May 12, 2006
RT0697
Computer Science 5 pages

Research Report

Automatic determination method for machine in performance trouble in distributed system

Sei Kato, Toshiyuki Yamane and Takahide Nogayama

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

分散系システムにおける性能問題箇所特定の自動化

加藤　　整† 　山根　敏志† 　野ヶ山尊秀†

† 日本アイ・ビー・エム株式会社東京基礎研究所
〒 242-8502 大和市下鶴間 1623-14
E-mail: †{seikato,tyamane,nogayama}@jp.ibm.com

あらまし　分散系システムにおいて性能問題箇所を自動的にする特定する手法を提案する. 本手法はトランザクションの混合比率の変化やインテンシティの増加など外部環境の変化によって変化しない性能指標であるサービスデマンドに着目し, その値の変化を監視することで, マシンレベルで性能問題判別を行う. 本手法をネットワークベースで実装し, 実験環境で検証実験を行った結果, その有効性を確認した.

キーワード　性能問題判別, 分散系システム, 性能管理

Automatic determination method for machine in performance trouble in distributed system

Sei KATO†, Toshiyuki YAMANE†, and Takahide NOGAYAMA†

† IBM Research, Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato-shi,Kanagawa-ken, 242-8502 Japan
E-mail: †{seikato,tyamane,nogayama}@jp.ibm.com

Abstract　A new approach for performance problem management in distributed system is proposed. The method detects and localizes the machine in trouble automatically by estimating and monitoring a performance metric, service demand, which is not affected by the external environment changes, such as transaction mix change and intensity increase. We implemented this method as a networkbase performance problem determination system and the verification experiment showed its effectiveness in practice.

Key words　Performance problem determination, distributed system, performance management

1. はじめに

最近の分散系の情報システムは, システムを構成する機器は数十から数百台に達し, 更にそれらの上で複数のサーバーアプリケーションが稼動するなど, 多くのコンポーネントが協調し構成されている. この様に複雑化した分散環境においては, 性能問題の原因となる箇所はハードウェア系ではディスク故障, ネットワーク機器故障など, OS 並びにミドルウェア系では構成ミスやバグなど, アプリケーション系ではバグ, データの異常値混入や仕様不具合など幅広く散在し, 性能問題が発生した際に, どのコンポーネントが性能劣化の原因なのかを自動的に特定することが困難であり, 問題判別に多くの人的労力がかかっている.

また性能問題の特徴として, 問題の検知に時間がかかるという点がある. 特定のユーザーの特定のリクエストみの応答時間が劣化する場合などにおいては情報処理システムの利用者からの報告によって性能問題発生を初めて検知し, 再現性の確認に時間がかかる場合も数多く発生している.

ウェブシステムの障害管理の時間において回復作業にかかる時間は 20%であり残りの 80%が問題検知、判別にかかる時間であるという報告もある [1]. 従って如何に早く問題を検知し, 問題箇所を特定できるかが障害を短期間に修復するための必要条件となる.

このような問題に対して, 性能問題を診断し, 定量的に診断結果を返すシステムが提唱されている [2]〜[4]. これら提唱されている方法では計測ナヴィゲーショングラフ (MNG) と呼ばれる有向グラフを用いて, 性能指標間の関連を定量的に表現し, 応答時間を根とし原因を葉とするグラフによってシステムをモデル化し, 葉の中から原因の候補を選択することにより, 問題箇所を特定している. この方法では, システム毎に MNG を構成し, アークの重みを測定しなければいけないことから使用範囲が小規模のシステムに限られ, 複雑な分散環境において充分機能することが難しく, 自動問題判別まで至っていない.

他方, 応答時間やキュー長といったシステムの”性能特性”の時間変化を解析することにより, 構成やチューニングパラメータ, スケジューリングアルゴリズムなどの”性能ファクター” の変化を検知する方法が研究されている [5]. この研究では変化点を検出すると同時に, 変化の原因の候補も推定可能になることから, 性能問題の判別も行えることが期待される. しかし, 性能ファクター間に非線形な相関があることにより, 性能特性の時間変化だけから間接的に変化した性能ファクターを推測することは成功していない.

