論　文

ネットワークデータを用いた分散システムにおける異常検出

鹿島　久嗣[†]　津村　直史[†]　井手　剛[†]　野ヶ山尊秀[†]
平出　涼[†]　江藤　博明[†]　福田　剛志[††]

Network-Based Problem Detection for Distributed Systems

Hisashi KASHIMA[†], Tadashi TSUMURA[†], Tsuyoshi IDÉ[†], Takahide NOGAYAMA[†], Ryo HIRADE[†], Hiroaki ETOH[†], and Takeshi FUKUDA[††]

あらまし　本論文で，我々はネットワークデータを用いた分散システムの異常検出システムを提案する．その中で我々は，(1) 対象とするシステムに負荷をかけることなく，ネットワークを流れるデータからトランザクションの情報を復元する枠組み，(2) 分散したサービスの間の動的な依存関係をトランザクションの実行時間の包含関係を手がかりに，オンライン EM アルゴリズムに基づいて直接的な依存関係を発見することのできる競合モデル，及び，(3) 発見された依存関係情報に基づいた様々な粒度での指標を用いた階層的な異常検出の枠組みを提案する．また，プロトタイプシステムを用いた実験によって，我々の提案するシステムが，実際にシステムに仕組まれた障害を発見できることを確認する．

キーワード　異常検出，ネットワークデータ，分散システム，EM アルゴリズム

1. はじめに

今日の典型的な e-ビジネスの基盤は，分散した Web サーバーやアプリケーションサーバー，データベース，ストレージサーバー，ネットワークデバイスといった多様なコンポネントから構成されている．そして，各々のコンポネントは各自の持つ様々な機能を**サービス**としてお互いに提供している．そして，これらのサービスは自らの機能を果たすために，お互いに強く依存しあっている．例えば，HTTP サーバーはユーザーに対し，ある URL で指定される Web ページをサービスとして提供しているといえる．もしも，そのページがサーブレットを含むものであった場合，ユーザー（ブラウザー）からのリクエストを受け取った HTTP サーバーは，サーブレットを実行するためにアプリケーションサーバーを呼ぶ．この場合，その URL で指定される Web ページへのリクエストは，サーブレットに**依存している**ということができる．同様にして，サーブレットが EJB を呼ぶ場合には，サーブレットは EJB に依存し，さらに EJB がデータベースに対する問い合わせを行う場合には，EJB はデータベースに依存することになる．図 1 は，このような依存関係を図示した例である．このように，依存関係の情報は，サービスを頂点，それらの間の依存関係を辺とした有向グラフとしてみることができる．

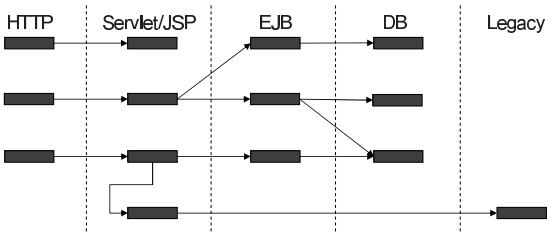

図 1　分散システムにおける依存関係の例
Fig. 1　Example of dependencies in a distributed computer system

Gupta *et al.* [1] らは，多くのコンポネントが持っている共通のパフォーマンス監視用の仕組みを用いて集めたトランザクションの実行期間からサービス間の依存関係を自動的に発見する手法を提案した．しかしながら，彼らのアルゴリズムはワークロードが高くなる

† 日本 IBM 株式会社 東京基礎研究所，神奈川県
IBM Tokyo Research Laboratory, 1623-14, Shimotsuruma, Yamato-shi, Kanagawa, 242-8502 Japan
†† 日本 IBM 株式会社 大和ソフトウェア研究所，神奈川県
IBM Yamato Software Laboratory, 1623-14, Shimotsuruma, Yamato-shi, Kanagawa, 242-8502 Japan

と，多くの偽の依存関係を出力してしまうという欠点がある．加えて，彼らの手法では，直接の依存関係と間接的な依存関係を区別できないという欠点がある．この性質は依存関係のグラフを複雑にしてしまうため，システムの理解や障害箇所の特定が困難になる．一方，図 1 のように直接の依存関係のみによって構成される依存関係のグラフはシンプルであり，これらの目的により適しているといえる．

本論文では，我々は 2 つのアイデアに基づき，前述した問題を解決する．まず，サービスの到着間隔と実行時間から推定される確率モデルを用いて，偽の依存関係の割合を軽減する方法を提案する．そして次に，直接の依存関係だけを発見するための競合モデルと呼ぶモデルを提案し，EM アルゴリズム [2] に基づきこのモデルのパラメータ推定を行う方法を提案する．また，我々のモデルは，あるサービスが他のサービスを直接何回呼び出したかを推定することができ，これは依存関係の有無のみを推定しようという Gupta らの定式化の一般化となっている．また，我々はトランザクションの実行期間データを収集するための情報源として，ネットワークを流れるデータを用いるため，対象となるシステムのパフォーマンスに殆ど影響を与えることなくシステム全体の情報を利用することができる．

依存関係はしばしばシステム管理者ですら気づいていなかったシステムの振る舞いを理解するのに役に立つ．また，システムに異常が発生した際には，これらの情報は異常の検知や，その致命度の解析，あるいは異常個所の特定といった目的にも利用できる．ハードウェアの故障やソフトウェアのバグなどによって，あるサービスがうまく働かなくなった場合，依存関係の情報から障害の影響を受けたサービスを知ることができる．

さらに，本論文では，発見された依存関係を基にした異常検出の方法を論じる．個々の依存関係など，特定の値の変化に基づいて異常検出を行う方法については, [3], [4] をはじめ，いくつかの研究例が存在する．一方，井手ら [5] はサービスやシステム全体といったより大きな視点から障害を検出するための手法を提案している．そこで，本論文では，システムの微視的な特徴量と巨視的な特徴量を用いることで，異常検出から，異常箇所の特定へと結びつける階層的な解析を行うという階層的な異常検出の枠組みを提案する．この枠組みによって，障害箇所の絞り込みや，個々の箇所を独立に監視するだけでは大量に発生してしまう偽の異常報告を効果的にフィルタリングすることが可能になる．

以下の構成は次のとおりである．まず 2. 章において，システムにおける依存関係の定義を行う．次に，3. 章では，依存関係の発見問題と，それに対する我々のアプローチである，ネットワークを流れるデータを対象とした監視システムのアーキテクチャについて述べる．4. 章では，既存の依存関係発見手法である Gupta らのアルゴリズムを説明し，その問題点を指摘する．そして，それらの問題を解決する我々の新しい依存関係発見アルゴリズムを提案する．5. 章では，発見された依存関係に基づく，システムの異常検出の枠組みを説明する．6. では，実際のシステムを用いた実験により，提案手法の有効性を検証する．最後に 7. 章では，関連研究を紹介し，8. 章にて，結論を述べる．

2. 依存関係発見

1. 章において，依存関係を用いた分散システムにおける障害の検出と解析を動機付けたが，この章では，稼動中のシステムから依存関係を発見する問題を定義する．

Σ を対象のシステムにおける**サービス**の集合とする．それぞれの $S \in \Sigma$ は例えば，URL で指定される Web ページやサーブレット，あるいは EJB のあるメソッドやデータベースへのクエリなど，様々な粒度で提供される機能を指す．**トランザクション** T とは，サービスのインスタンスであり，T の属するサービス S と開始時刻 s および終了時刻 t の組によって $T = (S, [s, t])$ と定義される．ここで，$[s, t]$ を T の**実行期間**と呼ぶことにする．例えば S が，ある URL で指定される Web ページであった場合，s はそのページに対するリクエストの時刻，t はリクエストに対して Web ページの送信が終了する時刻というように定義できる．

トランザクション履歴 D とは，対象のコンピュータシステムに設置された計測機構を用いて得られるトランザクションの集合である．多くの計測手段が D としてモデル化されるデータを提供することができる．場合によって D は，含まれるトランザクションの終了時刻順にソートされているとする．

トランザクション $T_1 = (S_1, [s_1, t_1])$ の実行期間が，別のトランザクション $T_2 = (S_2, [s_2, t_2])$ の実行期間を包含しているとき，T_1 は T_2 を**包含する**といい，$T_1 \succeq T_2$ と表す．トランザクション T_1 が別のトランザクション T_2 を**呼び出す**とき，T_1 が T_2 に**直接に依**

存するという．また，そのような T_1，T_2 があるとき，これを $S_1 \Rightarrow S_2$ と表す．そして，T_1 は T_2 の**親トランザクション**であり，T_2 は T_1 の**子トランザクション**であるという．各トランザクションは，高々1 つの親トランザクションを持つことに注意する．T_1 が直接に T_2 に依存するか，あるいは，T_1 が，T_2 に依存する別のトランザクション T_3 に依存するとき，T_1 は T_2 に**依存する**といい (再帰的な定義となっているところに注意する)，$T_1 \rightarrow T_2$ と表す．

