号化方式リストを渡す
 - ◆ Server Helloで暗号化を指定

SSL_RSA_WITH_RC4_128_MD5
 SSL_RSA_WITH_RC4_128_SHA
 SSL_RSA_WITH_3DES_EDE_CBC_SHA
 SSL2_RC4_128_WITH_MD5
 SSL_RSA_EXPORT_WITH_RC4_40_MD5

Client Hello

Server Hello

SSL_RC4_WITH_RC4_128_MD5

- ◆ IHSのSSLCipherSpecディレクティブで暗号化方式を指定
 - 複数指定も可能
 - 指定されたものに対応していないブラウザからの接続は拒否される
- ◆ IHSv6.0、SSLv3で対応している暗号化方式

Short name	Long name	Description
3A	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Tripw-DES SHA (168-bit)
33	SSL_RSA_EXPORT_WITH_RC4_40_MD5	RC4 SHA (40-bit)
34	SSL_RSA_WITH_RC4_128_MD5	RC4 MD5 (128-bit)
39	SSL_RSA_WITH_DES_CBC_SHA	DES SHA (56-bit)
35	SSL_RSA_WITH_RC4_128_SHA	RC4 SHA (128-bit)
36	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	RC2 MD5 (40-bit)
Cipher specification 36 requires Netscape Navigator V4.07; it does not work on earlier versions of Netscape browsers.		
32	SSL_RSA_WITH_NULL_SHA	
31	SSL_RSA_WITH_NULL_MD5	
30	SSL_NULL_WITH_NULL_NULL	
62	TLS_RSA_EXPORT1024_WITH_RC4_56_SHA	RC4 SHA Export 1024 (56-bit)
64	TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA	DES SHA Export 1024 (56-bit)

© 2012 IBM Corporation
110

SSLハンドシェイクでは、ClientHello時にクライアントが対応している暗号化方式のリストをサーバーに渡し、ServerHelloでサーバーから暗号化を指定しますが、IHSのSSLCipherSpecディレクティブで暗号化の方式をサーバー側で指定することができます。このディレクティブは複数指定することも可能ですので、暗号強度の弱いものしかサポートしないブラウザ用と暗号強度の強いものをサポートするブラウザで暗号化強度を使い分けて指定することもできます。また指定されたものに対応していないブラウザからの接続は許可されません。

Designing Web System Infrastructure with WAS V8.0

ファイル圧縮の設定

httpd.conf

```
<IfModule mod_deflate.c>
SetOutputFilter DEFLATE
SetEnvIfNoCase Request_URI "%.pdf$" no-gzip
SetEnvIfNoCase Request_Method POST no-gzip
DeflateFilterNote ratio
</IfModule>

LogFormat "%r" %b (%{ratio}n) "%{User-agent}i" %s deflate
CustomLog logs/deflate_log deflate
```

Annotations for httpd.conf:

- pdfファイルやPOSTメソッドのリクエストは圧縮しない
- 圧縮比率をログに出力させる

deflate_log

```
"GET /index.html HTTP/1.1" 793 (47) "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)" 200
"GET /httpTech.main.gif HTTP/1.1" 110163 (99) "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)" 200
"GET /httpTech.view1.gif HTTP/1.1" 1001 (94) "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)" 200
"GET /sample.pdf HTTP/1.1" 848303 (-) "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)" 200
```

Annotations for deflate_log:

- 圧縮比率がログの()内に表示される。圧縮しないときより、送信バイト数が47%に減少
- pdfファイルは圧縮されない

Resulting log entry after compression:

```
"GET /index.html HTTP/1.1" 1617 (-) "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)" 200
```

© 2012 IBM Corporation

111

上の例は、SetEnvIfディレクティブで、リクエストURIが.pdfで終わるもの（PDFファイルのアクセス）とリクエストメソッドがPOSTのものは圧縮対象としない設定を行っています。また、DeflateFilterNoteディレクティブにより、圧縮率を示す変数を指定し、LogFormatでその圧縮比率を出力させることで、ファイルの圧縮を確認することが可能です。

Designing Web System Infrastructure with WAS V8.0

Webコンテナー・トランスポート・チェーン

■ トランスポート・チェーン

◆ 3つのインバウンド・チャンネル階層を持つ

The screenshot displays the 'Transport Chain' configuration for 'WCIInboundDefault'. It shows three inbound channels: 'WCIInboundDefault', 'WCIInboundDefaultSecure', and 'WCIInboundDefaultSecure'. The 'WCIInboundDefault' channel is selected, showing its configuration details. The configuration includes a list of inbound channels and their settings, such as 'Port', 'Thread Pool', 'Max Open Connections', and 'Idle Timeout'.