本稿では, かかる従来技術, アプローチの問題点に鑑み, システムに依らず成立する普遍的な性質に基づき, 最終的な根本原因までは特定できないまでも, 原因箇所を含むマシンというレベルで特定することで, 問題箇所を局所化し, システム技術者が問題判別を行うことを容易にする実現可能なシステムを提案する.

2. サービスデマンドによる問題判別

応答時間や資源利用率はインテンシティの増加やトランザクションの混合比率の変化に応じて変化する. ユーザーやシステム技術者にとっては, 応答時間や資源利用率の値そのものの増加が問題であり, それらの指標に閾値を設け, 閾値超過を監視することには意味があろう. しかし前述のようにそれらの指標は環境の変化に応じて変化するものであり, 応答時間や資源利用率が閾値を超過したということが即ち性能問題の発生を意味しない.

本研究のアイディアはトランザクションの混合比率やインテンシティといった環境の変化に依存しない値を推定し, その値を監視することで, 性能問題の検知と判別を行うことである.

2.1 サービスデマンドの推定

監視開始時間 T からある時間間隔 ΔT_t で監視終了時刻 $T+\sum_{t=1}^{m}\Delta T_t$ まで m 点のデータをサンプリングすることを考える. 間隔 ΔT_t はその間の観測量が統計的に定常と見做せる限りにおいて, 一定であってもいいし, 変更されてもいい. 実際の情報システムにおいては観測量は日ごとに変動するため観測間隔を 24 時間とするとその観測区間内におけるデータは統計的に非定常であるが, 例えば観測間隔を 1 分とすればその観測区間内に観測されたデータは統計的に定常であることが期待でき, 24 時間観測することで約 1000 点の独立なサンプルを取得できるという点がポイントである. 監視区間 $\left[T+\sum_{t=1}^{j-1}\Delta T_t, T+\sum_{t=1}^{j}\Delta T_t\right]$ での観測量を添字 j をつけて表記し, その監視区間を区間 j と呼ぶことにする. 区間 j におけるマシン $M_k(1 \leqq k \leqq l)$ のビジー時間を b_{jk}, その区間にマシン M_k でサービス $S_i(1 \leqq i \leqq n)$ が呼ばれた回数を a_{jik}, サービス S_i のマシン M_k での真のサービスデマンドを d_{ik} と表すと, ユーティライゼーション律よりそれらの間には以下の関係式,

$$b_{jk} = \sum_i a_{jik} d_{ik} \tag{1}$$

が成立する [6], [7]. 実際の実験データに当てはめた場合計測誤差が発生するため, 以下の線形モデル,

$$b_{jk} = \sum_i a_{jik} d_{ik} + \epsilon_{jk} \tag{2}$$

を導入する. 但しここで ϵ_{jk} は区間 j でのマシン M_k におけるビジー時間 b_{jk} の計測誤差であり, 以下の条件,

$$< \epsilon_{pq} >= 0, \quad < \epsilon_{pq}\epsilon_{rq} >= {\sigma_q}^2 \delta_{pr}, \quad \mathrm{N}(0, \sigma_q^2) \tag{3}$$

に従うと考えられる. $< \cdot >$ は $\cdot$ の区間に関するアンサンブル平均, σ_q はマシン q での観測誤差の標準偏差, δ_{pr} はクロネッカーのデルタを表す.

この時, あるマシン M_k に対して, 誤差の 2 乗和 $Q = \sum_j \epsilon_j^2$ を最小にするようにサービスデマンド d_i を推定することで得られたデータから最も精度良く推定することができ, その推定値 $\hat{\boldsymbol{d}}$ は以下の正規方程式,

$$\boldsymbol{A}^t \cdot \boldsymbol{A}\hat{\boldsymbol{d}} = \boldsymbol{A}^t \boldsymbol{b} \tag{4}$$

の解として求めることができる. 但しここで, $\boldsymbol{A} = (a_{ji}), \hat{\boldsymbol{d}} = (\hat{d_1}, ..., \hat{d_n})^t, \boldsymbol{b} = (b_1, ..., b_m)^t$ であり, 特定のマシン M_k に着目していることから添字 k は省略している. また $\hat{\cdot}$ は $\cdot$ の推定量を意味する. 今, 得られた m 個のデータセットからサービスデマンド d_i を推定したい訳だが, 短い観測期間のデータだけでは計画行列 $\boldsymbol{A}$ の列ベクトル $\tilde{\boldsymbol{a}}_j = (a_{j1}, ..., a_{jn})$ の向きの変化は期待されず, 精度良くサービスデマンドを求めることは望めない. 24 時間という長い時間スケールにおいてはベクトルの向きは変化することが予想され, かつサービスデマンドは時間によって変化しないマシン固有の値であるので, そのような時間スケールで長い期間, 観測したデータを用いることによって精度よくサービスデマンドを推定することが出来ると期待される.