同様にして，トランザクション $T_2 = (S_2, [s_2, t_2])$ に直接に依存するようなトランザクション $T_1 = (S_1, [s_1, t_1])$ が存在するとき，サービス S_1 はサービス S_2 に**直接に依存する**という．また，$T_2 = (S_2, [s_2, t_2])$ に依存するようなトランザクション $T_1 = (S_1, [s_1, t_1])$ が存在するとき，サービス S_1 はサービス S_2 に**依存する**といい，$S_1 \rightarrow S_2$ と表す．サービス S_1 がサービス S_2 に依存するとしても，D における全ての S_1 に属するトランザクションが S_2 に属するトランザクションに依存するとは限らない点に注意する．また，S_1 のトランザクションが S_2 のトランザクションを複数回呼び出すこともありうる．後ほど定義するように，我々は直接の依存関係 $S_1 \Rightarrow S_2$ あるいは依存関係 $S_1 \rightarrow S_2$ の**強さ**を S_1 のトランザクションが S_2 のトランザクションを呼び出す確率，あるいは回数の期待値として定義する．$G(D)$ をトランザクション履歴 D から導かれる全ての依存関係の集合とする．依存関係は 2 項関係の集合であるため，$G(D)$ はグラフとして表現できる．これを**依存関係グラフ**と呼ぶ．

ここで我々は，次の重要な仮定を行う．我々は全てのトランザクションは**同期的**，すなわち，T_1 が T_2 を呼び出すとき，T_1 は必ず T_2 が終了するのを待ってから終了するとする．つまり，$T_1 \rightarrow T_2$ ならば $T_1 \succeq T_2$ が成立するとする．

3. 依存関係発見問題とネットワークデータからのトランザクション履歴の復元

ここまでのところでは，我々はどのトランザクションがどのトランザクションを呼び出していることを知っている，すなわちシステムにおける全ての依存関係は自明にわかるものと仮定してきた．

例えば，ARM (Application Response Measurement) [6] はアプリケーションの挙動を捉え，トランザクション間の真の呼び出し関係を知ることのできる API の集合である．しかしながら，ARM は対象となるアプリケーションがトランザクションごとに ARM の API を何度か呼び出す必要があるため，対象のシステムに負荷をかけてしまうという問題がある．加えて，ARM を使用可能にするために，アプリケーションのソースコードに改変を加えなければならない．従って，トランザクション間の呼び出し関係を直接計測するのはコストがかかりすぎ，多くの場合現実的ではなくなってしまう．

一方，各トランザクションの実行期間を計測すること，すなわちトランザクション履歴 D を得ることは比較的簡単で，対象のシステムにあまり負荷がかからない場合が多い．そこで，我々はトランザクション履歴からサービス間の依存関係を**推定する**というアプローチをとる．

まずは，稼動中の分散システムから如何にしてトランザクション履歴 D を復元するかを考える．Gupta ら [1] は，多くのコンポネントが持っている共通のパフォーマンス監視用の仕組みを定期的に呼び出すことで，トランザクションの実行期間を取得するアプローチを提案した．彼らの手法はノード内，すなわち 1 つのマシン内の依存関係を発見するのには効果的であるが，この方法を分散したノード間の依存関係発見に用いようとすると，依存関係発見のアルゴリズムを適用するためにデータを一箇所に集めなければならず，また，ノードの時計を高い精度で同期させねばならない．

一般的な企業システムにおいて，各ノードは別々のマシンにインストールされており，コンポネント間の通信はネットワークを介して行われる．我々はこの点に着目し，ネットワークを流れるデータをネットワークデバイスにおいて監視することで，トランザクションの実行期間を取得するというアプローチをとる．現在では，ルーター，スイッチ，ハブなどの多くのネットワークデバイスにおいて殆ど全て (あるいは，定義されたいくつかの) トラフィックを殆どパフォーマンスに影響を与えることなく取得することのできるミラーポートを備えている．そして，ミラーポートから取得したパケットデータを解析することで，トランザクションの開始時間，終了時間，およびサービスに関する情報を得ることができる．例えば，HTTP のトランザクションの開始時刻は ‘GET’ メソッドを含むパケットから，終了時刻は ‘HTTP/1.1 200 OK’ のメッセージを含むパケットからそれぞれ取り出すことができる．また，複数のノード間を行き交う情報を，ミラーポートに接続された 1 つのノードで取得できるため，デー

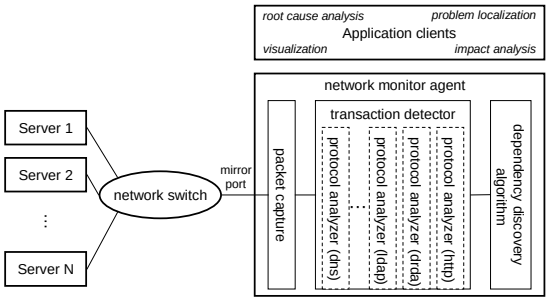

図 2　プロトタイプシステムの概念図
Fig. 2　A prototype system

タ収集と時刻同期のコストが削減される．さらに，パケットはリクエストの要求元の IP アドレスをも含んでいるため，これらを用いて親トランザクションの候補を限定するのにも利用することができる．(親トランザクションのサービスは，その IP アドレスで指定されるノードで動いていなければならない．)

以上の考えに基づき，我々は図 2 に示されるようなプロトタイプシステムを開発した．現在のプロトタイプでは，HTTP，DRDA，DNS，及び LDAP をサポートしているが，アーキテクチャ自体は，そのほかのプロトコルに対応したプロトコル解析モジュールを追加することにより，容易に拡張可能となっている．

4. 依存関係発見手法

この章では 3. 章で得られたトランザクション履歴 D から，トランザクションが同期的であるという仮定を利用して，依存関係を推定するための手法を考える．

4.1 従 来 手 法

Gupta らは [1] において，サービス間の依存関係 $S_1 \to S_2$ の強さを，以下で定義する 2 つの確率 p と r を用いて定義し，これをオンラインで計算できる依存関係発見アルゴリズムを提案した．

まず彼らは，S_1 のトランザクションが少なくとも 1 つの S_2 のトランザクションを包含する確率 p を

$$p = \sharp(S_1|S_1 \succeq S_2)/\sharp(S_1)$$

のように定義した．ここで，$\sharp(S_1)$ は D 中の S_1 のトランザクションの回数，$\sharp(S_1|S_1 \succeq S_2)$ は D 中の S_1 のトランザクションの中で，少なくとも 1 つの S_2 のトランザクションを包含しているものの個数であるとする．S_1 のあるトランザクションが S_2 のトランザクションを複数回包含していても，$\sharp(S_1|S_1 \succeq S_2)$ への寄与は 1 回と数えることに注意する．もしも S_1 と S_2 の間に**真の**依存関係があるならば，S_1 のトランザクションが S_2 のトランザクションを少なくとも 1 回は包含しているはずであり，従って，p の値が 0 より大きくなるはずである．

また彼らは，S_2 のトランザクションが少なくともひとつの S_1 のトランザクションに包含される確率 r を，

$$r = \frac{\sharp(S_2|S_1 \succeq S_2)}{\sharp(S_2)} \quad (1)$$

のように定義した．ここで，$\sharp(S_2|S_1 \succeq S_2)$ は D 中の S_2 のトランザクションの中で，少なくとも 1 つの S_1 のトランザクションに包含されているものの個数であるとする．

そして彼らは，これら 2 つの指標を組み合わせて $\max(p, r) + pr$ を依存関係 $S_1 \to S_2$ の強さを測る指標として提案している．Gupta らのアルゴリズムは全てのサービスの組 (S_1, S_2) について，依存関係の強さをオンラインで計算する．

Gupta らの手法はシンプルであるが，トランザクション履歴のみから自動的にサービス間の依存関係を発見する強力なツールである．しかしながら，我々は以下に示す 3 つの問題点を指摘することができる．

（1） ワークロードの高い状況では，依存関係を過大評価，すなわち，**偽の**依存関係を発見してしまう．
ワークロードが高くなるにしたがって，トランザクション同士の包含が偶然によって起こる確率が高くなる．つまり，S_1 が S_2 に依存しない場合でも，S_1 のトランザクションは S_2 のトランザクションを偶然包含してしまうことが起こる場合がある．

（2） 直接の依存関係と，間接的な依存関係が区別されていない．
この性質は依存関係のグラフを複雑にし，対象システムの挙動の把握や，異常個所の特定を困難にする．

（3） 依存関係の強さの指標の定義がアドホックである．
依存関係の強さとしては，より定量的な指標のほうが望ましいであろう．
次の章において，我々はこれらの問題に対処する新しい方法を提案する．

4.2 偽の依存関係の推定

この節ではまず，前節の最後で述べた，Gupta らの方法が依存関係を過大評価してしまうという問題を考察する．

2 つのサービス S_1 と S_2 の間に依存関係がないとする．言い換えれば，S_1 のトランザクションが少なく

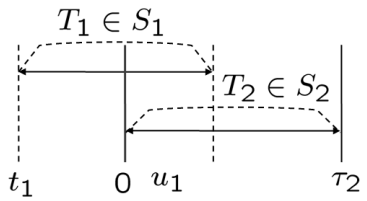

図 3　偶然による包含の確率の推定
Fig. 3　Estimation of accidental containments

とも 1 回の S_2 のトランザクションを呼び出す確率が 0 で，S_2 のトランザクションが S_1 のトランザクションに呼び出される確率が 0 であるとする．ワークロードが高くなるに従って，S_1 のトランザクションが少なくとも 1 回の S_2 のトランザクションを包含してしまう確率と，S_2 のトランザクションが少なくとも 1 回の S_1 のトランザクションに包含されてしまう確率が大きくなり，すなわち，Gupta らの方法が偽の依存関係を導いてしまう原因になる．そこで，偶然の包含の影響を抑えるために，まずは，偶然の包含が発生する確率を推定することにする．

$i = 1, 2$ に対し，サービス S_i へのリクエストが，ワークロード λ_i(リクエスト/秒) で到着すると仮定する．すると，S_i への 2 つの連続するリクエストの到着間隔 X_i は，平均 $E[X_i] = 1/\lambda_i$ と累積確率分布 $F_i(t) = Pr[X_i \leqq t] = 1 - e^{-\lambda_i t}$ をもつ指数分布に従う．同様に，S_i の実行時間 τ_i は，平均 $E[\tau_i] = 1/\mu_i$ と累積確率分布 $T_i(t) = Pr[\tau_i \leqq t] = 1 - e^{-\mu_i t}$ をもつ指数分布に従うとする．これらの仮定は，標準的なパフォーマンス解析では，一般的な仮定としてよく用いられる [7].

