有向グラフの昇順探索に基づくWebシステムのボトルネック検出法
― パフォーマンス統合分析ツールとしての実装 ―

清水 淳也　山根 敏志　加藤 整　中村 理之

Detecting Web System Bottlenecks by Searching a Directed Graph Upward
-- Implementation as Integrated Performance Analysis Tool --

Junya Shimizu　Toshiyuki Yamane　Sei Kato　Tadayuki Nakamura

本論文では, 知識ベースに基づいてWebシステム全体にわたるボトルネックを自動的に検出する手法を提案する. この手法はまず, 計測ナビゲーション・グラフと呼ばれる表現法を用いて, 負荷テストの測定データの関連をグラフのノードとして表現する. そして, グラフの末端ノードから出発し, 判定ルールを適用しながらルート方向に探索を進め, ボトルネック要因を検出していくというものである. この手法によって, ルール・デザインの客観的基準を持つことができ, 体系的に新たな事例追加が可能となりチューニング事例の資産化を容易として, 同時に, 大量のログ・データをつき合わせてボトルネック原因を特定する作業が大幅に簡略化されるといった効率化を達成できる.

We have developed a method to automate the detection of bottlenecks in a Web system using inference rules in a knowledge base. Firstly load test measurement data are placed at nodes in a directed graph called a Measurement Navigation Graph in such a way that the relationships between pieces of the data can be traced. Then, search for bottlenecks proceeds from the lowest-level nodes toward the root as inference rules are applied. This method allows an objective criterion for designing rules, enables systematic addition of new cases of performance tuning to the inference engine as reusable assets, and at the same time significantly simplifies bottleneck detection work by eliminating the need for accumulation and analysis of a large volume of log data.

Key Words & Phrases：Webシステム, 負荷テスト, ボトルネック検出, 知識ベース, グラフ探索
Web systems, load test, bottleneck detection, knowledge base, graph search

1．はじめに

e-ビジネスの進展に伴い, ミッションクリティカルな業務がインターネットやWebテクノロジーにますます強く依存するようになったため, Webシステムの十分なパフォーマンスを安定的に供給することは企業にとって生命線ともいえる. そのため, Webシステムの高負荷時のパフォーマンスを測定することによって, サービスインの基準を満たしていることを確認し, ボトルネックを早期発見するための負荷テストの実施が必須となる. しかし, WebシステムはHTTPサーバー, アプリケーション・サーバー, データベース, ネットワークといった数多くの要素が絡み合って構築されるシステムであり, パフォーマンスを左右する要素が多岐にわたっているため, パフォーマンス・チューニングという観点からは, 極めて複雑なシステムである[1].

そこで, 負荷テストの手間を軽減するため, Microsoft® Web Application Stressツール, マーキュリー・インタラクティブ社のLoadRunner, IBM Rational® Performance Testerなどの各種の負荷テストツールが開発されている. 通例, これらのツールを用いて得られた測定結果に基づいて, Webシステムのエキスパートが専門的知識を駆使し, ボトルネックとなる箇所を割り出していく. しかし, そうして得られた過去のチューニング事例を体系的に資産化して, 他の場合に適用できるような方法論については, あまり議論されることはなかった. 実際, 現在の分析ツールの多くは, 定形の分析を行うか, その判断の多くをツール使用者に任せてしまうものが多く, 柔軟な拡張性を伴った自動分析ツールは, あまり見られない. 特にボトルネック検出を行うためには, 計測データを体系的に管理して, 一定の基準に基づいて分析を進めることが必要となる

提出：2003年8月29日　再提出：2004年10月20日

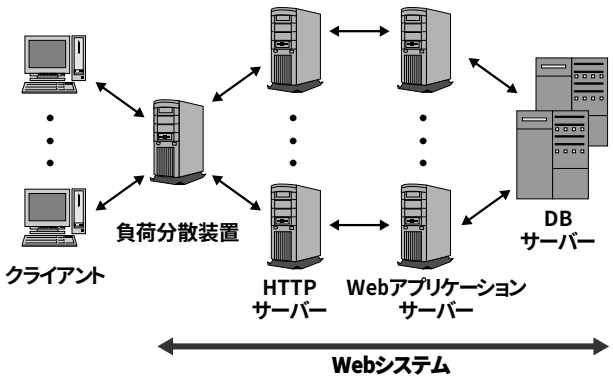

図1. Webシステムの構成

が, これまでボトルネック候補の検出は, 経験的な計測データ分析が行われがちで, 統一的なアプローチについて検討されることは少なかった.