2.2 残差分析による問題検知

2.1 によれば充分長い時間, システムを監視することにより精度よくサービス毎のサービスデマンドが推定できるようになる. このことは逆にサービス毎のサービスデマンドの時間変化を解析することによってサービス毎の性能問題の検知を行おうとすると, 検知までに長い時間を要してしまうことを意味する. 検知遅延時間を短くするために考えうる最良の策は, サービスレベルの性能問題検知を諦め, ホストレベルで性能問題検知を行うことである. 式 (2) は, トランザクションの混合比率やインテンシティの変化に依らず成立するものであり, 従って, 逆に, ある区間 j であるマシン M_k において式 (2) が成立しない場合には, どのサービスかは特定できないが, 区間 j でマシン M_k のあるサービスの真のサービスデマンド d_{ik} が変化していることを意味する. この様にしてサービス毎の真のサービスデマンドそのものを随時, 観測できなくても, 式 (2) の正否を判定することで, ホストレベルで性能問題の発生を自動的に検知, 判別可能になる. 式 (2) の正否は, 例えば以下の残差分析によって判定可能である. まず初めに, ある一定のトレーニング期間のデータ $b_{jk}, a_{jik}, (0 \leqq j \leqq m')$ を用いて,$\hat{d}_{ik}, \hat{\sigma}_k^2$ を決定する. 推定誤差分散 $\hat{\sigma}_k^2$ は $\hat{\sigma}_k^2 = \sum_{j=1}^{m'}(b_{jk} - \sum_i a_{jik}\hat{d}_{ik})^2/m'$ として求まる. 次にサービスデマンドの推定値 $\hat{d}_{ik}$ とトレーニ

ングラン期間以降のデータ $b_{jk}, a_{jik}(m' < j \leqq m)$ から式 (2) を用いて残差 $r_{jk} = b_{jk} - \sum_i a_{jik}\hat{d}_{ik}$ を算出する. 式 (2) の正否を判定するには, 随時, r_{jk} と $\hat{\sigma}_k$ を比較すればよく, 例えば判定の閾値を $\hat{\sigma}_k$ の 3 倍と定義すれば区間 j でマシン M_k において以下の関係式,

$$|r_{jk}| > 3 \times \hat{\sigma}_k \tag{5}$$

が成立したら, 区間 j でマシン M_k において性能問題があったと検知, 判別することができる.

サービスデマンドの推定値が真の値の変化に迅速に追随する場合には, 残差には変化が余り観測されないものの, サービスデマンドの推定値が変化するので, サービスデマンドの推定値を監視することで問題発生の検知が可能である. 一方, 真のサービスデマンドが変化しても, 推定値が迅速に変化しない場合には, 残差に急峻な変化が観測されるので, 残差を監視することによって, 問題発生の検知が可能になる. このようなことから, サービスデマンドの推定値と残差を共に監視することによって, 推定値が真の値に迅速に追随するか否かに関わらず, 問題を検知することが可能になる.

式 (2) の正否の幾何学的解釈についてコメントしておこう. 式 (2) は観測点 $\exists k \forall j(a_{j1k}, a_{j2k}, \ldots, a_{jnk}, b_{jk})$ が空間 $\boldsymbol{R}^{n+1}$ において, 原点を通過する超平面上に"厚さ"$\hat{\sigma}_k$ で分布することを意味する. ここで超平面の各軸に沿った偏微分係数が対応するサービスのサービスデマンドの推定値 $\hat{d}_{ik}$ となり, 式 (5) はある観測点 $\exists k \exists j(a_{j1k}, a_{j2k}, \ldots, a_{jnk}, b_{jk})$ が超平面の"板"から外れて位置することを意味する.