これらの仮定のもと，S_2 のあるトランザクション T_2 が少なくとも 1 つの S_1 のトランザクションに偶然包含されてしまう確率 $\psi(S_1, S_2)$ を推定する(注1)．簡単のため T_2 の開始時間を $t = 0$ とする．また，$t = 0$ において，実行中の S_1 のトランザクションの数を N とする．すると，N の確率分布はポアソン分布 $Po(\lambda_1/\mu_1)$，すなわち，

$$Pr[N = n] = \frac{\lambda_1^n}{\mu_1^n n!} e^{-\frac{\lambda_1}{\mu_1}}$$

に従う．いま，S_1 のトランザクションが n 個実行中であり，その中のひとつのトランザクションが時刻 t_1 に始まり，時刻 u_1 で終了するとする (図 3 を参照のこと)．すると，u_1 は平均 $1/\mu_1$ の指数分布に従い，T_1 が T_2 を包含しない確率は，以下のように書くことができる．

$$fail(\tau_2) = \int_0^{\tau_2} \mu_1 e^{-\mu_1 u_1} du_1 = 1 - e^{-\mu_1 \tau_2}$$

n 個のトランザクションの終了時刻はお互いに独立であるから，S_1 のトランザクションが n 個存在し，それらの全てが T_2 を包含しない確率は，

$$x_n = \int_0^{\infty} \frac{\lambda_1^n}{\mu_1^n n!} e^{-\frac{\lambda_1}{\mu_1}} fail(\tau_2)^n \mu_2 e^{-\mu_2 \tau_2} d\tau_2$$

のようになり，最終的に，

$$\psi(S_1, S_2) = 1 - \sum_{n=0}^{\infty} x_n, \tag{2}$$

が得られる．ここで，

$$x_n = \frac{\lambda_1^n e^{-\frac{\lambda_1}{\mu_1}}}{\prod_{k=1}^{n}(k\mu_1 + \mu_2)} = \frac{\lambda_1}{n\mu_1 + \mu_2} x_{n-1}$$

であるとする(注2)．

4.3　競合モデルによる直接的な依存関係の発見

この節では，前節の結果に基づき，2 番目の課題であった，直接の依存関係の発見問題を考える。既存手法 [1] が間接的な依存関係をも取り出してしまう原因は,「各トランザクションは高々1 つの親トランザクションしか持ち得ない」という制約を無視しているためである。そこで我々は，この制約を明示的にとりいれた競合的なモデル

$$\sum_i \rho(S_i, S_j) = 1 \quad (0 \leqq \rho(S_i, S_j) \leqq 1), \tag{3}$$

を採用する。ここで，$\rho(S_i, S_j)$ は，サービス S_i がサービス S_j のトランザクションにに直接的に依存する確率をあらわす(注3)。

我々はトランザクション履歴 D に対する最尤推定

(注1)：同様に，S_1 のあるトランザクション T_1 が少なくとも 1 つの S_2 のトランザクションを偶然包含されてしまう確率を計算することも可能であるが，後の議論では用いないため本論文ではその導出は省略する．

(注2)：$\psi(S_1, S_2)$ は無限の和を含んでいるため厳密な評価は困難であるが，実用的には適当な n まで計算すればよい．

(注3)：どのサービスにも依存されないサービスを統一的に扱うために，仮想的なサービス S_0 に属する仮想的な $T_0 = (S_0, [-\infty, \infty])$ を考える。

によってモデル推定を行うというアプローチをとることにする。まずは例を用いて考える。4つのサービス $\Sigma = \{S_1, S_2, S_3, S_4\}$ が存在するとし，S_1 のあるトランザクション T は，S_2 と S_3，それぞれ少なくともひとつのトランザクションに包含されており，S_4 のトランザクションにはまったく包含されていないとする。すると，このような状況が起こる確率は，次の二つの場合の確率の和である。

- T は S_2 のトランザクションに直接呼び出され，S_3 のトランザクションには偶然包含されている。また，S_4 のトランザクションに偶然包含されることはなかった。
- T は S_3 のトランザクションに直接呼び出され，S_2 のトランザクションには偶然包含されている。また，S_4 のトランザクションに偶然包含されることはなかった。

従って尤度は，$\rho(S_2, S_1)\psi(S_3, S_1)(1 - \psi(S_4, S_1)) + \psi(S_2, S_1)\rho(S_3, S_1)(1 - \psi(S_4, S_1))$ となる。ここで，$\psi(S_i, S_j)$ は，S_j のあるトランザクションが S_i のトランザクションに偶然包含される確率である。一般的に書くと，トランザクション履歴 D に対する対数尤度の和は，

$$L = \sum_{T \in D} \log \sum_{S \in C(T)} \nu(S, C(T)|T), \tag{4}$$

のように書ける。ここで，$C(T) \subseteqq \Sigma$ は，トランザクション T が包含されているトランザクションのもつサービスの（重複を許さない）集合とし，また

$$\nu(S, C(T)|T) = \rho(S, S(T)) \prod_{S' \in C(T) \setminus S} \psi(S', S(T)) \cdot \prod_{S'' \notin C(T)} (1 - \psi(S'', S(T)))$$

で，$S(T)$ は T のもつサービスとする。

我々の目標は，対数尤度の和 L を (3) の制約のもとで最大化することであるが，この最適化問題を解析的に解くことは困難である。従って，我々は Expectation Maximization (EM) アルゴリズム [2] に基づく反復解法を提案する。各反復ごとに必ず L が大きくなることが保証され，また，次に示すように L は局所最適解をもたないため，提案するアルゴリズムは，常に最適解を求めることができる。

定理: トランザクション履歴 D に対する対数尤度の和 (4) は局所最適解をもたない。

証明: L に対するヘッセ行列は半負定であるため。詳細は付録 1. を参照のこと。□

パラメータ推定は以下のように行う。Expectation ステップにおいて，現在の ρ に基づき，トランザクション T がサービス S_i のトランザクションに直接呼び出される確率は

$$\Pr(S_i|C(T), T) = \Pr(S_i, C(T)|T)/\Pr(C(T)|T) \tag{5}$$
$$= \frac{\delta(S_i \in C(T))\rho(S_i, S(T)) \prod_{S' \in C(T) \setminus S_i} \psi(S', S(T))}{\sum_{S \in C(T)} \rho(S, S_j) \prod_{S' \in C(T) \setminus S} \psi(S', S(T))}$$

のように計算することができる。ここで，δ は引数が真ならば 1 を，そうでないならば 0 を返すような関数であるとする。従って，**サービス呼び出し頻度** $\hat{\sharp}(S_i \Rightarrow S_j)$，すなわち S_i のトランザクションが S_j のトランザクションを呼び出した回数の期待値は

$$\hat{\sharp}(S_i \Rightarrow S_j) = \sum_{T \in D(S_j)} \Pr(S_i|C(T), T) \tag{6}$$

によって得られる。

Maximization ステップにおいては，パラメータの最尤推定値は，

$$\rho(S_i, S_j) = \frac{\hat{\sharp}(S_i \Rightarrow S_j)}{\sharp(S_j)}. \tag{7}$$

によって更新される。

$$\frac{\partial L}{\partial \rho(S_i, S_j)} = \sum_{T \in D(S_j)} \frac{\delta(S_i \in C(T)) \prod_{S' \in C(T) \setminus S_i} \psi(S', S_j)}{\sum_{S \in C(T)} \rho(S, S_j) \prod_{S' \in C(T) \setminus S} \psi(S', S_j)}$$

であることに注意すると，上記のステップを纏めて書くこともでき，$\rho(S_i, S_j)$ の更新則は，

$$\rho(S_i, S_j) \leftarrow \rho(S_i, S_j) \frac{1}{\sharp(S_j)} \frac{\partial L}{\partial \rho(S_i, S_j)} \tag{8}$$

となる。

従って，任意の初期パラメータから開始して，(6) と (7) を収束するまで交互に適用してやることで，最適なパラメータを求めることができる。

上記のやりかたは，D 全体を一括処理することを想定しているが，実際には D はストリーム形式で与えられ，トランザクションを逐次的にオンライン処理する必要がある場合が考えられる。その場合には，オンラ

