# ログを利用した性能異常シグネチャの作成手法

高橋宏明[†] 岩田聡[†]
山田浩史[†,††] 河野健二[†,††]

## 1. はじめに

近年，インターネットシステムにおけるサーバの性能異常が問題になっている．サーバの性能異常は様々な悪影響を及ぼすため早急に対処する必要がある．

サーバの性能異常の解決法を発見する 1 つの方法として “過去に発生した性能異常と比較する” という方法がある．これは，サーバでは同じ性能異常が繰り返し発生することがあるという前提に基づいている．例えば，クライアント数が増加することによって発生する性能異常があるとする．この性能異常に対してサーバの管理者が設定を変えるという対策をしたとしても，これをさらに超えるクライアントからの接続があると再度同じ性能異常が発生してしまう．また，サーバアップデート時に設定が初期化された場合も同じ性能異常が発生する可能性がある．

現在発生している性能異常が以前に発生したものだと分かれば，既に分かっている対処法を適用することで性能異常が拡大するのを防ぐことができる．また，解決法が分からなかった場合でも性能異常の種類を特定することができる．例えば，毎日同じ時間に同じ性能異常が発生していることが分かれば，解決法発見のために役立つ情報となる．

そこで本研究ではログを用いた性能異常シグネチャの作成手法を提案する．性能異常シグネチャとは性能異常を一意に定める指標である．これにより性能異常発生時にその解決法を早期に発見できるようになると考えている．

## 2. 提案手法

本研究ではサーバのログを用いた性能異常シグネチャの作成手法を提案する．異常発生時の情報をシグネチャとして保存することで，異常発生と同時に過去に観測された性能異常と比較できる．指標として用いるサーバのログはシステムの詳細なイベントを出力するため性能異常の解決法を発見するのに適しているといえる．また，ログの種類が多ければ発生している性能異常の識別だけでなく，性能異常の予兆を発見したりできるようになると考えている．

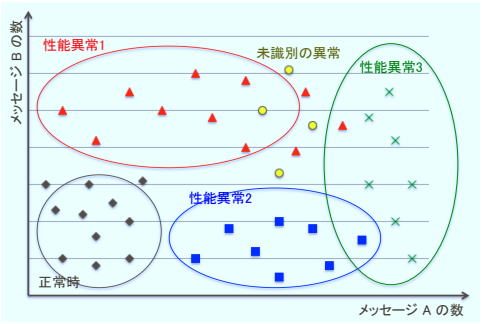


**図 1** シグネチャの例

本研究ではこれを 2 つのステップに分けて提案手法を実現する．ログ出現パターンの表現手法とログ出現パターンのマッチング手法である．

### 2.1 ログの出現パターンの表現手法

本研究ではログの出現パターンをシグネチャとして表現する．具体的には，ログメッセージの種類ごとに一定時間間隔で現れる数をシグネチャにする．ログメッセージはサーバが発生イベントの詳細やステータスを出力するものであり，$n$ 種類のログメッセージがあればシグネチャは $n$ 次元のベクトルとなる．出現回数をシグネチャに用いる理由は正常時にはほぼ一定で，異常時に急激に変化することが多いからである．

図 1 は 2 次元のシグネチャの例である．2 次元の場合，2 種類のメッセージ (A,B) の数を用いて性能異常を識別する．ただし，性能異常ごとにメッセージの数は毎回一定であるという訳ではないため，図のように複数の点の集まりを用いて性能異常を表現する．

### 2.2 ログの出現パターンのマッチング手法

出現パターンのマッチング手法ではシグネチャ同士のマッチングを行う．ここでは，過去に発生した性能異常を人がラベル付けした結果を用いて学習を行い，未知のシグネチャが入ってきたときに性能異常で分類することが目的である．

本研究ではニューラルネットワークを誤差逆伝播法で訓練することでこれを実現する．ニューラルネットワークとは与えられた入力から出力を計算するモデルであり，本研究では入力はシグネチャ，出力は発生し

† 慶應義塾大学
†† 科学技術振興機構 CREST

ている性能異常とする．ただし，ニューラルネットワークを作成するだけでは正しい出力とはならないため，これを目標出力に近づけるために誤差逆伝播法を用いる．誤差逆伝播法により，学習データ (過去に発生した性能異常とそのときのシグネチャ) を用いてニューラルネットワークの内部パラメータを適切な値にする．

## 3. 実　　験

提案手法が有用であることを確認するために，Apache HTTP サーバを用いて性能異常シグネチャを作成し，その精度を測定する実験を行った．本実験では RUBiS[1] というオークションサイトを模したベンチマークアプリケーションを用いた．