そこで, 本論文では知識ベースを用いて過去の事例をグラフとそれに付随するルールの形に体系化し, このグラフを末端からルートに向かって探索することで, 判定ルールに基づく推論を行いながらボトルネック要因を検出するアルゴリズムおよびそのABLE (Agent Building and Learning Environment) [2] による実装について述べる. この手法によって, 過去の事例の蓄積, 新規事例の追加が容易になり, 同時に, ボトルネック検出の作業が大幅に効率化される.

2. Webシステムの概要

Webシステムは通常, **図1**に示されるような複数の階層構造をもっている. クライアントからのリクエストがインターネットを介してWebシステムに到達すると, 負荷分散装置によって各HTTPサーバーへ処理が割り振られる. HTTPサーバーではすべてのユーザーに共通な静的コンテンツに対するリクエストに対しては自分で処理し, ユーザーごとに異なる動的なコンテンツの処理には後段のアプリケーション・サーバーに処理を依頼する. アプリケーション・サーバーではその要求に応じてサーブレット・エンジン上でBeanを呼び出す. このBeanがデータベース・アクセスを実行し, 必要なデータを取得し, 動的にHTMLコンテンツを作成する. そして, 例えば, HTTPサーバーの最大スレッド数, JVMの最大ヒープサイズ, Webアプリケーション・サーバーの最大接続数, データベース接続プールサイズといったパラメータが絡み合って全体のパフォーマンスを決定する [3].

3. ボトルネック検出のアプローチ

前章で述べたとおり, Webシステムを構成する各コンポーネントあるいはサブ・システムのボトルネック分析に必要となる情報の組み合わせは多岐にわたり, 複雑性が増している. したがって, より効率的にパフォーマンス分析を行うためには, 蓄積ログ・パターンから自動的に分析を行い, ボトルネックへの対応策を提示できることが望ましい. ボトルネック検出を自動化するためには, 過去のボトルネック事例を蓄積した知識ベースを用い, その上で何らかの推論アルゴリズムに基づいて, ボトルネックの原因を探るというアプローチが考えられる.

そうしたアプローチを取るためにはデータの関連を表現する手段が必要となるが, パフォーマンスのボトルネック分析において, 汎用性の高いものとして典型的に用いられるのが, 有向グラフを使って計測データの依存性を表現する方法である [4] [5]. 本論文では, 文献 [4] で提案されている計測ナビゲーション・グラフ (Measurement Navigation Graph; MNG) をパフォーマンス分析のルール適用手順の背景として用いるので, まず簡単にMNGについて述べる.

MNGとは有向非巡回グラフの一種で, ノードは計測変数を表し, それらのノード間を結ぶアークは親子変数間の関連を表している. 実際の例を**図2**に示す. すなわち, MNGを用いれば, システム全体の応答時間の構成要因としてサーバー1の応答時間とサーバー2の応答時間が考えられ, さらに各サーバーの応答時間はCPU使用率やメモリー使用率に影響され, またCPU使用率は, さらにユーザー・アプリケーション用, システム用, そしてアイドル用の各使用率に分解できる, といった計測データの関係を階層的に表現できる. 上の例では, 子ノードに対する親ノードが一つのものを示したが, データの種類によっては親ノードが二つ存在する, つまり, 子ノードの計測値が複数の上位パフォーマンス・ノードと関連する場合も存在するため, ツリーではなくグラフと呼んでいる. また, アークの重みは, しきい値分析 [6] [7], ボトルネック分析,

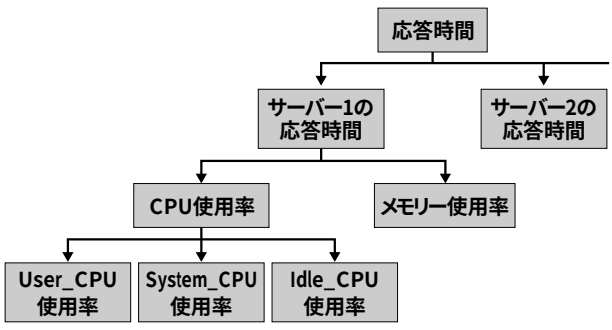

図2. MNGの例

相違箇所分析, 相関分析などの手法[8]を用いて計算することができる.