TCPインバウンド・チャンネルでは、ポート、スレッド・プールの選択、最大のオープン接続数、タイムアウト、アクセス制御を設定

HTTPインバウンド・チャンネルでは、判断ウェイト、最大パーシスタント要求数、ユーザー・パーシスタント(キープアライブ)接続の有無、タイムアウト、アクセス・ロギング、エラー・ロギングの有無を設定

Webコンテナー・インバウンド・チャンネルでは、判断ウェイト、書き込みバッファ・サイズを設定

© 2012 IBM Corporation

112

上図は9080番のポートを所有するWCIInboundDefaultというWebコンテナー・トランスポート・チェーンの例です。このWCIInboundDefaultではTCPインバウンド・チャンネル、HTTPインバウンド・チャンネル、Webコンテナー・インバウンド・チャンネルという3つのチャンネルを持っており、それぞれに対し設定項目があります。

Designing Web System Infrastructure with WAS V8.0

3. チューニング

- 3-1 チューニング概要
- 3-2 ネットワーク関連のチューニング
- 3-3 リクエスト・キューイング
- 3-4 コンポーネント別のチューニング**
- 3-5 パフォーマンス・チューニングのポイント

© 2012 IBM Corporation

113

Designing Web System Infrastructure with WAS V8.0

Webサーバー: ListenBackLogのキューの確認方法

- 処理待ち可能なキューを確認する方法 (AIX only)

```

(ポート番号におけるTCP番号を確認)
host1@root[/usr/IBMHS/conf]netstat -aAn | grep LISTEN | grep 80
73686200 tcp        0      0 *.80          *.*          LISTEN

(kdbコマンド実行し、以下のコマンドを入力)
host1@root[/usr/IBMHS/conf]kdb
(0)> sockinfo 73686200 tcpcb

---- SOCKET INFO ----(@ 73686200)----
type..... 0001 (STREAM)
opts..... 000E (ACCEPTCONN|REUSEADDR|KEEPALIVE)
linger..... 0000 state..... 0180 (PRIV|NBIO)
pcb.....@ 73686154 proto.....@ 024C8CE8
lock.....@ 736CF280 head.....@ 00000000
q0.....@ 00000000 q.....@ 00000000
qlen..... 0000 qlen..... 0000 qlimit..... 01FF
timeo..... 0000 error..... 0000 special..... 0A08
pgid..... 00000000 oobmark..... 00000000
          
```

qlimitが処理待ち可能なキューの最大数
(16進数)を示す。デフォルトでは1FF=511。

© 2012 IBM Corporation
114

AIXでは、ポートにおける処理待ち可能なリクエストの数を確認することが可能です。

- ① netstat -aAn | grep LISTEN | grep <port number>を実行し、<port number>に対するTCP番号を確認します。一番左に現れる番号がTCP番号です。
- ② kdbコマンドを実行します。
- ③ プロンプトが(0) > になりますので、sockinfo <TCP number> tcpcb と入力します。
- ④ 何回かEnterキーを押して先へ進むと、SOCKET INFOの画面が表示されます。このときのqlimitの値が16進数で示された、処理待ち可能なキューの数になります。

上の実行例は、デフォルト時の80番ポートについての例です。qlimitが1FFですので10進数で511を示していることが確認できます。

Designing Web System Infrastructure with WAS V8.0

セッション管理の設定

- 管理コンソールからアプリケーション・サーバーを選択し、[Webコンテナ設定]→[セッション管理]
- メモリー内の最大セッション・カウント、セッション・タイムアウトを設定
 - ◆ デフォルトは1000セッション、タイムアウトは30分
 - ◆ 「オーバーフローの許可」はセッション情報が複数サーバー間で共有されていない場合のみ有効

© 2012 IBM Corporation
115

管理コンソールからアプリケーション・サーバーを選択し、[Webコンテナ設定]→[セッション管理]を選択すると、設定画面になります。ここでは最大セッション・カウントとセッションのタイムアウトを設定します。また、DBによるパーススタンスやメモリー間での複製を行っていない環境では、「オーバーフローの許可」の有無も設定できます。

Designing Web System Infrastructure with WAS V8.0	TPVによるセッションのモニタリング		
	■ 「サーブレット・セッション・マネージャー」カテゴリー		
	カウンター	レベル	説明
	サーブレット・セッション・マネージャー		
	ActivateNonExistSessionCount	全て	セッション・タイムアウトとなり存在しないセッションに対するリクエスト数
	ActiveCount	全て	リクエストにより現在アクセスしているセッションの総数
	AffinityBreakCount	全て	中段されたセッション・アフィニティーの数
	CacheDiscardCount	全て	分散セッションでキャッシュから破棄されたセッションの数
	CreateCount	全て	作成されたセッションの数
	ExternalReadSize	拡張	外部から読み取られたセッションのサイズ
	ExternalReadTime	拡張	外部からのセッションの読み取りにかかる時間（ミリ秒）
	ExternalWriteSize	拡張	外部に書き込んだセッションのサイズ
	ExternalWriteTime	拡張	外部へのセッションの書き込みにかかる時間（ミリ秒）
	InvalidateCount	全て	無効化されたセッションの数
	LifeTime	全て	平均セッション持続時間（ミリ秒）
	LiveCount	基本	現在活動中のセッションの数
	NoRoomForNewSessionCount	全て	「オーバーフローの許可」にチェックされていない時に適用。最大セッション数を越えたため、新規セッション・リクエストを処理できない数
	SessionObjectSize	全て	セッション・オブジェクトのサイズ
	TimeSinceLastActivated	全て	直前のアクセス時刻と現在の時刻の差（ミリ秒）
	TimeoutInvalidationCount	全て	タイムアウトで無効になったセッションの数

© 2012 IBM Corporation

116

TPVでセッションの状態は「サーブレット・セッション・マネージャー」のカテゴリーから確認できます。

Designing Web System Infrastructure with WAS V8.0

セッション・オブジェクトの書き込み頻度の設定

- セッション・オブジェクトが書き込まれる頻度の設定
 - ◆ アプリケーション・サーバーを選択し、[Webコンテナ設定]→[セッション管理]→[分散環境設定]から[カスタム・チューニング・パラメーター]