イン EM アルゴリズム [8] を用いて，1 回のパラメータ更新には 1 つのトランザクションを用いるようにすることができる。トランザクション T が与えられると，親サービスの確率 (5) を現在の ρ に従って計算し，それを (6) に加えてパラメータを更新するようにすればよい。

Gupta らの方法が単に S_i が S_j に依存しているかどうかを考慮しているだけなのに対し，サービス呼び出し頻度 $\hat{\sharp}(S_i \Rightarrow S_j)$ は呼び出し回数の推定値を示しており，システムの挙動をより深く理解するための定量的な指標を提供する。各サービス間のサービス呼び出し頻度を行列の形で表したものを F とおく (個々の要素は $f_{ij} = \hat{\sharp}(S_i \Rightarrow S_j)$) ことにする。

次の章では，推定されたサービス呼び出し頻度行列に基づき，分散システムにおける問題検出へのアプローチを提案する。

5. 異 常 検 出

前章では，ネットワークを流れるデータから，分散システムにおける依存関係を高精度で抽出する方法を提案した．依存関係はそれだけでもシステムの構造を理解するために役に立つが，それ以外にも，ひとつの利用方法として，システムの異常検出に用いることが考えられる．依存関係はシステムの振る舞いをよく表現しているので，システムに障害が起こった場合，それらの多くは依存関係の変化として現れるであろうと考えられるためである．この章で我々はこの仮説に基づき，依存関係に注目した異常検出法を提案する．

前章で述べた手法により，ある時刻 τ(分) におけるサービス呼び出し頻度行列 $\mathsf{F}(\tau)$ を，$\tau - k$ 分から τ 分までの間のデータから計算するということを，k 分毎に行なうことによって，現時点 t までのサービス呼び出し頻度行列の系列 $\mathsf{F}(t), \mathsf{F}(t-k), \mathsf{F}(t-2k), \mathsf{F}(t-3k), \ldots$ が得られることになる．我々はこの系列から異常を検出する方法を考えることになる．

具体的な手法を考察するにあたり，まずは前章で求めたサービス呼び出しの行列の性質を調べてみる．まず，サービス呼び出し頻度がどのように振舞うかを調べる．図 4 の点線で示されたものは，ある HTTP サーバーへのリクエストがあるアプリケーションサーバーを呼び出している部分に対応する部分のサービス呼び出し頻度を，5 分毎に集計(注4) したものの時間変化を表す．システムは正常で，安定状態にあるにもかかわらず，この値はワークロードの変化に伴い激しく時間変化をしており，呼び出し回数そのものを異常検出に用いるのはあまり適当ではないことがわかる．

従って，システムの異常検出の際には f_{ij} をそのまま使うよりも，ワークロードの変化によるサービス呼び出し頻度の揺らぎを抑えるために，正規化された量 $h_{ij}(f_{ij})$ を使うほうが便利である．ここで，$h_{ij}(\cdot)$ は単調なスケーリング関数であるとする．

$h_{ij}(\cdot)$ の選び方としては，例えば，$h_{ij}(\cdot) = 1/\sharp(S_i)$ と定義することで，(i,j) 要素が

$$c_{ij} = \frac{\hat{\sharp}(S_i \Rightarrow S_j)}{\sharp(S_i)} \tag{9}$$

で与えられるような**サービス呼び出し割合**の行列 C を用いることが考えられる．c_{ij} は，S_i の 1 回のトランザクションにつき S_j のトランザクションが直接に呼ばれる回数の期待値を表しており，システムの働きを理解するのには非常に有効である．システムが定常状態のときには，c_{ij} は，多少のワークロードの変化に関わらず一定の値を示すことが期待できる．

また，別のスケーリング関数としては，(i,j) 要素が $h_{ij}(\cdot) = \ln(1+\cdot)$ で与えられるような，**対数サービス呼び出し頻度**の行列 K を用いることが考えられる．実験的には，対数変換はトラフィックのバースト的な性質を除去するのに効果的であることが知られている．このスケーリング関数は (i,j) には依存しないため，サービス間の頻度のバランスが保存されるという性質があり，こちらはシステム全体としての異常を発見する尺度として好ましいものとなっている．

図 4 で実線で示されているのは，サービス呼び出し割合の行列の要素である．サービス呼び出し頻度が強く時間変動する一方，HTTP サーバーへのリクエストは，1 回につきアプリケーションサーバーを 1 回呼び出すため，正常時にはサービス呼び出し割合は必ず 1 になる．このように正常時において，親サービスが子サービスを定数回呼び出す場合 (あるいはある程度の時間で平均すると定数回となるような場合) にはサービス呼び出し割合は安定した値となる．

5.1 個々の依存関係に基づく異常検出

以上の観察から，我々は，個々の正規化されたサービス呼び出し割合 c_{ij} を確率変数と考え統計的検定を

(注4)：後に 6. 章で述べる条件のもと，4. 章で提案したアルゴリズムを用い，システムが正常に動いているときのサービス呼び出し頻度の行列 F を 5 分毎に計算した．

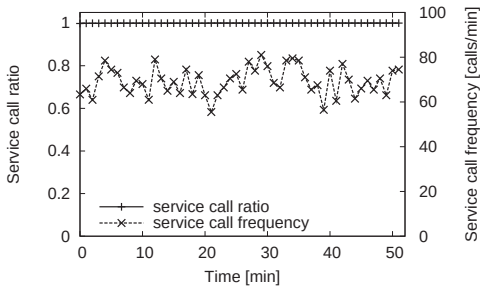

図 4　サービス呼び出し頻度とサービス呼び出し割合の比較
Fig. 4　Comparison between service call frequency and service call ratio

行なうことにより，個々の依存関係における異常を検出するためことができると考えられる．ここで用いる確率モデルとしては [9] で用いられているような時系列モデルや，各時点での値を独立であるとするモデル [10]〜[12] などが考えられるが，前節でみたような通常の設定では，システムが定常状態にあればサービス呼び出し割合は比較的安定していると考えられるため，ここでは単純に後者のモデルを考えることにする．なお，提案した c_{ij} が 0 以上の値をとるため，理論上はガンマ分布が自然であるが，実用上はガンマ分布の正規近似を行っても十分であり，ここでは単純に正規分布を用いることにする．

c_{ij} に対して，ある時刻 t における平均 $w_{ij}(t)$ と標準偏差 $\sigma_{ij}(t)$ の値は，よく知られたオンライン最尤推定の結果 [11] から，次のように計算できる．

$$w_{ij}(t) = (1-\beta)\cdot w_{ij}(t-k) + \beta c_{ij}(t), \qquad (10)$$

ここで β は忘却率である．

すると，以下の基準に従って，依存関係の異常を判定できる．

基準 1: もし時刻 t におけるサービス呼び出し割合 $c_{ij}(t)$ について，

$$|c_{ij} - w_{ij}(t)| > \sigma_{ij}(t) x_{\rm th}$$

が満たされた時，依存関係 $S_i \Rightarrow S_j$ には異常がある．ここで，$x_{\rm th}$ は

$$\int_{x_{\rm th}}^{\infty} duN(x) = p_c.$$

の解である．ただし，p_c は危険域境界 (< 1)，$N(x)$ は標準正規分布である．

5.2　サービス活動度に基づくサービス毎の異常検出

前節で述べた異常検出ルールは単純ではあるが，単一のサービス依存関係について起こるような障害について，多くのケースで有効である．しかしながら，この基準だけでは捉えられない障害も数多く存在する．まず，サービス呼び出し割合のいくつかは正常時でも不安定であり，図 4 のように一定値をとらないものも存在する．この場合，単純な閾値による検出法はうまく働かず，例えば，サービスの定義の中に定義されていない細かいパラメータが依存関係に影響するような場合，サービス呼び出し割合は不安定になりうる．

また，通常いくつかのサービス依存関係の時間変動は似通っており，互いに相関がある．そのため，仮に個々の依存関係に異常が観測されなくても，何らかのバランスの崩れとして異常が現れる場合がありうる(注5)．このような異常は個別のサービス依存度を見ていても認識しがたい．

そこで我々は，一段抽象的なレベルで，各々のサービスの異常を検出するために，Idé ら [5] の枠組みを用いる．以下，その概略を説明する．まず，サービス呼び出しの行列に対する特徴ベクトル $\boldsymbol{u}$ を次のように定義する．

$$\boldsymbol{u}(t) \equiv \arg\max_{\tilde{\boldsymbol{u}}} \left\{ \tilde{\boldsymbol{u}}^T \tilde{\mathsf{K}}(t) \tilde{\boldsymbol{u}} \right\} \qquad (11)$$

ただし，$\tilde{\boldsymbol{u}}^T\tilde{\boldsymbol{u}} = 1$．ここで，$T$ は転置を表す．ここでは，マルコフ遷移過程としての解釈の一貫性を保持する都合上 [5]，対称化した対数サービス呼び出し頻度 $\tilde{\mathsf{K}} = \mathsf{K} + \mathsf{K}^T$ を用いた．$\tilde{\mathsf{K}}$ は非負行列なので，たとえば $\tilde{k}_{2,3}$ が大きい値をとるならば，最大値の達成のためには，u_2 と u_3 が正の大きい値をとらねばならない．すなわち，あるサービス i が他のサービスを活発に呼んでいれば，$\boldsymbol{u}$ において i の重みは大きくなる．この意味で，この特徴ベクトルを，**サービス活動度ベクトル**と呼ぶことができる [5]．

重要な経験的事実として，F における強いランダム変動が，特徴ベクトル $\boldsymbol{u}$ においては非常に抑制されていることが挙げられる．このことから，個別のサービス活動度に対して，素朴に正規分布で確率モデルを考えてみる．もちろん一般にはサービス活動度同士の相関は有限の値をとるはずであるが，相関の効果はサービス呼び出し割合の行列に対する異常判定基準 1 によ

(注5)：典型的には，冗長化構成となっているサーバー同士のバランスの崩れが例として挙げられる．

り補われていると考えてよい.

