Vol.2010-SLDM-144 No.62
Vol.2010-EMB-16 No.62
Vol.2010-MBL-53 No.62
Vol.2010-UBI-25 No.62
2010/3/28

組込みシステム向けオンライン障害検出システムの提案

菅谷　みどり††　中　島　達　夫†

組込み機器の高性能化によりソフトウェアの複雑化，ネットワーク接続などにより運用時に不具合が発生する事が問題となっている．本研究の目的は，こうした不具合が発生した場合機械学習を用いたオンライン監視により検出する手法を提案することである．目的達成のために，オペレーティングシステム内部で取得できるプロセスごとの資源利用率の情報を取得する仕組みを実装した．本論文では，オンライン監視基盤から取得できる情報をもとに機械学習を行い，評価を行った．その結果，異常モデルの特定しない場合でも，オーバーヘッドを約 1%におさえつつ現実的な精度で異常を検出できた事を示す．

Suggest of Online Proactive Failure Prediction System for Embedded Systems

Midori Sugaya †† and Tatsuo Nakajima†

In this paper, an failure prediction infrastructure by Resource Monitoring is presented for embedded systems.It provides a monitoring function for detecting anomalies, especially a symptom of resource abuse or memory leaks, by using the resource patterns of each process. Our system takes a application black-box approach, based on machine learning methods. It uses the clustering method to quantize the resource usage vector data and then learn the normal patterns with a hidden Markov Model. This reduces the general overhead of the analyzer and makes it possible to monitor the process in real-time. The evaluation experiment indicates that our prototype system is able to detect anomalies such as CPU abuse and memory leaks, without previously defined anomaly models.

1. はじめに

近年, 組込み機器の高性能化，高機能化により，サービスの多様化が顕著となっている．サービスの多様化を実現するためにソフトウェアも巨大化，複雑化の傾向にある．こうした中，ソフトウェアの不具合の増加が経済産業省より報告されている[1)]．問題に対処するために，ソフトウェア開発における検証手法などが様々提案されているが，組込み機器の発期間は現状短く，再現性の低い障害に対し事前に完全に検証を行うことは限界がある．

こうした運用時に生じる再現性の低いバグに対し，何らかの兆候を運用中に検知し，障害を未然に防ぐ方法は汎用機などでは高信頼化技術として長期にわたり研究されているが，組込みシステムへの適用は今まで十分に検討されてこなかった．本研究では，この点に着目し，運用時にオンラインで障害検知を行うための手法を検討するものとした．

既存のオンライン監視システムの多くは, 大型コンピュータに適用されるもので, プロセスごとに出力される CPU の利用率やメモリの利用率情報を管理者に表示する[2)], またはあるスレッシュホールドを越えた場合に警告を出す[3)] など，閾値の設定やコンフィグレーションを事前に行う必要があった．しかし，組込み機器ではネットワークに接続する機器の爆発的増加や, そもそもの個数が多いことから，汎用機のようにシステムごとにコンフィグレーションを行うことは限界がある．

また，オンライン監視の検知のために利用する手法や取得するイベントは，アプリケーション固有のエラーなどのイベントを利用することが異常検知にとっては有効である．しかし，アプリケーション固有のエラー処理が書かれていないケースなどはエラーの検知ができない問題がある．また，こうしたエラー記述は，

† 早稲田大学理工学術院
Waseda University, Department of Computer Science
†† 独立行政法人科学技術振興機構 ディペンダブル組込み OS 研究開発センター
Japan Science and Technology Agency (JST), Dependable Embedded OS RD Center

Vol.2010-SLDM-144 No.62
Vol.2010-EMB-16 No.62
Vol.2010-MBL-53 No.62
Vol.2010-UBI-25 No.62
2010/3/28

多くはアプリケーションに固有である汎用性が低い問題がある．

本研究では，こうした従来のオンライン監視システムの課題に対して，組込みシステム向けに機械学習を用いたオンライン監視による事前障害検知手法およびシステムを提案する．本システムは，機械学習により障害の予兆を検出する事を目的とした．本システムでは，機械学習を用いることでコンフィグレーションを自律的に行う．また，プロセスごとの資源情報を利用することで，汎用性と精度の高い検知を実現する．資源情報をオペレーティングシステム内で取得することでオーバーヘッドを低下させる方法を示した．本システムにより，従来精度と汎用性はトレードオフであったが，これを解消することができると考える．

本稿では，これらの内容について 2 節にて，本システムの提案について述べる．3 節にて，異常検知の手法について述べ，4 節にて評価について述べる．5 節にて関連研究について述べる．最後に 6 節にて現状の課題と結論について述べる．

2. 組込みシステム向けオンライン監視システムの提案

2.1 組込みシステム向けオンライン監視への要求

障害に対処するためには，障害検知をオンラインで行う必要があり，その期待は高い．しかし，従来実用性を考えた場合適切な手法が十分ではない．従来手法の問題点を以下の 4 点にまとめた．

- 機器ごとのコンフィグレーションが必要:
  汎用機で利用される障害検知システムは一般的に，監視マシンごとに管理者が警告を出すための設定を手動で行う必要がある[3)2)]．しかし，今後ネットワークに接続する組込み機器は，攻撃や障害のリスクは汎用機器と同等ながらも，その数の多さや単体にかけられるコストからみて，ひとつひとつを管理者がコンフィグレーションを作成することは困難である．
- 特定の言語で書かれたアプリケーションのみを対象とする:
  汎用的に利用されるためには，監視システムが障害のモデル化に利用する入力データが，特定のアプリケーションに限定されないことが重要である．
- 検知精度が低い:
  実用化されている監視システムにおいては，Nagios の例のように，検知できる障害の範囲は広いが比較的シンプルなエラー評価に基づいて行われているため，検知精度を上げることが難しい問題がある[3)]．
- オーバーヘッドが高い:
  オンライン監視においては，本来のサービスを阻害するオーバーヘッドは敬遠される．システムコールなどの入力データを利用する方法は IDS(Intrusion Detection System) の分野で成果が上がっている[4)]．しかし，IDS 手法は論理的な因果関係までを解析することから，取得する情報量が多く，その結果オーバーヘッドが高い．一般に検知精度が高い手法はオーバーヘッドが大きい傾向にあり，現場で利用しづらい問題がある．