チューニング・レベル(非常に高)
パフォーマンスの最適化

↑ ↓

チューニング・レベル(低)
フェイルオーバーの最適化

「カスタム」を選択すると、
書き込み頻度、書き込みの内容、
セッション・クリーンアップのスケジュールが
設定可能。

© 2012 IBM Corporation

管理コンソールからアプリケーション・サーバーを選択し、[Webコンテナ設定]→[セッション管理]→[分散環境設定]から[カスタム・チューニング・パラメーター]画面から設定を行います。

Designing Web System Infrastructure with WAS V8.0

セッション・データベースの設定

- セッションをデータベースに保管する際の設定
 - ◆ アプリケーション・サーバーを選択し、[Webコンテナ設定]→[セッション管理]→[分散環境設定]から、分散セッションにデータベースを選択

© 2012 IBM Corporation
118

管理コンソールからアプリケーション・サーバーを選択し、[Webコンテナ設定]→[セッション管理]→[分散環境設定]から、分散セッションにデータベースを選択すると、設定画面になります。

Designing Web System Infrastructure with WAS V8.0

3. チューニング

- 3-1 チューニング概要
- 3-2 ネットワーク関連のチューニング
- 3-3 リクエスト・キューイング
- 3-4 コンポーネント別のチューニング
- 3-5 パフォーマンス・チューニングのポイント**

© 2012 IBM Corporation

119

Designing Web System Infrastructure with WAS V8.0

ボトルネック発見のアプローチ①(1/4)

- 現象
 - ◆ 飽和点に達してもCPUが使われていない
- 予測
 - ◆ 各サーバーの処理能力に余裕があるにも関わらず、スループットやレスポンスタイムが頭打ち
- 検証
 - ◆ まずは滞留している場所を突き止める
- 取得項目
 - ◆ IHSのアクセス・ログ、プラグインのログ、WASのSystemOutログ(要求メトリックを設定)(p.10-17参照)

```

graph TD
    A[IHSのアクセス・ログからレスポンスタイムを取得] --> B[要求メトリックを取得し、各コンポーネントのレスポンスタイム取得]
    B --> C[それぞれのレスポンスタイムの差から各コンポーネントでの処理時間を計算]
    C --> D[著しく処理時間の長いところに注目！]
          
```

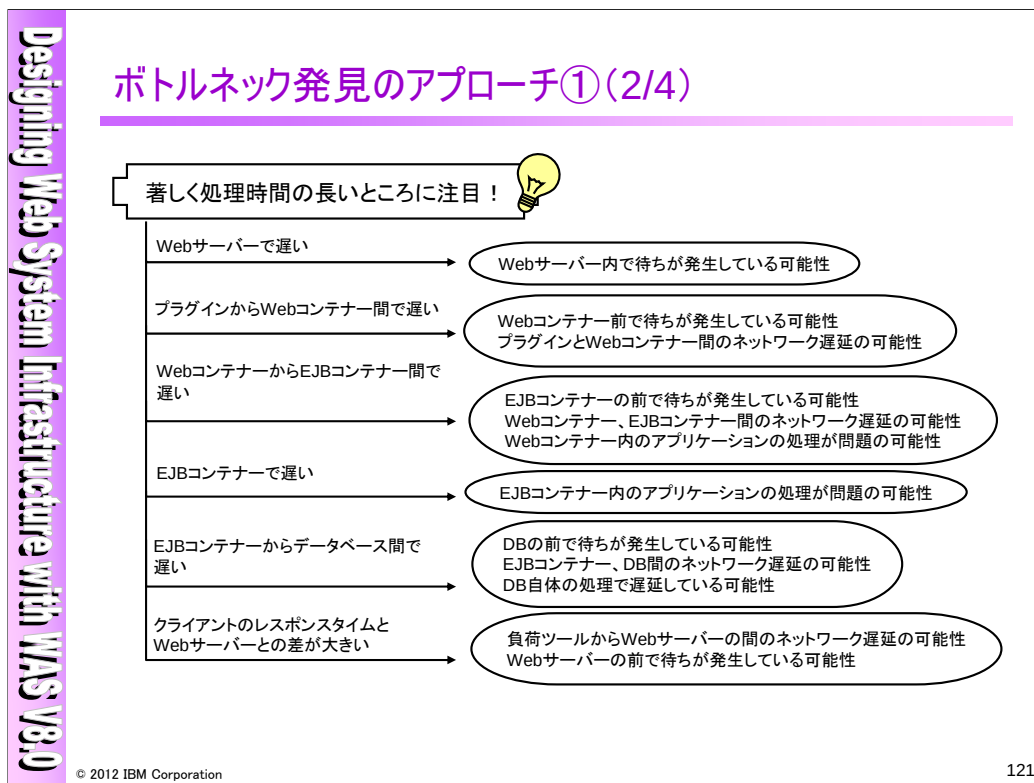
© 2012 IBM Corporation
120

まずは飽和点に達しているがリソースのCPUが使われていない場合です。

この場合は、各サーバーの処理能力には充分余裕があるにも関わらず、スループットやレスポンスタイムが頭打ちになっている状態ですので、システムはさらに多くのリクエストを処理できる状態にも関わらず、どこかでリクエストが滞留しているためにリクエストが処理が届いていないことが原因として考えられます。