このMNGを用いて, 従来のアーク重み計算手法の特徴も考慮した定式化によって定量的にパフォーマンス分析を行うアルゴリズムが文献[5]で検討されている. この方法は, MNGの各アークの重みを計測データに基づいて計算することで, グラフのルートからの蓄積重みが大きくなるパスを探索することで, 最もボトルネック要因になる候補をピックアップするものであり, システム細部の計測が可能で, システム・ソフトウェアの内部構成がわかっている場合には有効な方法である. しかし, 本論文が対象としているように構成が複雑でかつ細部までのデータ取得が必ずしも可能ではない場合, あるいはシステムや計測対象リソースに変化がある場合には, 考察対象となるMNGの構造が変化するため, MNG自身の再構築とアークの重みを, 場合に応じて再計算する必要がある.

一方, パフォーマンス診断については, 従来からエキスパート・システムの研究がなされてきたが[9], その対象となっているシステム構成は固定されているか, ある程度わかっていることを想定していることが多かったため, システム構成変更時への対応も柔軟に行うのは難しいものと考えられる. また, 適用するルールを階層レベルに分けてパフォーマンス分析を行う方法が文献[10]で提案されているが, そのルール適用手順は経験的であり, その背景にはMNGのようなデータの関連を表現するものが考察対象とはなっていないため, 適用順序を体系的に考慮することは難しい.

4. グラフ昇順探索に基づくボトルネック検出のルール・デザイン

前章で述べた問題を回避するため, 本論文では, MNGを末端ノードから上位レベルに向かって再帰的にルール適用を繰り返していく方法を提案する. まず, **図3**のようなMNGとノードに付随するルール構造を考える.

多くの場合, ボトルネックの兆候を検出する際に用いるCPU使用率に関する末端ノードをスタート地点として, **図4**のようなチェック・ルール1によってパフォーマンス分析を開始する.

このようにサーバー・ソフトウェアのグラフ内で末端ノードである各種CPU使用率を分析スタート地点とし, そのノード階層での診断を行い, 問題が発見されない場合は階層を一つ上げて, CPU使用率と同一階層に位置づけられるメモリー使用率など他のパフォーマンス要因ノードへ分析対象を広げるという手順を繰り返し, 最終的には 各サーバーの応答時間ノードの階層に至って終了する.

次に, スタート階層レベル自体を一つ上げて, チェック・ルール1で必要となる計測データよりも多くのデータを用いつつ分析を行う. すなわち, CPU使用率ノー

```
If User_CPU Usage is the highest Then
        Investigate CPU ability for the focusing application
        If the above check is no problem Then
                Up investigating levels to check other resource
capacity of system such as memory

ElseIf System_CPU Usage is the highest Then
        Check the number of thread and process
        If the above check is no problem Then
                Up investigating levels

ElseIf Idle_CPU Usage is the highest Then
        Check physical capacity relevant to I/O
        If the above check is no problem Then
                Up investigating levels

Otherwise
        Check the profile of application code
```

図4. チェック・ルール1

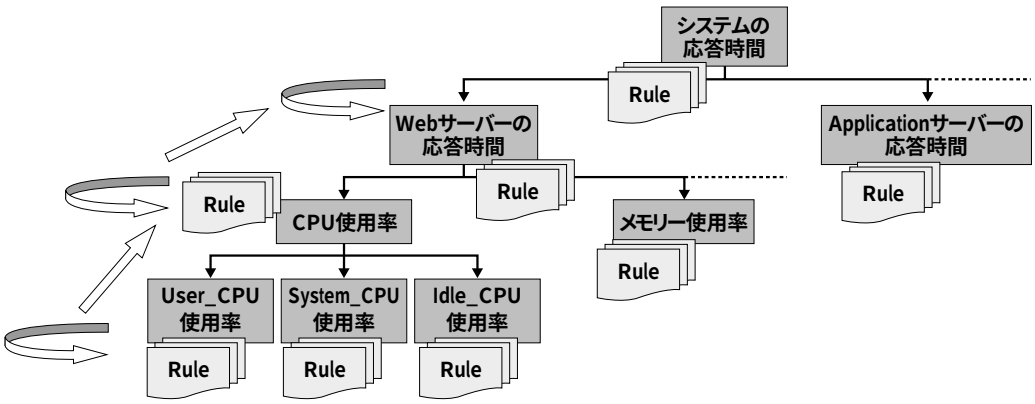

図3. 使用するMNG表現とルールの組み合わせ

```
If  Sudden downs of CPU usage exist Then
        Up investigating levels
Else
        Check other nodes on same levels
```