ある時刻 t における，各サービス i の活動度 $u_i(t)$ について，平均 $w_i(t)$ と標準偏差 $\sigma_i(t)$ の値は，Eq. (10) と同様の式でオンラインに計算可能である.

基準 2: サービス活動度 $u_i(t)$ について,

$$|u_i(t) - w_i(t)| > \sigma_i(t) x_{\rm th}$$

が満たされた時，サービス i は異常である．ここで，$x_{\rm th}$ は

$$\int_{x_{\rm th}}^{\infty} du N(x) = p_c.$$

の解である．ただし，p_c は危険域境界 (< 1)，$N(x)$ は標準正規分布である.

5.3 システム全体の異常検出

次に，システムを全体的に見た場合の異常発見について述べる(注6)．活動度ベクトルの時系列について，システム全体としての異常度 $z(t)$ を次のように定義する.

$$z(t) \equiv 1 - \boldsymbol{r}(t)^T \boldsymbol{u}(t), \tag{12}$$

ここで，$\boldsymbol{r}(t)$ は時刻 t における典型的な活動パターンであるとする．$z(t)$ は，活動度ベクトルが時刻 t における典型的なパターンに直交する場合，$z(t)$ は 1 となり，一致する場合 0 となる.

$\boldsymbol{r}(t)$ を計算するために，まず $\mathsf{U}(t)$ を

$$\mathsf{U}(t) = [\boldsymbol{u}(t-k), \boldsymbol{u}(t-2k), \cdots, \boldsymbol{u}(t-Wk)] , \tag{13}$$

とおく．ここで W は考慮する時間枠の大きさ，$\boldsymbol{r}(t)$ は $\mathsf{U}(t)$ の最大固有値をもつ左特異ベクトルとする.

この異常度を，危険域境界のような普遍的な数値と関係付けるために，$\boldsymbol{u}$ についてのパラメトリックなモデルを考えてみる．$\boldsymbol{u}$ はいわゆる方向データであるから，エントロピー最大原理の見地からもっとも自然なモデルは次の von Mises-Fisher 分布である [13].

$$p(\boldsymbol{u}|\kappa, \boldsymbol{\mu}) = \frac{\kappa^{\frac{N}{2}-1}}{(2\pi)^{N/2} I_{\frac{N}{2}-1}(\kappa)} \exp\left(\kappa \boldsymbol{\mu}^T \boldsymbol{u}\right) \tag{14}$$

ここで $I_m(\cdot)$ は m 階の第 1 種の変形ベッセル関数，$1/\kappa > 0$ は角分散と呼ばれる定数，N は方向データ $\boldsymbol{u}$ の形式的な次元である．$\boldsymbol{\mu}$ は方向ベクトルの平均方向で，われわれの文脈では $\boldsymbol{r}(t)$ と等値される.

(注6)：詳細は, [5] を参照されたい.

この $\boldsymbol{u}$ の分布から，異常度 $z(t)$ についての分布を導くために，$\boldsymbol{r}$ を極軸とする球座標系を導入し，特に，角度変数 $\theta \in [0, \pi]$ を $\cos\theta = \boldsymbol{r}(t)^T \boldsymbol{u}$ で定義する．残りの角度変数を $\theta_2, \theta_3, ..., \theta_N$ などと書くことにすれば，θ についての周辺分布は,

$$p(\theta) = \int p(\boldsymbol{u}|\kappa, \boldsymbol{\mu}) J(\theta, \theta_2, ..., \theta_N) d\theta_2 \cdots d\theta_N \tag{15}$$

と表せる．ただし，J は変換のヤコビアンであり，実験的に $|\theta| \ll 1$ が成り立つと仮定すると，$\sin\theta \simeq \theta$ が成り立つことより，J は θ^{N-2} と θ に依存しない部分との積の形となる．したがって，θ に依存する部分は

$$p(\theta) \propto e^{\kappa \cos\theta} \theta^{N-2}$$

のように書ける．これを $z(t) \simeq \frac{\theta^2}{2}$，$\cos\theta \simeq 1 - \frac{\theta^2}{2}$ を介して z に変数変換すると，結局，z についての確率分布が,

$$q(z|N, \Sigma) \propto \exp\left[-\frac{z}{2\Sigma}\right] z^{\frac{N-1}{2}-1} \tag{16}$$

のように求まる．ここで，$\theta d\theta = dz$ という関係を用い，また，$1/(2\kappa)$ を Σ とおいた．この関数は，本質的には，自由度 $N-1$ の χ^2 分布と同じである.

このモデルは，角分散 Σ に加えて，N というパラメータを含む．我々のモデル化の次の重要なステップは，N をフィッティングパラメータとしての「有効次元」n で置き換える，というものである．χ^2 分布を含むガンマ分布族では，パラメータ推定にいわゆるモーメント法が有用であることがよく知られている．すなわち，z の 1 次と 2 次のモーメントは，パラメータ n および Σ と

$$\langle z \rangle = (n-1)\Sigma, \quad \langle z^2 \rangle = (n^2 - 1)\Sigma^2 \tag{17}$$

のように結ばれている．ただし $a = 1, 2$ に対して $\langle z^a \rangle \equiv \int z^a q(z|n, \Sigma) dz$ である．これらはパラメータ n と Σ について逆に解ける：

$$n - 1 = \frac{2\langle z \rangle^2}{\langle z^2 \rangle - \langle z \rangle^2}, \quad \Sigma = \frac{\langle z^2 \rangle - \langle z \rangle^2}{2\langle z \rangle}. \tag{18}$$

右辺のモーメントについては恒等式

$$\frac{1}{t}\sum_{i=1}^{t} z(t) = \left(1 - \frac{1}{t}\right) \frac{1}{t-1} \sum_{i=1}^{t-1} z(i) + \frac{1}{t} z(t)$$

において，$1/t$ を忘却率 β と読み替えることにより,

$$\langle z \rangle^{(t)} = (1-\beta)\langle z \rangle^{(t-1)} + \beta z(t) \tag{19}$$

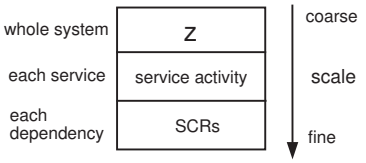

図 5　階層的な異常検出
Fig. 5　Hierarchical view of problem detection

$$\langle z^2 \rangle^{(t)} = (1-\beta)\langle z^2 \rangle^{(t-1)} + \beta z(t)^2 \qquad (20)$$

のように容易にオンライン化できる．当然 β は $0 < \beta < 1$ の範囲で選ぶ．これはデータ点の数の逆数に対応するので，興味ある時間スケールと行列生成間隔を元に，値を選択することができる．

われわれの第 3 の異常判定基準は次のようになる．

基準 3: システムは，

$$z(t) > z_{\rm th}(t)$$

が満たされた時異常である．ただし $z_{\rm th}(t)$ は

$$\int_{z_{\rm th}}^{\infty} dz\, q(z|n,\Sigma) = p_c.$$

の解である．パラメータ n と Σ は各時刻 t で計算された値を使う．

5.4　階層的な異常検出の枠組み

以上の手法に基づき，我々は図 5 に示すような階層的な障害検出の枠組みを提案する．この枠組みでは，システム全体，サービスレベル，依存関係レベルでの 3 段階の抽象度でシステムを捉え，次のような手続きで，それぞれのレベルでの異常検出を行なう．まず，最上段ではシステム全体を大まかに捉えるような異常度を計算する．ここで異常が検出される (基準 3) と，一段下がって，それぞれのサービスについての異常度を検証する (基準 2)．問題のあるサービスが特定されると，更に一段下がって，そのサービスに関連する依存関係を調べ問題箇所を特定する (基準 1)．このように，まずは大きな視点から問題の発生を検出し，次第に詳細を掘り下げていくという自然な問題解析を自動化することができる．また，この枠組みは，複数の依存関係の絡み合いを纏めあげ，適切なレベルでの警告をあげる効果があることに加え，多数の依存関係を個別に監視している際にするだけでは，頻繁に上がってきてしまう偽の警告を抑える効果も期待できる．

6. 実　　験

この章では，提案した異常検出手法の検証を，ベンチマークシステムを用いて検証を行う．実験は大きく分けて 2 つの部分から構成されている．まず，直接の依存関係を発見する手法の精度を検証し，次に，発見された依存関係に基づく異常検出フレームワークの有効性を検証する．

6.1　実 験 環 境

実験環境として，我々は図 6 に示すような 3 層の Web ベースのシステムを，IBM HTTP Server 1.3.26, IBM WebSphere Application Server 5.0.2，及び IBM DB2 Universal Database Enterprise Server Edition Version 8.1 を用い構築し，この上で Trade3 [14] と呼ばれる，オンラインの株売買サイトをシミュレートする end-to-end のシステムパフォーマンス・ベンチマークを稼動させた．