問題解決は、リクエストが滞留している箇所を突き止めることから始まります。

IHSのログにレスポンスタイムを出力させるように設定し、さらにWASで要求メトリックを設定します。IHSのアクセス・ログ、プラグインのログ、アプリケーション・サーバーのSystemOut.logを取得して、各コンポーネントでかかるレスポンスタイムから、どこで処理に時間が取られているのかを判断します。



どこで処理時間がかかっているかによって、考えられる滞留の原因はさまざまです。例えばプラグインからWebコンテナの間で処理に時間を取られている場合、Webコンテナの前で待ちが発生している可能性もありますし、リモート環境であればネットワークで遅延が発生している可能性もあります。また、データベースへの通信で処理に時間を取られている場合は、データベース前で待ちが発生している場合もありますし、データベース内の処理が遅い可能性も考えられます。

Designing Web System Infrastructure with WAS V8.0

ボトルネック発見のアプローチ①(3/4)

以下の順番で遅延の原因を特定

1. 待ちが発生している場合の確認方法
 - ◆ TPVでスレッド・プールを確認、Webサーバーはログを確認
 - ◆ Webサーバーの前で待ちが発生している場合
 - 同時リクエスト数を超えると、超えたことを示す警告がWebサーバーのエラーログに出力される
 - ➡ Webサーバーの最大アクセス数を大きくする
 - ◆ Webコンテナー、EJBコンテナーの前で待ちが発生している場合
 - TPVで各々のスレッド・プールを確認し、使用率が100%のまま維持され続けていることを確認
 - ➡ Webコンテナー、EJBコンテナーの最大サイズを大きくする
 - ◆ データベースの前で待ちが発生している場合
 - TPVで接続プールを確認し、使用率が100%のまま維持されていることを確認
 - ➡ 接続プールの最大接続数を上げる。DBのMAXAPPLSパラメーターを上げる
2. ネットワーク遅延の確認方法
 - ◆ 各々のネットワーク機器、サーバーで帯域での使用率やパケット量を確認
 - ➡ ネットワーク機器のパラメーター調整、ネットワーク・カードやケーブルを変える
3. アプリケーション処理の遅延の確認方法
 - ◆ アプリケーションのコードを確認、必要があればプロファイラーを使用
 - ➡ アプリケーションの変更

© 2012 IBM Corporation
122

滞留の原因は、許容できるリクエストの数が少ないためにリクエストの待ちが発生している場合、サーバー間のネットワークで遅延が発生している場合、アプリケーションなどのロジックに遅延の原因がある場合の3つが考えられます。

待ちが発生している場合、Webサーバーはログで、アプリケーション・サーバーはTPVを用いて確認します。Webサーバー前で待ちが発生しているとWebサーバーのエラーログに同時処理できるリクエスト数を超えたことを示す警告が出力されますので、そのメッセージで判断します。アプリケーション・サーバーのTPVでは、WebコンテナーやEJBコンテナー（リモートEJBクライアントからのアクセスの場合）、接続プールの使用率を確認し、使用率が100%の状態がずっと続いているようであれば、リクエストの待ちが発生している可能性が高くなりますので、各々の箇所でも最大アクセス数を大きくするようなチューニングを検討します。

ネットワークによる遅延の疑いがある場合、各ネットワーク機器やサーバーで帯域の使用率や流れるパケットの量を確認します。使用率が高い状態が続くような場合は、ネットワーク遅延によるボトルネックが考えられますので、ネットワーク機器のパラメータの調整やネットワーク・カードの変更などを検討します。

以上2つに問題はないにも関わらず遅延が続くような場合は、アプリケーションの処理そのもので遅延が発生している可能性が高くなります。

ボトルネック発見のアプローチ①(4/4)

■ アプリケーション分析

◆ レスポンスの悪いロジックの特定

- IHSのアクセス・ログから、時間がかかっているリクエストURLを判断し、レスポンスの悪いロジックに目星を付ける

◆ レスポンスの悪いロジックがある程度特定できている場合

- 関連するコードを追う
- 処理が複雑なときはプロファイラーを使用し、ボトルネックを探し出す

◆ プロファイラー

- アプリケーションのボトルネックやメモリー・リークを発見するためのツール
 - 各クラス、メソッド単位の実行時間
 - オブジェクトのサイズ、参照関係
 - ヒープの使用状況、ガーベージ情報
- ボトルネックと判断された箇所についてロジックの見直しを行う

© 2012 IBM Corporation

123

アプリケーションロジックにパフォーマンス遅延の可能性がある場合には、どのロジックで処理が遅れているのか目星を付けることから始まります。IHSのアクセス・ログにはレスポンスタイムとともにリクエストURIが記載されていますので、遅延が発生しているリクエストURIから、共通するロジックを見つけ、リクエストの遅延箇所の目星を付けます。

遅延が発生しているロジックに目星がついたら、アプリケーションのボトルネックを調査します。アプリケーションのボトルネックを分析する方法には、プログラムのコードレビューを行う方法とプロファイラーを用いる方法の2つがあります。どちらの方法を取るか明確な基準はありませんが、ロジック内の処理が非常に複雑な場合であれば、プロファイラーを用いたほうが効果的です。

プロファイラーは、アプリケーションのボトルネックやメモリー・リークを発見するためのツールです。プロファイラーでは、各クラス、メソッド単位の実行時間やオブジェクトのサイズ、参照関係、ヒープの使用状況、ガーベージ情報などの情報を取得することが可能ですので、その結果に従ってボトルネックと判断された箇所についてロジックの見直しを行います。