図5. チェック・ルール2

```
If  Response time increases & Throughput is plateau Then
        Check other servers & Check own children 's node
```

図6. チェック・ルール3

ド・レベルで, 計測データを時系列的に利用しながら**図5**のようなルールで分析を行う.

すなわち, CPU使用率ノード・レベルで使用率のデータに落ち込みが存在する場合, グラフの階層を上がって, 他のサーバー・ソフトウェアの応答時間ノードに関するチェックを行う. 他のサーバー・チェックに移行した場合は, 移行先でのチェック・ルール1による分析結果を利用する. もし使用率の落ち込みがなく, 階層を上がる必要がない場合は, CPU使用率ノードと同じ階層にある他のノードに付随するルール・チェックを行う. これは, 図には示していないが, 例えば, ガーベッジ・コレクションの頻度など応答時間に影響を及ぼす要因ノードに関するルール・チェックを行うことを意味している.

同様に, 分析階層レベルをさらに一つ上げて, クライアント数を増やした場合の応答時間計測データといった, 上述の二種類のチェック・ルール適用時よりも多くのデータを用いつつ, **図6**のルールによってサーバーの応答時間ノード・レベルでの分析を行う.

このチェック・ルールは, 上がるべき階層がないので, これまでのものとは少し異なる. すなわち, スループットは一定であるが, 応答時間が増加していくような場合は, 同一ノード・レベルの他のサーバー・チェックへ移行する. その場合は移行先のチェック・ルール1やチェック・ルール2の結果を利用するか, 同時に, 自分の子ノードに関するチェック・ルール1やチェック・ルール2の結果を利用する, あるいはそれらの結果に問題がなければ, 子ノードに関するリソースの物理容量についてチェックを行う.

このように, 末端ノードから有向グラフの逆方向にルール適用を再帰的に繰り返す手順を採用することによって, リソース追加やシステム構成に変化が発生してMNGの構造が変化しても末端ノードからの再帰手順に新しいノード部分を追加するだけで, 分析ツールのアルゴリズム自体に大きな変更は生じないようにすることができる. ノードだけではなく, MNGの階層も3段に限らず細分化していくことも可能である. さらに, 新たな追加ルールをデザインする場合も, MNG構造の中で位置づけることによって, 冗長なルール・デザインを回避することが可能となる. また, 計測データの収集程度に応じて, 再帰する分析階層ノードを適当な段階で停止させることが容易であるため, 必ずしもすべてのデータの計測が完了していなくても, ある程度のボトルネック分析が可能となる利点もある.

5. パフォーマンス統合分析ツールとしての実装

前章で述べたボトルネック検出手法を実装するには, システムを構成する各コンポーネントのパフォーマンス・データや各種パラメータ設定値が同時に必要となる. 我々は, これらのデータを効率よく取得, 管理する環境をパフォーマンス統合分析ツールとして構築した. この環境により, 負荷テストに要するワークロードを大幅に削減することができる.

5.1 負荷テストの実施環境

図7は我々の負荷テスト環境のGUIを表している. パフォーマンスの測定に際しては, Web パフォーマンス・ツール(WPT)[1]を用いて仮想的な負荷を発生させ, 一定時間システムに負荷を与え続ける. これらの個々の測定を本論文ではランと呼ぶことにする. さらに, 複数のパラメータを同時に変化させたとき, システム全体のパフォーマンスがどのように振る舞うかを調べる, という一連のランをまとめ, これをテストと呼ぶことにする. **図7**のGUIの下部は, ランおよびテストの実施記録をツリーとして表している. 一方, このGUIの上部はWebシステムの構成情報を表し, ホスト名やハードウェアの情報, サーバー名などを入力できる.