システムは概ね以下のように動作する．まず，HTTP サーバーはユーザーからのリクエストを受け付けると，これをアプリケーションサーバーに転送する．アプリケーションサーバーの上では，サーブレットと JSP が動いており，これらがリクエストを処理し，必要に応じて EJB のメソッドを呼び出す．そして，呼ばれた EJB は，JDBC コールを行うことでデータベースとのやり取りを行う．アプリケーションは，‘login’，‘home’，‘account’，‘update profile’，‘quote’，‘portfolio’，‘buy’，‘sell’，‘register’ および ‘logout’ の 10 種類の値のうちのどれかの値をパラメータとしてもつ．それぞれの値はユーザーのとることのできる行動に対応しており，例えば，HTTP サーバーに対してパラメータのどれかの値をもつ HTTP リクエストを送ることによってシステムへのログインや，株の購入，価格や自分のポートフォリオの照会などを行うことができる．サービスは，例えば (ip_1, 80, HTTP, /trade/app?action=login) のように (“IP アドレス”, “TCP ポート番号”, “プロトコル”, “パラメータ”) によって定義した．我々のアプリケーションでは，全部で 23 のサービスが観測された．図 6 における矢印はこれらのサービスを表している．

全ての実験において，対象の Web アプリケーションに対して同時に複数のユーザーのアクセスをシミュレートする仮想的なワークロード生成器を用いて，HTTP サーバーへのリクエストを生成した．同時にアクセスする仮想的なユーザーの人数を変化させることで，ワークロードを変化させた．また，トランザク

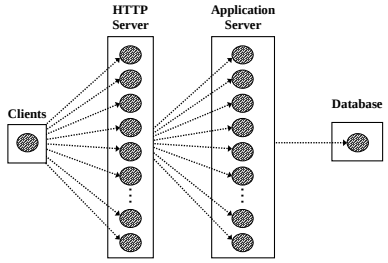

図 6　対象となる 3 層構造 Web システムと，定義されるサービス
Fig. 6　Target 3-tier system and services defined in Trade3 application

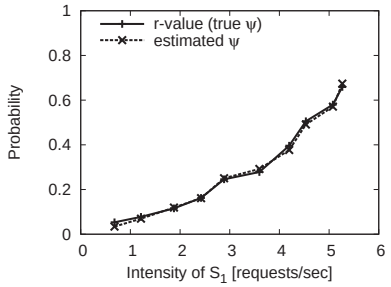

図 7　偶然の包含確率の推定
Fig. 7　Estimated probabilities of accidental containments

ション情報は 3. 章で述べた仕組みを用いて，ネットワークを流れるデータから復元した．

6.2　依存関係の発見

まずは，異常検出において用いられる直接の依存関係発見手法の精度を検証する．

まず，お互いに依存関係のないサービスの組に対し，(2) によって定義される偶然による包含確率 ψ と，(1) によって定義される，データから計算された包含確率 r を比較する．お互いに依存関係のないサービスの組に関しては，r は**真の**偶然による包含確率に一致することに注意する．図 7 は，お互いに依存関係がないことがわかっている S_1=(ip_1, 80, HTTP, /trade/app?action=portfolio) および S_2=(ip_2, 9081, HTTP, /trade/app?action=home) の組に対する比較結果を示す．ワークロードが上昇するに従って偶然による包含確率 (r-value (true ψ) で示された点)，すなわち偽の依存関係が大きくなっていくことがわかる．また，我々の推定 (estimated ψ で示された点) はワークロードが上昇しても，よい精度であることが確認できる．

次に，推定された偶然による包含確率を，競合モデルに用いた場合のサービス呼び出し割合の推定値の精度を検証する．図 8 において，‘indep’ とラベル付けされたグラフは S_1 と S_2 の間のサービス呼び出し割合の推定値を，‘dep’ とラベル付けされたグラフは S_1'=(ip_1, 80, HTTP, /trade/app?action=home) と S_2 との間のサービス呼び出し割合の推定値を示している．‘Gupta’ とラベル付けされたグラフは Gupta ら [1] による r 値から $r \cdot \sharp(S_2)/\sharp(S_1)$ によって計算されるサービス呼び出し割合の値を，‘prop1’ とラベル付けされたグラフは提案手法によって推定された包含確率 ψ を用いて $r' = (r-\psi)/(1-\psi)$ によって補正された r 値に基づくサービス呼び出し割合の値を示している(注7)．また，‘prop2’ とラベル付けされたグラフは直接の依存関係の制約 (3) を取り入れた競合モデルによるサービス呼び出し割合の値 (9) を示している．$S_1' \Rightarrow S_2$ の真の値は 1 であり，いずれの手法も非常によい精度であることが確認できる．しかしながら，依存関係のない S_1 と S_2 の場合，真のサービス呼び出し割合は 0 であるが，Gupta ら [1] の手法ではワークロードが高くなるにつれ偶然の包含による悪影響が見られる．一方，推定された包含確率 ψ を用いて補正を行うことによって，サービス呼び出し割合の値が大幅に改善されていることが分かるが，さらに競合モデルを導入することによって，さらに改善されることが確認できる．

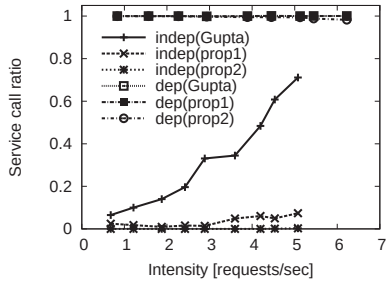

図 8　2 つのサービスの間のサービス呼び出し割合
Fig. 8　Service call ratios between two services

なお，ここでは，2 つの依存関係の例のみを示したが，他のサービス同士の組に対しても，同様の傾向が確認された．

6.3　異常検出と異常個所の特定

次に，提案手法の異常検出能力の検証を行う．そのために，我々は Trade3 のアプリケーションに，ある時点でサーブレットに Runtime Exception が発生するという人工的な問題を仕掛けた．サーブレットは

(注7)：真の包含確率を r' としたときの関係 $r = r' + \psi - r'\psi$ より．

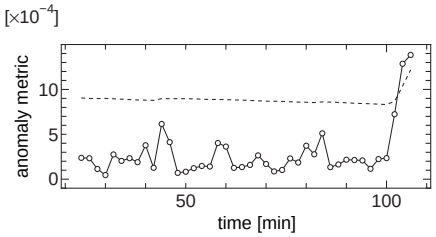

図 9　システム全体での異常度と閾値
Fig. 9　Time dependence of the anomaly metric and its threshold.

Exception を受け取ると，壊れた HTTP ページをユーザーに対して返す．このタイプの問題では，各サービスはエラーを返さずログには通常モードでもあまりに多くの Runtime Exception が出るため，リターンコードやログを監視する方法では検出するのが困難である．従って，システム管理者はしばしばユーザーからのクレームを受けてはじめて障害の発生に気づくという事態がしばしば起こる．障害発生によって，サーブレットは Exception を処理するために内部での実行パスを変更し，これは続いて呼び出されるサービスの頻度の変化を引き起こすため，提案手法は，この手の障害をうまく検知できることが期待できる．

我々は，図 5 に示されるように，まずはシステム全体を巨視的な特徴量で見ることによって異常の発生を検出し，次第に細かく見ていくことで，問題の箇所を特定するというシナリオに従い，リアルタイムで障害を検出する実験を行った．

システムに対しては毎秒平均 8.28 の HTTP サーバーへのリクエストを発生させ，開始から $t = 104$ 分後に障害を発生させた．また，サービス呼び出し割合行列 C とサービス頻度行列 F は，2 分毎に，2 分間集計した値から計算し，出力させた．また，その中で用いられる偶然の包含確率の推定値 ψ (2) は，最近の 2 分間のデータから計算した．

まず，我々はシステム全体としての異常度 z の値と，その閾値をリアルタイムで計算した．パラメータとしては，$W = 10$，$\beta = 0.01$，$p_c = 0.01$ のように定めた．図 9 にその結果を示す．最後の 3 時点で，z の値が $z_{\rm th}$ の値を超えているのが分かり，ここで何らかの異常が発生したことを示している．また，閾値をリアルタイムで計算しているので，z の値が一旦高くなったあとは，$z_{\rm th}$ の値も上昇していることが分かる．

次に，異常の発生しているサービスを特定するため，**基準 *2*** に従い，サービス活動度を調べた．図 10 は，時刻 $t = 104$ 分における

$$\gamma_i \equiv \frac{|u_i(t) - \mu_i|}{\sigma_i(t) x_{\rm th}}$$

の値を示している．上限の閾値か下限の閾値をオーバーすると，$\gamma_i > 1$ となり，そうでないときには $0 < \gamma_i < 1$ となることに注意する．19 番目のサービスにおいてのみ，γ の値が非常に高い値を示しており，このサービスにおいて何らかの異常が起こっていることが示唆される．また，図 11 は，このサービスの活動度を時間変化で見たものである．図の中には点線で上下限の閾値を示しているが，$t > 102$ において，下限の閾値を下回っていることが確認できる．また，トラフィックのバースト的な性質にも関わらず，正常時における活動度は比較的安定していることもわかる．この結果からも，個別のガウス分布による仮定の有効性がわかる．19 番目のサービスは (ip_2, 9081, HTTP, /trade/app?action=home) で指定されるサービスであり，従って，このサービスへの呼び出しか，あるいは，このサービスによる呼び出しに関連する箇所に異常があるのではないかということが推測される．