Designing Web System Infrastructure with WAS V8.0

ボトルネック発見のアプローチ②(1/5)

- 現象
 - ◆ レスポンスが出し切れていない
 - ◆ スループットが出し切れていない
- 予測
 - ◆ レスポンスタイムはいいが、スループットが悪い

 負荷がかけきれていないのでは？
 - ◆ レスポンスタイムもスループットも悪い

 どこかでマシンの処理能力が限界に達してるのでは？
- 検証
 - ◆ 負荷が正しくかけられているか、マシンのハードウェア・スペックに異常はないかを突き止める

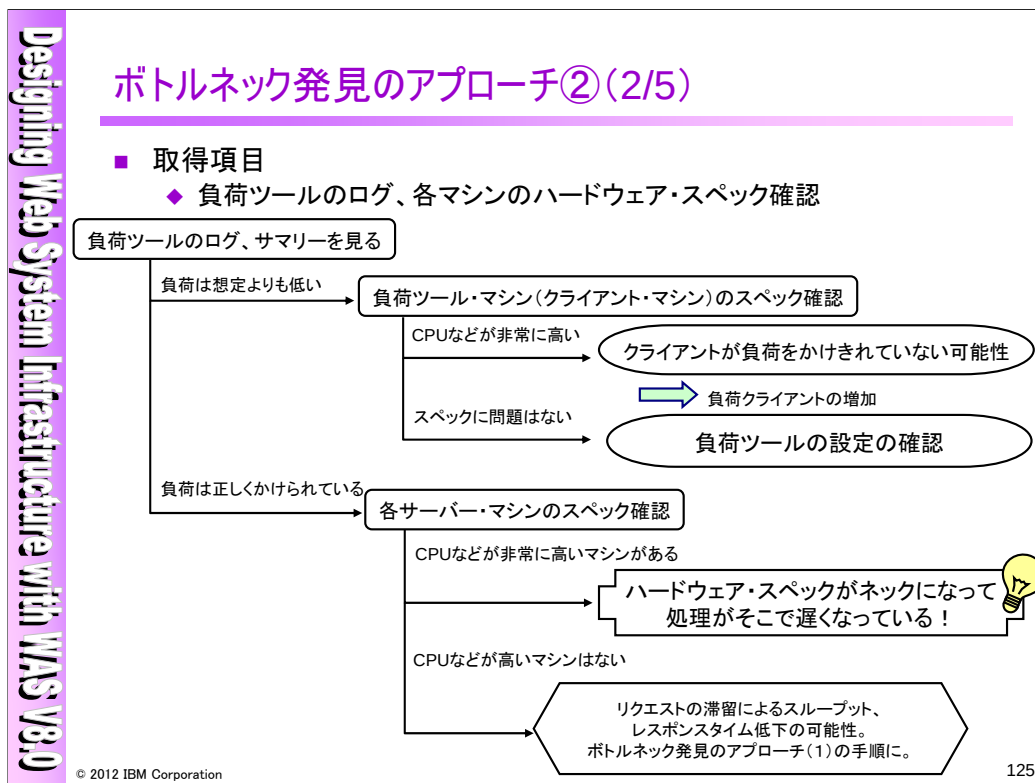
© 2012 IBM Corporation
124

次にレスポンス、スループットが出し切れていない場合を考えます。

レスポンスタイムとスループットの間には相関関係がありますので、レスポンスタイムも悪くなれば、スループットも悪くなります。

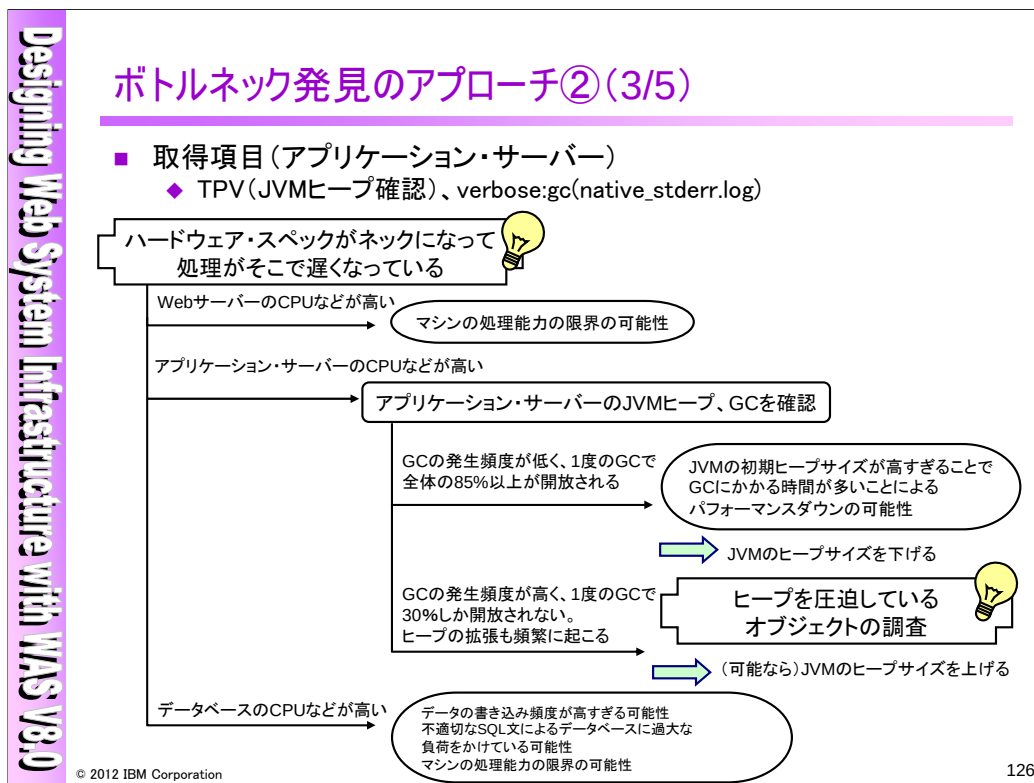
レスポンスタイムがいいのにスループットが悪いときは、設定されている負荷をかけきれていない可能性があります。また、レスポンスタイムやスループットが悪い場合、どこかでマシンの処理能力が限界に達している可能性があります。