3. ネットワークベースによる実装

我々は, ネットワークベースでサービスデマンドを求める手法を開発した. この手法に依れば, 情報システムを構成するマシン間を流れるネットワークパケットをキャプチャーすることによりサービスデマンドを推定することが可能になる. ネットワークパケットは例えばマシンが接続されたネットワークスイッチのミラーポートなどから取得でき, ネットワークパケットを取得するだけで済むことから, 監視対象の情報システムを構成するサーバーマシンに何らかのプログラムを仕込む必要がなく, 従って大規模構成のシステムに関しても容易に対応できるというのが本手法の利点である. 以下にその手法を示す.

分散系の情報処理システムにおいては, システムを構成するマシン上のあるサーバープログラムのトランザクションが, 他のマシンのサーバープログラムのトランザクション (親トランザクション) から呼ばれ, また別のマシンのサーバープログラムのトランザクション (子トランザクション) を呼ぶことで処理が進められる. ある一定期間, これらマシン間でやり取りされるトランザクションのパケットを取得, 解析することにより, その期間に発生した各トランザクション $TXN_p(1 \leqq p \leqq m)$ に関し,

開始時刻 $T_{start,p}$,
終了時刻 $T_{end,p}$,
トランザクションの属するサービス $S(p)$

を知ることができる. これらよりある観測区間 j でのサービスの呼び出し回数 a_{jik} は

$$\begin{aligned} a_{jik} = \#\Big\{ & p | S(p) = S_i, \\ & T + \sum_{t=1}^{j-1} \Delta T_t \leqq T_{start,p} < T + \sum_{t=1}^{j} \Delta T_t, \\ & T + \sum_{t=1}^{j-1} \Delta T_t \leqq T_{end,p} < T + \sum_{t=1}^{j} \Delta T_t \Big\} \end{aligned} \tag{6}$$

と求まる. 式 (1) に依れば, ある観測区間 j におけるビジー時間 b_{jk} とその観測区間内におけるサービス S_i のサービスの呼び出し回数 a_{jik} が判れば各サービスのサービスデマンド d_{ik} を推定することが出来る. 各サービスの呼び出し回数 a_{jik} は前述のように求まるので, 当該観測区間におけるビジー時間さえ判れば, 各サービスのサービスデマンドが推定できると考えられる. 以下ビジー時間を推定する手法について説明する.

今, ある観測区間において, あるマシンに注目した場合, そのマシンでのビジー時間, 即ちそのマシンの計算機資源が使用されている時間を求めたいわけだが, ここで少なくとも 1 つ以上の処理が存在している時間が処理に費やされた時間であるという点に着目すると, サービス規律に依らずビジー時間は以下のように, マシンに滞在する処理の数（系内人数）をカウントし系内人数が 1 以上である時間を積み上げることで推定できると期待される. 以下の擬似プログラムで示すアルゴリズムで系内人数をカウントし, 系内人数が 1 以上である時間の和を求めることでビジー時間を計算できる.

```
double calcBusyTime (){
    int 系内人数 = 0;
    double busyTime = 0.0;
    while(観測区間) {
        if (親より処理要求を受信) ++ 系内人数;
        else if (親へ処理完了を送信) −− 系内人数;
        else if (子へ処理要求を送信) −− 系内人数;
        else if (子より処理完了を受信) ++ 系内人数;
        else doNothing;

        if (系内人数が 0 から 1 へ増加) {
            timeStamp = 時刻;
        }else if (系内人数が 1 から 0 へ減少) {
            busyTime+ =diff(timeStamp, 時刻);
        }
    }
    return busyTime;
}
```

このようにして, 複数の処理が同時に存在している場合においても, 計算機資源のサービス規律に依らずビジー時間を推定できることがこの手法の利点である. 実際の情報システムに適用した場合, 次のようないくつかの課題が発生する. 例えば, 親