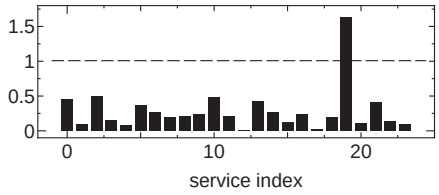

図 10　時刻 $t = 104$ 分における γ の値
Fig. 10　The γ value at $t = 104$

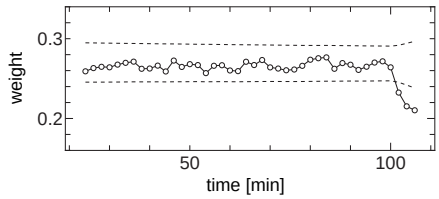

図 11　19 番目のサービス活動度の時間変化と異常判定の閾値
Fig. 11　The activity of service 19 and its threshold for anomaly detection

最後に，サービス呼び出し割合の行列において，サービス呼び出し割合の行列においてサービス (ip_2, 9081, HTTP, /trade/app?action=home) が呼び出している列（第 19 列）と，呼び出されている行（第 19 行）を調べることで，さらに詳細に問題のある箇所を特定する．図 12 にはそのうち二つの要素の時間変化を示

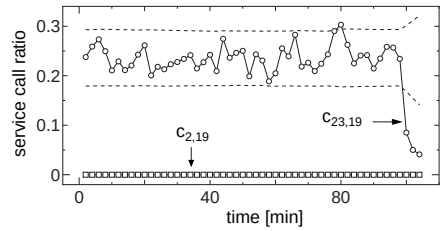

図 12　19 番目のサービスに関連するサービス呼び出し割合の時間変化と異常判定の閾値
Fig. 12　The service call ratios related to service 19 and their threshold for anomaly detection

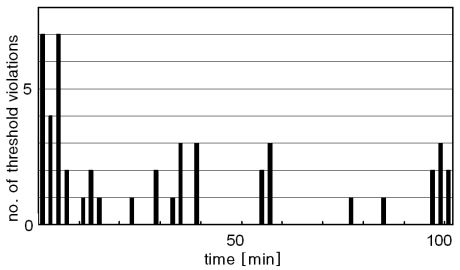

図 13　サービス呼び出し割合のみに基づく異常報告数
Fig. 13　The number of alerts only based on service call ratios

しているが，$c_{19,23}$ 要素において，$t > 102$ で閾値を下回っていることがわかる．これは，サービス (ip_2, 9081, HTTP, /trade/app?action=home) の呼び出しているサービス (ip_3, 50000, DRDA, TRADE3DB) に対応しており，すなわちこの呼び出しの部分に何か問題があることが示唆される．

この章の初めに述べたように，サービス呼び出し頻度はサービス間の呼び出し回数そのものであり，ワークロードとともに増減するため，こういった推論を行うことは難しい．また，**直接の**依存関係でなく，**間接的な**依存関係を用いると，一箇所の障害が，障害箇所を経由する間接的な依存関係すべてに影響するため，障害箇所の特定が困難になってしまう．

また，図 13 は，階層的な異常検出によらず，5.1 節のサービス呼び出し割合の行列の要素のみに基づいて異常検出を行った場合に異常が報告される回数を各時間においてカウントしたものである．各要素のみに注目するような検出方法では正常時にも偽の異常が検知されていることがわかる．階層的な異常検出法はこのような偽情報を排除するという利点があることが示されている．一方，実際に異常が起こった $t = 104$ 付近では，16 番や 21 番のサービスに関連するサービス呼び出し割合にも異常が検出されていた．このように，複数の箇所において異常が検出されるような場合にも階層的な異常検出法によってさらに候補を絞りこむことができることも示唆される．

7. 関 連 研 究

依存関係情報は静的な依存関係と動的な依存関係の 2 つのカテゴリーに分類することができる．静的な依存関係システムの設定や，インストール時のデータ，あるいは，コードなどを解析することで発見される．Kar ら [15] は，ソフトウェア・コンポーネント間の静的な依存関係を，レポジトリから自動的に抽出する方法を提案している．しかしながら，多くのコンポネントは実行時に動的にバインドされ，また依存関係は環境の時間とともに変化するため，このような方法は，実行時における**動的な**依存関係を見つけるためには使うことができない．

動的な依存関係を発見するためには，システムの動作に関する情報を稼動時に集め，解析するような手法が必要となる．このような目的を達成するために，特殊な計測の仕組みを用いて行うような方法がいくつか存在する [6], [16], [17] が，これらの方法は対象のシステムに負荷をかけてしまうという欠点がある．時には，システムにあまり負荷をかけずに簡単に採取できるデータから，依存関係を推定する手法 [1], [18] の方が望ましい場合もある．例えば，Ensel [18] は，CPU 負荷などの情報からニューラルネットワークを用いて，動的な依存関係を推定する手法を提案している．我々のアプローチも，まさにこの範疇に属している．

依存関係の応用については，診断や異常個所の特定などに関して，多くの研究がなされている．例えば，イベント相関システム [19]〜[21] では，アラートやイベント情報を集計し，依存関係における対応箇所にマッピングすることで，問題箇所の特定を行う．また，他にも，依存関係の情報をもとにして，プローブやテスト用のトランザクションを設計し，効率よく問題箇所を特定するような試みもある [22], [23]．Hellerstein ら [24] は，データマイニング手法を用いることで，対象システムより得られたイベント系列から，特徴的なパターンを発見する試みを行っている．また，Ma ら [25] はイベントやパターンを視覚化し，ユーザーの問題解析を助けるシステムを開発している．本論文で我々が提案した手法は，相関ルール発見 [26] や系列パターン発見 [27] の一種であるという見方もできる．本論文では 2 つのサービスの組の依存関係の発見のみに

注目して議論したが，この考え方は 3 つ以上のサービスの組にも自然に一般化できる．

近年，ネットワークのトラフィック解析などにおいて，変化点検出手法の有効性を示す研究がなされている [3], [4]．Hajji [10] は Yamanishi ら [11], [12] と同じく，確率モデルを用いた LAN における異常検出のアプローチを提案しているが，1 観測点のみでの低い層におけるデータを対象としている．これらの研究は主にネットワークの低い層における非常に集約されたデータを対象にしているのに対し，我々の手法はより抽象度の高い層，とくに Web ベースのシステムにおけるアプリケーション層を対象としている．Thottan ら [9] は，複数観測点におけるデータを AR モデルによって関係付ける興味深い方法を提案している．

8. おわりに

本論文で我々は分散システムにおける異常を自動的に発見する技術を考察した．また，この目的のために (i) 対象のシステムに殆ど負荷をかけることなく採取できるネットワークデータから，トランザクションの情報を抽出するシステムと，(ii) 得られたトランザクションの実行期間をもとに，サービス間の直接の依存関係を推定する新しい手法，および (iii) 発見された依存関係を基にした，階層的な異常検出の枠組みの 3 つを提案した．また，実験システムを用いてこれらの手法の有効性の検証を行った．

今後の研究の方向として，いくつかの可能性を挙げることができる．我々のネットワークデータを用いるアプローチは，複数のノードを同時に監視できるというメリットがあるが，更に広域にわたるシステムの場合には，やはり複数の観測データを統合する必要があり，3. 章で述べたような時刻の同期が問題となってくる．しかしながら，通信の内容をもとに，これらを紐付けるような方法が有効である可能性がある．また，プロトコル解析モジュールはプロトコル毎に準備しなければならないが，対象となるサービスが未知のプロトコルを使用している場合，その作成は非常に困難である．未知のプロトコルの解析を助けるツールや，未知のプロトコルを自動でクラスタリングし，分類する手法などが必要になっていくと思われる．また，トラフィック量が多くなりミラーポートの許容量 (例えば 1 Gb/s) を超えてしまうと，通過するすべてのデータを取得することができなくなってしまう点である．現在の手法では，パケットロス率にほぼ比例した形で本来の包含関係が失われてしまうこととなるが，パケットロスに対してよりロバストな推定手法を開発することも今後の課題といえる．また，本論文では異常検出の閾値決定に正規分布を用いたが，これは正常時のシステムが安定動作しているときには比較的正しく，1 回の異常検出，すなわち異常が検出されるまでモデルが正しければある程度は目的が達成されるような状況においてはうまく働くことが経験的にわかっているが，より複雑なシステムや現実のトラフィックにおいて，異常時，特にバースト性やサスペンドなどの現象が起こっている場合に，どこまで我々の仮定が成立するかは明らかではなく，これを見極めることは今後の課題である．

文　献

[1] M. Gupta, A. Neogi, M. K. Agarwal and G. Kar: “Discovering dynamic dependencies in enterprise environments for problem determination”, Proc. 14th IFIP/IEEE Int. Workshop on Distributed Systems: Operations and Management, Vol. 2867 of LNCS, Springer, pp. 221–232 (2003).

[2] A. P. Dempster, N. M. Laird and D. B. Rubin: “Maximum likelihood from incomplete data via EM algorithm”, Journal of the Royal Statistical Society, **B**, 39, pp. 1–38 (1977).