次に, Webシステムの各種パラメータの自動収集に

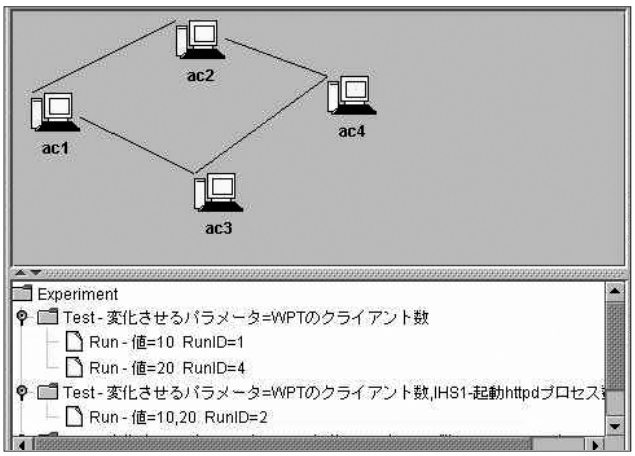

図7. 負荷テスト実施環境のGUI

[1] WPTはIBMが開発したWebテスト・ツール

ついては, 以下の方法を取った.

- IHS（IBM HTTP Server）：httpd.confを構文解析,
- WAS（WebSphere® Application Server）：XMLConfig.sh/wscp.shでリポジトリーをエクスポートし解析.
- DB2®：db2 set, getコマンドでdb2systm,SQLDBCONから抽出.

パフォーマンス・データの取得に関しては, 各ソフトウェアに標準で添付されているモニター・ツールを同時に起動し, すべてのデータを一度に取得し, 一つのデータ・リポジトリーに格納する方法を取った. **表1**に主な測定項目と使用するツールを示した. この方法によって, テスト実施者のワークロードを削減するだけでなく, 各ホスト, 各サーバーのデータを一元的に扱うことが可能になり, データの解析が容易になる. 例えば, すべてのランの時系列データをグラフ表示させることにより, テスト実施者はテストが正常に行われたか, 妥当なストレスがサーバーにかかったかを確認することができる. さらに, このすべてのパフォーマンス・データの集合を様々な切り口からグラフ表示することによって, 複数のサブ・コンポーネントのパラメータに同時に依存するようなパフォーマンスの変化を直観的に把握することもできる. 例えば, IHSの最大接続数とWASの最大スレッド数を同時に変えたときのクライアント側の応答時間の変化を調べるといったグラフ表示が簡単に行える.

5.2 ABLEによるルール実装

次に, ボトルネック検出の推論部分であるが, 前章で述べたMNGの昇順検索に基づくルール・デザインに適した実装ツールの一つがABLEである[2]. ABLEとは, Java Beanに基づく学習・推論エージェント構築環境であり, オートノミック・コンピューティングのコア・テクノロジーの一つに位置づけられている. ABLEにおいては, if-thenルール, ファジー推論などの主要な推論アルゴリズムが（ABLE）Beanと呼ばれる形で部品化されており, 再利用が可能となっている. さらに, ユーザーが定義した新しいBeanを追加することもでき, 柔軟な拡張可能性を備えている. また, リレーショナル・データベースからのデータをエージェントに入力するBeanも備わっているので, リポジトリーからのインターフェース共通化も達成できる. したがって, 前章で述べたようなMNGの構造変化などに伴うルールの追加も, Beanの追加という形で容易に実現できる.

表1. パフォーマンス測定項目の一部

項目	使用するツール
HTTPリクエストの応答時間	WPTの測定データを利用
HTTP serverの応答時間	WPTの測定データを利用
WASの応答時間	WASのトレース機能
CPU使用率	VMSTAT
ネットワークトラフィック	SAR/NETSTAT
I/O使用率	IOSTAT

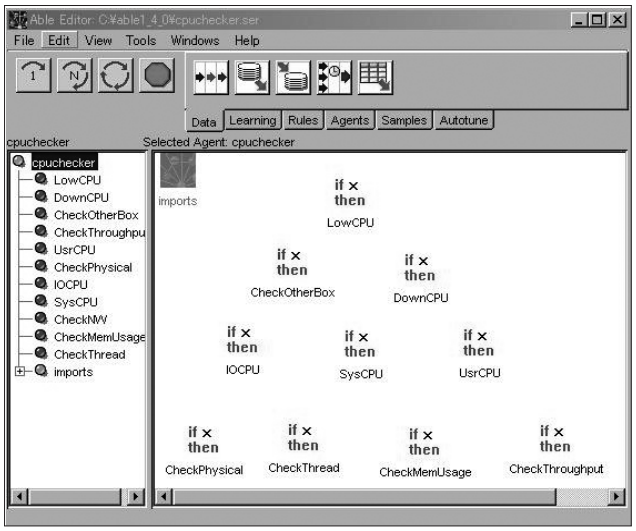

図8. ABLEによるルールの実装. 下から上へ推論が進行する.