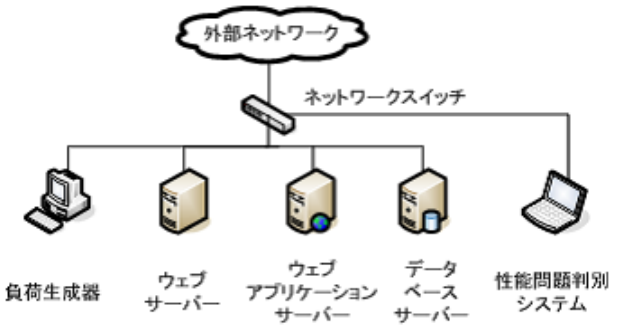

図 1　検証実験環境

トランザクションより処理要求を受信した後に複数の子トランザクションの処理要求を送信する場合や, 親トランザクションの処理要求を受信せずに子トランザクションに処理要求を送信する場合には系内に親トランザクションが存在しないにもかかわらず，子トランザクションが発生した場合に，系内人数が負になってしまうことがある．

前者の場合，複数の子トランザクションの応答を待たなければ，親トランザクションは終了することはない．従って子トランザクションが呼び出されているにもかかわらず，それを呼んでいる親トランザクションがいないという状況にはならない．このため，親トランザクションが系内にいる場合に限り，複数の子トランザクションが発生し系内人数が負になっても，問題はなく, 系内人数に負を許すことで解決できる．

後者のような自発的なリクエストは, トランザクションの呼び出し関係を推定する際に, 親がいないトランザクションとして検出できるので, それを基に判断し, 発生した際に系内人数に加算しないことで解決する．

4. 検 証 実 験

図 1 に検証実験を行ったシステムを示す．対象とする情報システムはウェブサーバー, ウェブアプリケーションサーバー, データベースサーバーの 3 層からなるウェブシステムであり, ウェブアプリケーションサーバー上でオンライン株取引アプリケーションが稼動している．またユーザーリクエストをエミュレートするために, クライアントマシンから擬似的なリクエストが送信されている．この検証実験においては 3 つのマシンが繋がれたスイッチのミラーポートからそれらのマシンを通過するトラフィックをコピーし, tcpdump コマンドを用いてパケットデータをダンプした．得られたパケットデータから呼び出されたサービス毎の開始, 終了時刻を推定し, マシン毎のビジー時間とマシン毎でサービスが呼び出された回数を推定し, 結果を基にサービスデマンドを推定している．

本検証実験では一定のページ遷移シナリオで監視対象システムにリクエストを送ることにより負荷をかけているので, トランザクションの混合比が時間的に変化しない．そこで, ここではウェブサーバー上の全てのサービス, アプリケーションサーバー上の全てのサービス, データベースサーバー上の全てのサービスをそれぞれ 1 つに纏めて 3 つのサービスに再定義し, それらのサービスのサービスデマンドを求めている．即ち先述の表記

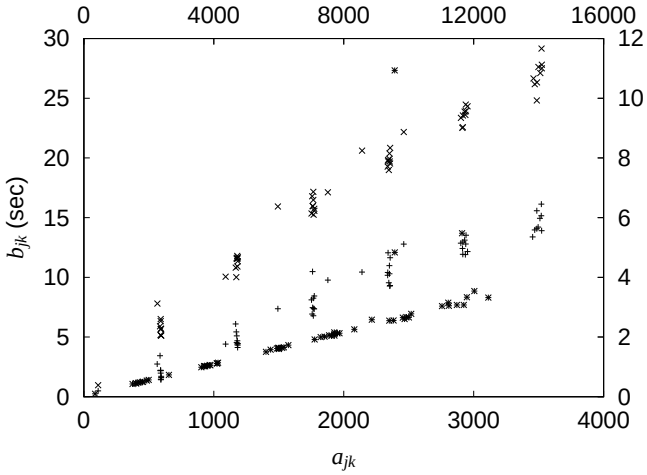

図 2　サービスの呼び出し回数とビジー時間の関係を示すグラフ．インテンシティを変化させた時の各監視区間のデータを, サービスの呼び出し回数を横軸に, ビジー時間を縦軸にとってプロットしている．+はウェブサーバー (下横軸, 左縦軸), × はウェブアプリケーションサーバー (下横軸, 左縦軸), *はデータベースサーバー (上横軸, 右縦軸) の値を示す．