問題解決は、負荷が正しくかけられているかを確認することと、マシンのCPUやメモリー、I/Oに異常はないかを突き止めることから始まります。

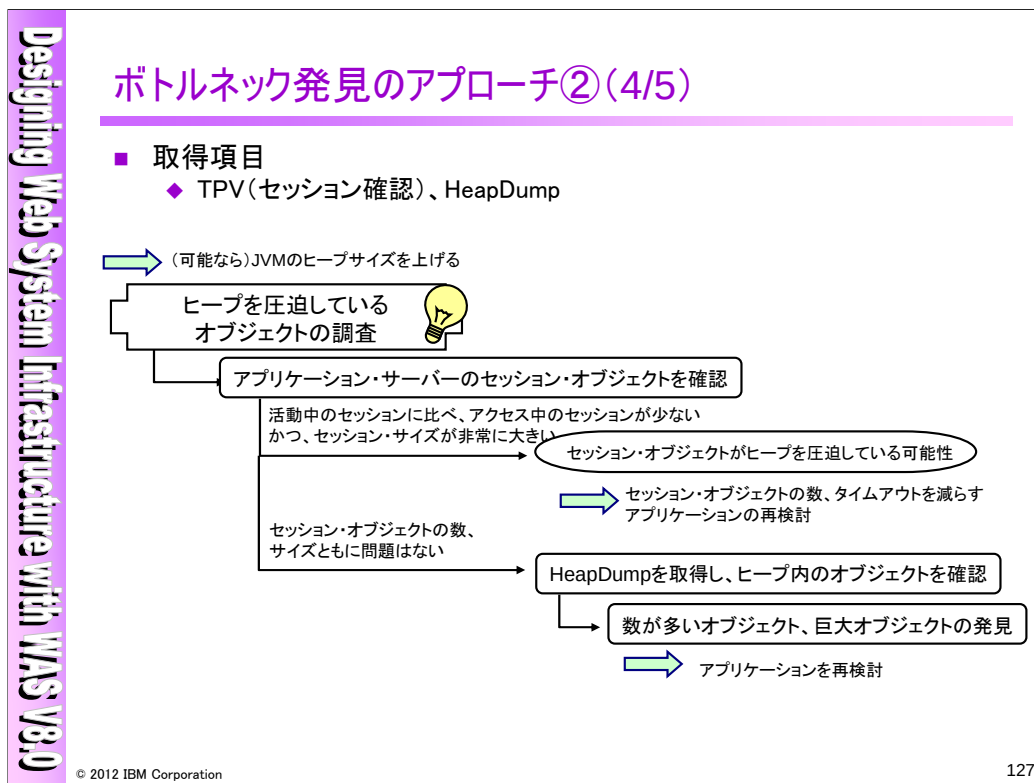


まずは負荷が正しくかけられているかを確認するために、負荷ツールのログやサマリーを確認します。想定されている負荷よりも低いことがわかったら、負荷ツール・マシン(クライアント・マシン)のスペックを確認します。負荷ツールクライアントはスペックの低いマシンが使われることが多いため、負荷ツールクライアントのCPUが非常に高くなり、必要な負荷をかけきれていないというケースが考えられます。その場合は負荷クライアントの増加を検討し、必要な負荷をかける環境を構築する必要があります。

必要とされる負荷が正しくかけられていることが確認できたら、vmstatコマンドやnmonを用いて各サーバー・マシンのCPUやメモリー、ディスクI/Oを確認します。スペックに問題のあるマシンがあれば、そのマシンのハードウェア・スペックがネックになって処理が遅れている可能性が考えられます。マシンのスペックに問題が見つからないときは、リクエストの滞留により、スループットやレスポンスが低下している可能性がありますので、「ボトルネック発見アプローチ(1)」の手順に従い、各コンポーネントでのレスポンスタイムを算出し、滞留の原因を突き止めます。



ハードウェア・スペックがネックとなって処理の遅延となっている場合でも、どのマシンでネックとなっているかにより、原因はさまざまです。アプリケーション・サーバーのスペックが高くなっている場合は、さらに詳しく原因を探るために、アプリケーション・サーバーのJVMヒープを確認する必要があります。TPVを用いてJVMヒープの状況を、verbose:gcの出力よりGCの状況をそれぞれ確認します。GCの発生頻度が低く、1度のGCで全体の85%以上が開放されている場合には、JVMヒープサイズが大きすぎるために、GCで多くの時間が取られ、パフォーマンスダウンに繋がっていると考えられます。この場合にはJVMヒープサイズを下げることを検討します。逆にGCの発生頻度が高く、1度のGCで30%しか開放されず、ヒープの拡張も頻繁に起こる場合には、ヒープサイズが小さすぎるためにGCが頻発し、パフォーマンスダウンに繋がっていると考えられます。この場合にはJVMヒープサイズを上げることを検討するとともに、どのオブジェクトがヒープを圧迫しているのか確認する必要があります。