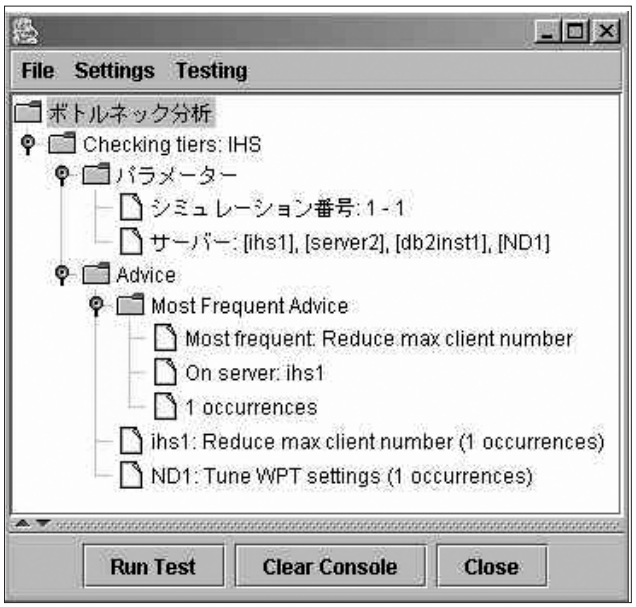

図9. ボトルネック分析の一例

従来の分析ツールのように個別のシステムに応じて作り込まれた独自のデータベースや知識ベースを使用している限り, それらのデータベースが将来にわたって, また, 別システムにおいて使用される可能性は低い. これらの知見を資産として再利用していくためには知識ベースの構成要素が, ある程度の拡張性とコンポーネント性を備えていることが重要である観点からもABLEによる実装は提案手法に適している. **図8**に, ABLEによる実装例を表す.

さらに, **図9**に4.1で述べた推論アルゴリズムを適用した結果の一例を示す. ボトルネック分析に用いるランや注目するサーバーなどの表示に加えて,「アドバイス」の部分にパフォーマンス改善のためのアドバイスが示される.

6. パフォーマンス・チューニングにおける効果

システムの高い可用性を実現する際に, 状態を監視して状況を把握し, 適切な動作や処置を行うためには, 管理者によるログの観測および分析が必要となる. しかし, ハイ・ボリュームWebサイトなどのシステム・ログは一日の量だけでも数百メガ・バイトあるいは数ギガ・バイトになるので, パフォーマンスが上がらない場合に, 手作業でサブ・システムのログ・データをつき合わせて原因を特定することは不可能になりつつある. また, オンデマンド環境下では, システムを構成する各コンポーネントあるいはサブ・システムのボトルネック分析に必要となる情報の組み合わせも多岐にわたり, 複雑性が増しており, この傾向は今後続くものと考えられる. 例えば, 10台程度のサーバーより構成される中規模Webシステム構築プロジェクトで, 19通りのシナリオで負荷テストを繰り返し, テスト結果の分析だけで1週間程度を要したという例も存在する.

一方, 蓄積ログ・パターンから自動的に分析を行ったり, システムへの対応策あるいはアドバイスを提示する必要がある場合, 提案システムにおいては, 一つのシナリオに基づくテストの分析にかかる時間は十数分程度である. さらに, 従来はこうしたボトルネック・チューニングに関する知見が再利用可能な形で蓄積されることは少なく, 同様のシステムのボトルネック調査を行う場合にも多くの人件費と日数を再びかける場合が多かった. しかし, 提案システムを用いれば, ルール・ベースでの知見の蓄積も容易であるため, 過去のボトルネック事例と類似ケースの調査にかかる人件費と日数を極めて削減できるのである.