法において $l = 3, n = 1$ の場合に相当する．

図 2 にインテンシティを変化させた時の, サービスの呼び出し回数とビジー時間の関係を示す．ここでは $n = 1$ であるので, 観測点は空間 $\boldsymbol{R}^2$ において, 原点を通過する直線上に分布する．パケットロスによる間欠的なビジー時間のバーストも見られるが, ほぼビジー時間とサービスの呼び出し回数は比例の関係にあり, インテンシティの変化に依らず, 式 (2) が成立することが解る．

次にデータベースサーバーに擬似的な性能問題を注入した場合に問題の発生の検知と問題のマシンの判別が可能かどうかを検証するために, 監視対象ウェブシステムに一定シナリオで一定の負荷をかけつつ, (a) 監視開始 16 分後にデータベースサーバーのデータベースのインデックスをオンラインで削除させる, (b) 監視開始 10 分後にデータベースサーバーのデータベースのバッファープールサイズをオンラインで 2000 ページから 50 ページへ変化させる実験を行った．データベースのインデックスとはデータベース内のデータに迅速にアクセスするために表に対して定義される索引であり, インデックスを作成することで全表走査をする必要がなくなり, 検索時間を短くすることが可能になる．またバッファープールとはアクセスされたデータベースのデータをキャッシュするメモリであり, より大きいバッファープールサイズを設定するとディスクに対して物理的に入出力アクセスすることが無くなることから, アクセス速度が向上する．

実験結果を図 3,4 に示す．図において時間変化しない直線はそれぞれ対応するサーバーの上方管理限界を示しており, (a) では監視開始後の 10 分間のデータ, (b) では監視開始後 5 分間のデータから推定誤差分散を計算し, 変化点検出には品質管理のアプローチであるシューハート管理図の 3 シグマ [8] を管理限界として採用している．また頑健性を与える為に, 3 回連続して

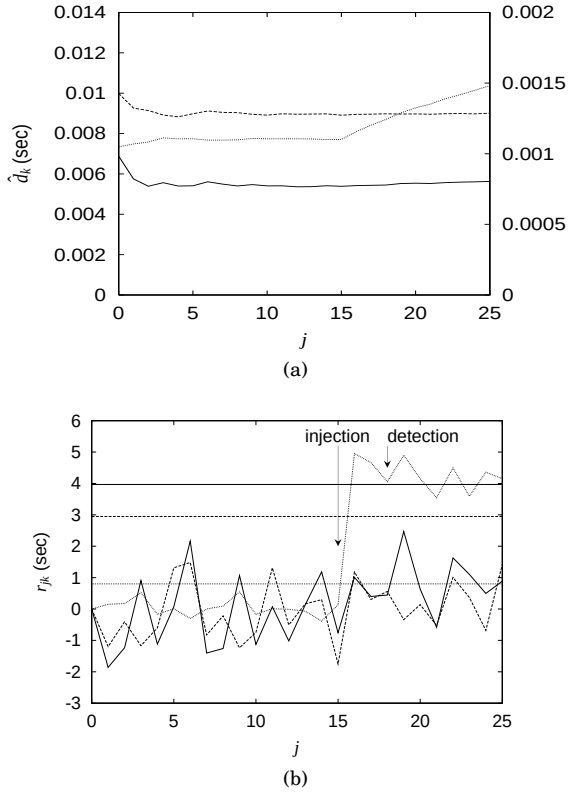

図 3　監視開始後 16 分後にデータベースサーバーのインデックスを削除することにより, 性能問題を発生させた際の, (a) 各サーバーのサービスデマンドの推定値, (b) 残差の時間変化を示す時系列グラフ. 実線はウェブサーバー, 破線はウェブアプリケーションサーバー, 点線はデータベースサーバーの値を示す.