ヒープサイズを圧迫するオブジェクトの原因を探るには、HeapDumpを取得し、ヒープ内のオブジェクトの情報を確認します。しかしセッション・オブジェクトの場合はTPVでも確認ができますので、TPVでまずセッションの情報を確認します。セッション・オブジェクトのサイズが非常に大きくなっている際には、セッション・オブジェクトがヒープを圧迫している可能性がありますので、セッション・オブジェクトの数やタイムアウトを検討するとともに、セッションを用いるアプリケーションの見直しを行います。活動中のセッションに比べ、アクセス中のセッションが非常に少ない場合には、使われていないセッション・オブジェクトがヒープに大量に残っていることになりますので、セッションのタイムアウトを少なくすることを検討します。

セッション・オブジェクトに問題がない場合は、HeapDumpにより、ヒープ内で巨大なオブジェクトがないか、大量に作られているオブジェクトがないかを確認し、アプリケーションの見直しを行います。

Designing Web System Infrastructure with WAS V8.0

ボトルネック発見のアプローチ②(5/5)

- データベースがネックとなる場合
 - ◆ 書き込み頻度
 - DiskへのI/Oでネック
 - ➡ 書き込み頻度を検討。セッションDBの場合はWASの管理コンソールから指定可能
また、以下に示すようにSQL文を改善することで効果が得られる可能性も大
 - ◆ DBのデッドロック
 - スナップショットを取得し、デッドロックの確認
 - ◆ SQL文の調査
 - 最適なSQL文は、もっとも少ないリソースを使用し、もっとも早く抽出可能な文
 - アクセスパス
 - INDEX
 - JOIN処理
 - ➡ 発行されるSQL文を再検討

© 2012 IBM Corporation
128

データベースがネックと考えられる場合の問題判別の方法を紹介します。

可能性として考えられる現象の一つに、ディスクI/Oが非常に大きくなっている場合があります。この場合はデータベースへの書き込み頻度が非常に高くなっていますので、書き込み頻度に問題がないかを検討します。セッション・データベースの場合は、WASの管理コンソールから、書き込み頻度を設定することが可能です。また、SQL文を改善することで大きな効果が得られる可能性もあります。

最適なSQL文とは、最も少ない資源を使用し、最も早く抽出できるようにしたSQL文です。そのためにはアクセスパスやINDEX、JOIN処理などを確認します。

アクセスパス:SQL文で要求されたデータにDBがアクセスするまでのパス。最短でアクセスできることがもっとも有効。

INDEXのないSELECT:検索に対してデータベースへのI/Oを増加させる原因になります。

データの列の値がユニークなものをINDEXとし、それをルートページ→リーフページ(INDEXとデータがどこにあるかを示すポインタが定義)と辿ることで、検索によるI/O回数を削減しパフォーマンスを向上させることができます。

不要なJOIN処理:複数の表から結果を表示させるJOINはパフォーマンスにも影響しますので、不要なJOIN処理は行わないようにします。

Designing Web System Infrastructure with WAS V8.0

ボトルネック発見のアプローチ③(1/2)