[3] P. Barford, J. Kline, D. Plonka and A. Ron: “A signal analysis of network traffic anomalies”, Proc. Second ACM SIGCOMM Workshop on Internet Measurment, pp. 71–82 (2002).

[4] H. Wang, D. Zhang and K. G.Shin: “Detecting SYN flooding attacks”, Proc. IEEE INFOCOM 2002, pp. 1530 –1539 (2002).

[5] T. Idé and H. Kashima: “Eigenspace-based anomaly detection in computer systems”, Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (2004).

[6] OpenGroup Technical Standard C807: “Systems management: Application response measurement” (1998).

[7] A. O. Allen: “Probability, Statistics, and Queuing Theory with Computer Science Applications”, Computer Science and Scientific Computing, Academic Press (1990).

[8] M. Sato and S. Ishii: “On-line EM algorithm for the normalized gaussian network”, Neural Computation, **12**, 2, pp. 407–432 (2000).

[9] M. Thottan and C. Ji: “Anomaly detection in IP networks”, IEEE Trans. Signal Processing, **51**, 8, pp. 2191– 2204 (2003).

[10] H. Hajji: “Baselining network traffic and online faults detection”, Proc. IEEE Int. Conf. on Communications, Vol. 1, pp. 301–308 (2003).

[11] K. Yamanishi, J. Takeuchi, G. Williams and P. Milne: “On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms”, Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 320–324 (2000).

[12] K. Yamanishi and J. Takeuchi: “A unifying framework for detecting outliers and change points from non-stationary time series data”, Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, pp. 676–681 (2002).

[13] K. Mardia, J. Kent and J. Bibby: “Multivariate Analysis”, Academic Press (1980).
[14] IBM: “Trade3; `http://www-306.ibm.com/software/webservers/appserv/benchmark3.html`”.
[15] G. Kar, A. Keller and S. Calo: “Managing application services over service provider networks: Architecture and dependency analysis”, Proc. 5th IFIP/IEEE International Symposium on Integrated Network Management (IM V) (2000).
[16] J. Aman, C. K. Eilert, D. Emmes, P. Yocom and D. Dillenberger: “Adaptive algorithms for managing a distributed data processing workload”, IBM Syst. J., **36**, 2, pp. 242–283 (1997).
[17] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox and E. Brewer: “Pinpoint: Problem determination in large, dynamic internet services”, Proc. Int. Conf. on Dependable Systems and Networks (2002).
[18] C. Ensel: “New approach for automated generation of service dependency models”, 2nd Latin American Network Operation and Management Symposium, LANOMS (2001).
[19] S. A. Yemini, S. Kliger, E. Mozes, Y. Yemini and D. Ohsie: “High speed and robust event correlation”, IEEE Commun. Mag., **34**, 5, pp. 82–90 (1996).
[20] J. Choi, M. Choi and S. Lee: “An alarm correlation and fault identification schema based on OSI managed object classes”, Proc. IEEE Int. Conf. on Communications, pp. 1547–1551 (1999).
[21] B. Gruschke: “Integrated event management: Event correlation using dependency graphs”, Proc. 9th IFIP/IEEE International Workshop on Distributed Systems Operation and Management (1998).
[22] J. Gao, G. Kar and P. Kermani: “Approaches to building self healing systems using dependency analysis”, Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS) (2004). to appear.
[23] M. Brodie, I. Rish and S. Ma: “Optimizing probe selection for fault localization”, Proc. 12th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM01) (2001).
[24] J. L. Hellerstein and S. Ma: “Mining event data for actionable patterns”, Proc. 26th Int. Computer Measurement Group Conference, pp. 307–318 (2000).
[25] S. Ma, C. Perng and J. L. Hellerstein: “Eventminer: An integrated mining tool for scalable analysis of event data”, Proc. Visual Data Mining Workshop, pp. 1–9 (2001).
[26] R. Agrawal, T. Imielinski and A. N. Swami: “Mining association rules between sets of items in large databases”, Proc. 1993 ACM SIGMOD International Conference on Management of Data, ACM Press, pp. 207–216 (1993).
[27] R. Agrawal and R. Srikant: “Mining sequential patterns”, Proc. 11th Int. Conf. on Data Engineering, IEEE Computer Society, pp. 3–14 (1995).

付　　録

1. 定理の証明

トランザクション履歴 D に対する対数尤度の和 L のヘッセ行列 H が，半負定であることを証明する．

H は，$j = l := m$ に対して

$$
\begin{aligned}
[H]_{(i,j),(k,l)} &= \frac{\partial^2 L}{\partial\rho(S_i,S_j)\partial\rho(S_k,S_l)} \\
&= -\sum_{T\in D(S_j)} \delta(S_i\in P(T))\delta(S_k\in P(T)) \\
&\quad\cdot\frac{\prod_{S'\in P(T)\setminus S_i}\psi(S',S_m)\prod_{S''\in P(T)\setminus S_k}\psi(S'',S_m)}{\left(\sum_{S\in P(T)}\rho(S,S_m)\prod_{S'\in P(T)\setminus S}\psi(S',S_m)\right)^2},
\end{aligned}
$$

そのほかの場合には $[H]_{(i,j),(k,l)} = 0$ と書ける．

従って，

$$
\begin{aligned}
[H_m(T)]_{j,l} &:= \delta(S_i\in P(T))\delta(S_k\in P(T)) \\
&\quad\cdot\prod_{S'\in P(T)\setminus S_i}\psi(S',S_m)\prod_{S''\in P(T)\setminus S_k}\psi(S'',S_m).
\end{aligned}
$$

で定義される $H_m(T)$ が，$\forall m = 1,2,\ldots,|\Sigma|$ と $\forall T \in D$ に対し，半正定であることを証明すればよい．

$H_m(T)$ は以下のように分解される．

$$
\begin{aligned}
H_m(T) &= \mathbf{h}_m(T)\mathbf{h}_m^\top(T), \\
\mathbf{h}_m(T) &:= (h_m^1(T),h_m^2(T),\ldots,h_m^{|\Sigma|}(T))^\top, \\
h_m^i(T) &:= \delta(S_i\in P(T))\prod_{S'\in P(T)\setminus S_i}\psi(S',S_m).
\end{aligned}
$$

従って，$\forall\mathbf{x}\in R^{|\Sigma|}$ に対し，

$$
\mathbf{x}^\top H_m(T)\mathbf{x} = \left(\mathbf{x}^\top\mathbf{h}_m(T)\right)^2 \geqq 0
$$

であることから $H_m(T)$ は半正定であることがわかる．

（平成 17 年 XX 月 XX 日受付，XX 月 XX 日再受付）

鹿島　久嗣

1999 年京都大学工学研究科応用システム科学専攻修士課程修了．同年日本アイ・ビー・エム（株）入社．東京基礎研究所に所属．機械学習，データマイニングの研究に従事．

津村　直史

2003 年大阪大学大学院工学研究科電気電子情報エネルギー工学専攻修士課程修了．同年日本アイ・ビー・エム（株）入社．東京基礎研究所に所属．オートノミックコンピューティングの研究に従事．

井手　剛　（正員）

2000 年東京大学大学院理学系研究科物理学専攻修了．博士（理学）．同年日本アイ・ビー・エム（株）入社．現在，東京基礎研究所にて実数値データに対するマイニング技術の研究に従事．2004 年人工知能学会全国大会優秀賞．電子情報通信学会，情報処理学会，人工知能学会，日本物理学会，ACM SIGKDD 各会員．

野ヶ山尊秀　（正員）

2004 年電気通信大学工学研究科電子情報学専攻博士前期課程了．同年日本アイ・ビー・エム（株）入社．東京基礎研究所に所属．オートノミックコンピューティングの研究に従事．

平出　涼

2002 年東京工業大学総合理工学研究科物理情報システム創造専攻修士課程修了．同年日本アイ・ビー・エム（株）入社．東京基礎研究所に所属．オートノミックコンピューティングの研究に従事．

江藤　博明

1985 年東京工業大学工学研究科情報工学専攻修士課程修了．同年日本アイ・ビー・エム（株）入社．東京基礎研究所に所属．最適化，セキュリティ，オートノミックコンピューティングなどの研究に従事．

福田　剛志

1991 年早稲田大学大学院理工学研究科修士課程修了．同年日本アイ・ビー・エム (株) 東京基礎研究所入所．データマイニング，オートノミックコンピューティングなどの研究に従事．現在，大和ソフトウェア開発研究所情報マネジメント製品開発担当．博士（情報科学）．日本データベース学会，情報処理学会，ACM, IEEE CS 各会員．

Abstract We introduce a network-based problem detection framework for distributed systems, which includes a data-mining method for discovering dynamic dependencies among distributed services from transaction data collected from network, and a novel problem detection method based on the discovered dependencies. From observed containments of transaction execution time periods, we estimate the probabilities of accidental and non-accidental containments, and build a competitive model for discovering direct dependencies by using a model estimation method based on an on-line EM algorithm. Utilizing the discovered dependency information, we also propose a hierarchical problem detection framework, where microscopic dependency information is incorporated with a macroscopic anomaly metric that monitors the behavior of the system as a whole. This feature is made possible by employing a network-based design which provides overall information of the system without any impact on the performance.

Key words problem detection, network data, distributed system, EM algorithm