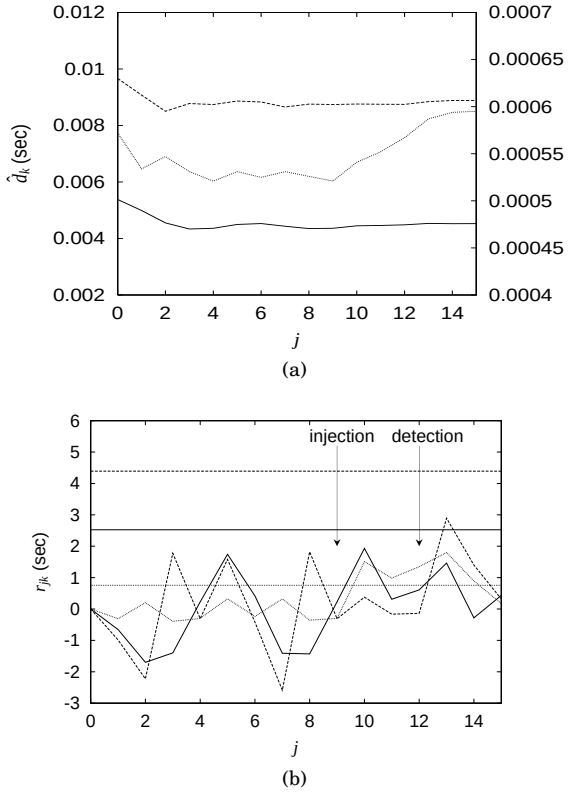

図 4　監視開始後 10 分後にデータベースサーバーのバッファープールサイズを変化させることにより, 性能問題を注入した際の, (a) 各サーバーのサービスデマンドの推定値, (b) 残差の時間変化を示す時系列グラフ. 実線はウェブサーバー, 破線はウェブアプリケーションサーバー, 点線はデータベースサーバーの値を示す.

上述のルールを違反した際に警告を上げている. ここでは監視間隔が 1 分であり, 従ってこの時の警告遅延は 3 分である. どちらの実験においてもウェブサーバー, ウェブアプリケーションサーバーの残差には変化が見られず, 障害を注入したデータベースサーバーのみ障害注入後に残差が増加し, 上方管理限界を超えるのが解る. またサービスデマンドの推定値は緩やかに変化するものの, 残差は急激に変化していることから, 残差の変化を監視することで, 警告時間の遅延無く, 性能問題の検知が可能になっている. このように本検証実験結果では, 問題発生から 3 分後に問題を検知し, データベースサーバーにおいて性能問題が発生していると判別できており, 性能問題箇所を自動的に特定する上で本手法が有効であることが確認できる.

5. ま と め

分散系システムにおいて性能問題の原因箇所をマシンレベルで自動的に特定する手法を提案した. 本手法はトランザクションの混合比率やインテンシティといった環境変化に依らない性能指標であるサービスデマンドを推定し, 残差分析によって性能問題の検知と判別を行うことを特徴とする. 3 層からなるウェブシステムに障害を模擬的に注入することで検証実験を行った結果, 問題の検知と判別が可能であることが示され, 本手法の有効性が確認できた.

トランザクションの混合比率が変化する場合や, 更に多くのマシンから構成されるシステムで複数のマシンで同時に性能問題が発生する場合など, より複雑な場合での検証に関しては現在実験中であり, 結果は別の稿で報告する.

文　献

[1] “Realigning IT for service delivery,” Banker Middle East, vol.36, June 2003.
[2] J.L. Hellerstein, “A comparison of techniques for diagnosing performance problems in information systems,” Proceedings of the 1994 ACM SIGMETRICS conference on Measurement and modeling of computer systems, pp.278-279, 1994.
[3] J.L. Hellerstein, “A general-purpose algorithm for quantitative diagnosis of performance problems,” Journal of Network and Systems Management, vol.11, no.1, March 2003.
[4] J. Shimizu, T. Yamane, S. Kato, and T. Nakamura, “Detecting web system bottlenecks by searching a directed graph upward,” IBM ProVISION, vol.44, pp.83-89, Winter 2005.
[5] R. Berry, and J.L. Hellerstein, “An approach to detecting changes in the factors affecting the performance of computer systems,” ACM SIGMETRICS Performance Evaluation Review, vol.19, no.1, pp.39-49, May 1991.
[6] P.J. Denning, and J.P. Buzen, “The operational analysis of queueing network models,” ACM Computing Surveys, vol.10, no.3, pp.225-261, September 1978.
[7] D.A. Menasce, L.W. Dowdy, and V.A. Almeida, Performance by Design: Computer Capacity Planning by Example, Prentice Hall PTR, 2004.
[8] W.A. Shewhart, Statistical Method From the Viewpoint of Quality Control, Dover Publications, 1939.