7. おわりに

本論文では, 知識ベースに基づいてWebシステムのボトルネック検出を自動化するルール・デザインのアプローチについて述べた. 我々のアルゴリズムは, MNGの末端ノードから順番に上位ノードに向かって分析ルールの適用を繰り返していくことで, 測定データからボトルネック要因の推論を進めていくというものである. さらに, ABLEを用いて知識ベースを実装することにより, その提案アプローチの特徴である新規事例やルールの追加が容易となっている統合パフォーマンス分析ツールの実装を行った. このツールにより, ボトルネック検出に要していたコストを大幅に削減することが期待できる.

謝辞

サマー・インターンシップ・スチューデントとして実装の一部を手伝ってくれたマサチューセッツ工科大学のChang Sheさんに感謝します.

参考文献

[1] WebSphereパフォーマンス・チューニングと問題判別 http://www-6.ibm.com/jp/software/websphere/developer/performance/v5/index.html

[2] J.P. Bigus et al, *ABLE: A toolkit for building multiagent autonomic systems, IBM Systems Journal*, Vol.41, No.3, pp.350-371, 2002.

[3] Webシステムのボトルネック回避, Nikkei System Integration, 2004.2.

[4] R.F. Berry and J.L. Hellerstein, *A Unified Approach to Interpreting Measurement Data in Performance Management Application, Proc. IEEE 1st International Workshop on Systems Management*, pp.81-89, April, 1993.

[5] J.L. Hellerstein, *A General-Purpose Algorithm for Quantitative Diagnosis of Performance Problems, Journal of Network and Systems Management*, Vol.11, No.2, June 2003.

[6] D.R. Irvin, *Monitoring the performance of commercial T1-rate transmission service, IBM Journal of Research and Development*, Vol.35, No.5/6, pp.805-814, September/November 1991.

[7] K. Itoh and T. Konno, *An Integrated Method for Parameter Tuning on Synchronized Queueing Network Bottlenecks by Qualitative and Quantitative Reasoning, IEICE Trans. Inf. & Syst.*, Vol. E75-D, No.5, pp.635-647, September 1992.

[8] J.L. Hellerstein, *A Comparison of Techniques for Diagnosing Performance Problems IN Information Systems, Proc. ACM SIGMETRICS 94*, pp.278-279, May 1994.

[9] A.E. Irgon, A.H. Dragoni, Jr., T.O. Huleatt, *FAST: A Large Scale Expert Systems for Application and System Software Performance Tuning, Proc. ACM SIGMETRICS 88*, pp.151-156, 1988.

[10] B. Samadi, *TUNEX: A Knowledge-Based System for Performance Tuning of the UNIX Operating System, IEEE Trans. Software Engineering*, Vol.15, No.7, pp.861-874, July 1989.

日本アイ・ビー・エム株式会社
東京基礎研究所
主任研究部員

清水 淳也 Junya Shimizu

［プロフィール］
1998年，日本アイ・ビー・エム入社．東京基礎研究所にて，信号・画像処理システム，オートノミック・コンピューティングなどの研究を経て，現在，ビジネス・トランスフォーメーションに関する研究に従事．工学博士．
jshimizu@jp.ibm.com

日本アイ・ビー・エム株式会社
東京基礎研究所
副主任研究員

加藤 整 Sei Kato

［プロフィール］
2002年，日本アイ・ビー・エム入社．東京基礎研究所にて，オートノミック・コンピューティングに関する研究，パフォーマンス・モデリング，パフォーマンス最適化に関する研究に従事．
seikato@jp.ibm.com

日本アイ・ビー・エム株式会社
東京基礎研究所
副主任研究員

山根 敏志 Toshiyuki Yamane

［プロフィール］
2000年，日本アイ・ビー・エム入社．東京基礎研究所にて，光通信用符号理論に関する研究，Webシステムのパフォーマンス・エンジニアリングに関する研究に従事．
TYAMANE@jp.ibm.com

日本アイ・ビー・エム株式会社
テクニカル・コンピテンシー．Web技術 課長

中村 理之 Tadayuki Nakamura

［プロフィール］
1992年日本アイ・ビー・エム入社．システム・エンジニアとして，多くのお客様のシステム構築プロジェクトに参画．現在は Web技術担当として，Webアプリケーション開発に関わる技術エリアの情報発信およびプロジェクト・サポートに注力している．