### 3.1 方　　法

本実験では RUBiS 稼働中のクライアント数を 500 から 4000 まで変化させながら 5 分間に現れるログの数を記録していく．このログを用いて 4 種類の性能異常 (1 種類は正常時) を表すシグネチャを作成し，その精度を測定する．これは KeepAlive,MinSpareServer という 2 種類の Web サーバの設定を組み合わせた状態を表したものである．本実験では KeepAlive が off のとき，または MinSpareServer の数が少ないときに応答時間が増加した．本実験ではこれらの設定のとき性能異常が発生したこととする．本実験では KeepAlive の on，off と MinSpareServer が 1，300 の設定を組み合わせた 4 種類のシグネチャを作成する．

なお，本実験ではログが 1 種類しか出力されなかった．出力された `proxy_util.c` というログは Apache がプロキシサーバに対してコネクションを確立する時に出力されるログである．1 種類では性能異常を正確に表現できないため，本実験ではクライアント数もシグネチャに入れた (クライアント数はアクセスログからおおよその値が計算できる)．したがって，本実験で作成したシグネチャは (クライアント数，`proxy_util.c` の数) の 2 次元である．

### 3.2 結　　果

学習，テストを行った結果，性能異常を正しく分類できたのは全体の 78.1%であった．それぞれのシグネチャの精度は表 1 のようになった．また，表 2 は各クライアント数ごとの正誤を表し，図 2 はシグネチャをグラフ化したものである．表 2 では，○が正しく識別できたことを表し，×が正しく識別できなかったことを表し，さらに括弧の中がどのシグネチャと判断したかを表す．MinSpareServer が 300 でクライアント数が 500〜2500 のとき，KeepAlive が on と off でログの出力回数が 0 回で識別できなかった．その他の場合はほぼ正確に識別することができた．

## 4. 関 連 研 究

資源利用率を利用した性能異常シグネチャ作成手法

**表 1**　各シグネチャの精度

| | keepalive = on | keepalive = off |
|---|---|---|
| MinSpareServer = 300 | 50.0% | 75.0% |
| MinSpareServer = 1 | 100.0% | 87.5% |

**表 2**　各シグネチャの正誤

| Client | 500 | 1000 | 1500 | 2000 | 2500 | 3000 | 3500 | 4000 |
|---|---|---|---|---|---|---|---|---|
| A | × (C) | × (C) | × (C) | × (C) | × (C) | ○ (A) | ○ (A) | ○ (A) |
| B | ○ (B) | ○ (B) | ○ (B) | ○ (B) | ○ (B) | ○ (B) | ○ (B) | ○ (B) |
| C | ○ (C) | ○ (C) | ○ (C) | ○ (C) | ○ (C) | × (B) | ○ (C) | ○ (C) |
| D | ○ (D) | ○ (D) | ○ (D) | ○ (D) | ○ (D) | ○ (D) | × (C) | ○ (D) |

A : KeepAlive=on,MinSpareServer=300, B : KeepAlive=on,MinSpareServer=1
C : KeepAlive=off,MinSpareServer=300, D : KeepAlive=off,MinSpareServer=1

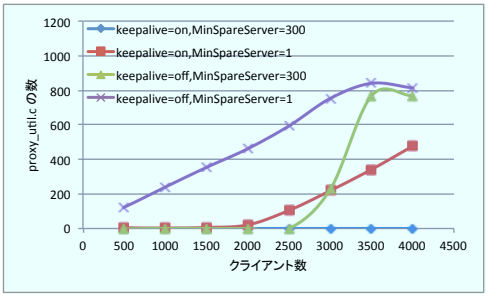


**図 2**　4 種類の性能異常シグネチャ

がある[2]．この研究では CPU 利用率や応答時間を用いてシグネチャを作成し，適切なパラメータを選択するとほぼ正確に性能異常を識別できることを示した．

この研究は本研究と目的は同じであるが用いる指標が異なる．本研究ではログを用いて性能異常の識別を行う．ログの方がファイル名やプロセスの詳細などより多くの情報が得られるため，性能異常識別の精度が高くなると考えている．

## 5. ま と め

本研究ではログを用いた性能異常シグネチャの提案をした．性能異常シグネチャはログの出現回数を用いて表現し，ニューラルネットワークを作成し，誤差逆伝播法を用いて学習を行った．そして，RUBiS を用いて実験を行ったところ 78.1%の精度を記録した．

## 参 考 文 献

1) RUBiS: Rice University Bidding System. Rice University. `http://rubis.ow2.org/`.
2) Bodik et al. Fingerprinting the Datacenter: Automated Classification of Performance Crises. In *Proc. European Conference on Computer Systems*, 2010.