- 現象
 - ◆ 飽和点に達した後、スループットが急激に悪化する
- 予測
 - ◆ システムの処理能力が限界値を大きく超えた

→ どこかでマシンの処理能力が限界に達し、ダウン(もしくはハング)しているのでは？

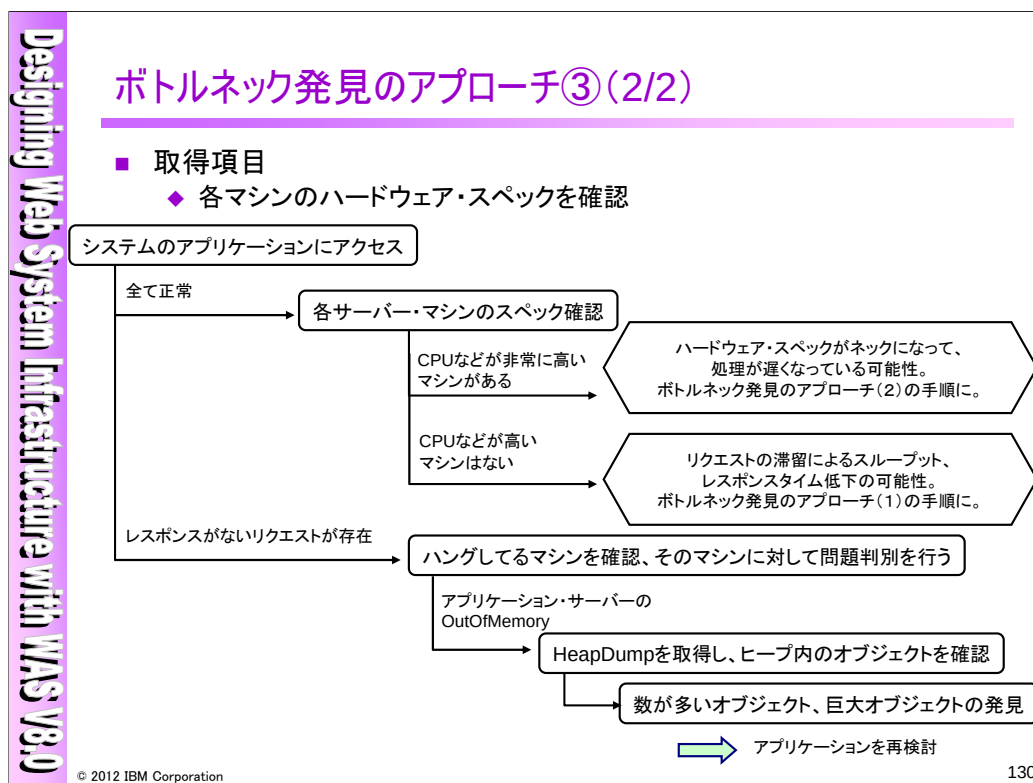
- 検証
 - ◆ マシンのハードウェアに異常はないかを突き止める

© 2012 IBM Corporation

129

飽和点に達した後にスループットが急激に落ちる場合があります。

この場合は不適切なチューニングや処理により、システムの処理能力が限界値を大きく超えてしまったためにダウンやハングが発生した可能性が考えられますので、まずはマシンのハードウェアに異常はないかを確認します。



まずはシステムに異常がないことを確認するために、システムのアプリケーションにアクセスし、全ての経路で正常に処理がされていることを確認します。全て正常であれば、各サーバー・マシンのスペックを確認し、CPUやメモリー、ディスク/I/Oが異常に高くなっているマシンがあれば、それらがボトルネックになっている可能性がありますので、「ボトルネック発見のアプローチ(2)」の手順に移ります。逆にCPUに全くの余裕がある場合には、リクエストの滞留によるパフォーマンス低下が考えられますので、「ボトルネック発見のアプローチ(1)」の手順に移ります。

レスポンスでエラーが返る、もしくは返事がないリクエストが存在した場合は、ハングもしくはダウンしているマシンがある可能性がありますので、マシンを再度調査し、ログなどを見て問題判別を行います。アプリケーション・サーバーのOutOfMemoryが発生している場合は、HeapDumpを取得し、ヒープ内のオブジェクトを確認し、問題があるオブジェクトに関わるアプリケーションの再検討やJVMヒープサイズの増加の検討を行います。

Designing Web System Infrastructure with WAS V8.0

ワークショップ、セッション、および資料は、IBMまたはセッション発表者によって準備され、それぞれ独自の見解を反映したものです。それらは情報提供の目的のみで提供されており、いかなる参加者に対しても法律的またはその他の指導や助言を意図したものではなく、またそのような結果を生むものでもありません。本講演資料に含まれている情報については、完全性と正確性を期するよう努力しましたが、「現状のまま」提供され、明示または暗示にかかわらずいかなる保証も伴わないものとします。本講演資料またはその他の資料の使用によって、あるいはその他の関連によって、いかなる損害が生じた場合も、IBMは責任を負わないものとします。本講演資料に含まれている内容は、IBMまたはそのサプライヤーやライセンス交付者からいかなる保証または表明を引き出すことを意図したもので、IBMソフトウェアの使用を規定する適用ライセンス契約の条項を変更することを意図したものでなく、またそのような結果を生むものでもありません。

本講演資料でIBM製品、プログラム、またはサービスに言及していても、IBMが営業活動を行っているすべての国でそれらが使用可能であることを暗示するものではありません。本講演資料で言及している製品リリース日付や製品機能は、市場機会またはその他の要因に基づいてIBM独自の決定権をもっていつでも変更できるものとし、いかなる方法においても将来の製品または機能が使用可能になると確約することを意図したものではありません。本講演資料に含まれている内容は、参加者が開始する活動によって特定の販売、売上高の向上、またはその他の結果が生じると述べる、または暗示することを意図したものでも、またそのような結果を生むものでもありません。パフォーマンスは、管理された環境において標準的なIBMベンチマークを使用した測定と予測に基づいています。ユーザーが経験する実際のスループットやパフォーマンスは、ユーザーのジョブ・ストリームにおけるマルチプログラミングの量、入出力構成、ストレージ構成、および処理されるワークロードなどの考慮事項を含む、数多くの要因に応じて変化します。したがって、個々のユーザーがここで述べられているものと同様の結果を得られると確約するものではありません。

記述されているすべてのお客様事例は、それらのお客様がどのようにIBM製品を使用したか、またそれらのお客様が達成した結果の実例として示されたものです。実際の環境コストおよびパフォーマンス特性は、お客様ごとに異なる場合があります。

IBM、IBM ロゴ、ibm.com、AIX、DataPower、DB2、Rational、Tivoli、WebSphereは、世界の多くの国で登録されたInternational Business Machines Corporationの商標です。

他の製品名およびサービス名等は、それぞれIBMまたは各社の商標である場合があります。

現時点での IBM の商標リストについては、www.ibm.com/legal/copytrade.shtmlをご覧ください。

Linuxは、Linus Torvaldsの米国およびその他の国における登録商標です。

Windowsは Microsoft Corporationの米国およびその他の国における商標です。

JavaおよびすべてのJava関連の商標およびロゴは Oracleやその関連会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。

© 2012 IBM Corporation131