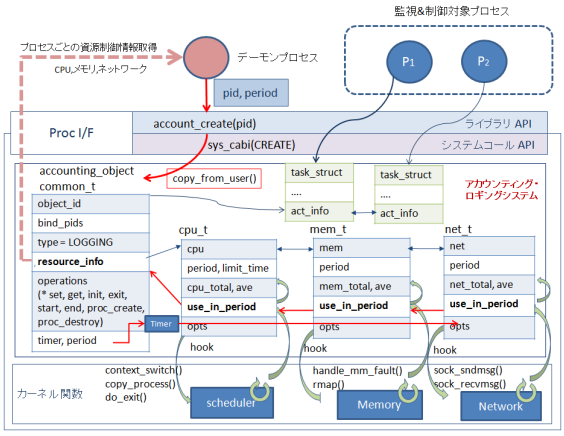

図 1 監視システムのデータ構造

2.2 組込み向けオンライン監視システムの提案

2.2.1 組込み向けオンライン監視システムの提案

組込みシステム向けオンライン監視機能への要求を踏まえ，本研究では我々は，オンライン監視において取得のオーバーヘッドが低い点から，プロセスが利用する CPU，メモリなどの資源情報に着目した．これらの資源の統計情報は言語に依存せずに，どのようなアプリケーションも監視対象にできる上，プラットフォームアーキテクチャに依存せずに取得できるため，オンライン監視への要求を満たす．

しかし，一般にプロセスごとに表示される資源利用率情報は精度が低く，その動作を逐次確認する用途には適さない．実際は Linux 内部ではナノ秒単位で情報を保持しているが，ユーザインターフェイスへの表示の際に μ 秒に丸めを行っているため，情報の精度が低い．また，CPU，メモリ，ネットワークなどの情報は別々に管理されているため，同じ周期内で発生している現象を正確に捕える事は困難である．

これに対して，我々は OS 内部で保持しているプロセスの統計情報を，同じ周期で直接取得することで，プロセスの実行状態の変化を観察できると考えた．統計

Vol.2010-SLDM-144 No.62
Vol.2010-EMB-16 No.62
Vol.2010-MBL-53 No.62
Vol.2010-UBI-25 No.62
2010/3/28

情報の収集は一連のカーネル実行コンテキストの一部として行うため, 取得コストが少ない利点がある. さらに, サンプリング周期を短することで, プロセスの実行時の動作を正確に把握できると考えた.

また,OS 内での情報処理は, ユーザアプリケーション側で監視処理を行う場合に問題となる, 監視プロセスの優先度やデータの取得タイミングに左右されない利点がある. 我々は, このように OS 内でプロセスごとの資源利用情報を高精度で取得する手法を提案する. また, 高精度で取得した情報をもとに機械学習を行い, 特別なコンフィグレーションなしに障害検知を行えるものとした.

- 自律性:
  機械学習手法を用いることでマシンごとのコンフィグレーションを不要とする. ドメインごとの正常状態を学習し, それとの状態が異なることで異常を判定する. また, 異常に至る状態を学習させることで異常の兆候を検知することができる.
- 汎用性:
  資源の統計情報は言語に依存せずに, どのようなアプリケーションも監視対象にできる上, プラットフォームアーキテクチャに依存せずに取得できるため, オンライン監視への要求を満たす.
- 低オーバーヘッド:
  プロセスが利用する CPU, メモリなどの資源情報を OS 内部で取得することで, オーバーヘッドを削減する.

2.3 監視システム基盤の設計と実装

2.3.1 特　　徴

監視システムは, オペレーティングシステム内部で保持しているプロセスの統計情報を直接取得する仕組みである. 本研究を行うにあたり, 著者はカーネルの中で直接統計情報を取得することで, プロセスの実行状態の変化を詳細に観察することができると考え, 細粒度情報収集のための監視システムを実装した. 本監視システムの特徴は次の 3 点である.

- プロセスごとの資源利用情報の取得
- 細粒度な監視情報
- 動的な監視周期変更

2.3.2 プロセスごとの資源利用情報の取得

本監視システムでは, 汎用性や低オーバーヘッドという観点で資源情報を用いることにした. 資源情報としては, 今回 CPU, メモリ, ネットワークの 3 つの統計情報を利用することとした. これらの資源の統計情報は言語に依存せずに, どのようなアプリケーションも監視対象にできる上, プラットフォームアーキテクチャに依存せずに取得できるため, オンライン監視への要求を満たす.

図 1 に監視システムのデータ構造を示した. 本監視システムは, 管理オブジェクトごとに監視対象となるプロセスを登録し, この監視グループ単位で異なるカーネルタイマを利用可能としている. このことにより, 監視対象グループごとに異なる周期の設定も可能とした. また, タイマは高解像度タイマ (High Resolution Timer[5]) を利用している. x86 上の高解像度タイマーは理論的にはナノ秒単位までの周期も扱える. このため, 高精度な情報を得ることができる.

プロセス毎の資源の情報収集を正確に行うために, カーネル内に設置した資源ごとの計測ポイントを次に示す.

- CPU : 周期内で実行したプロセスの時間は, context_switch にフックをいれて計測した. 計測区間内のプロセスの実行時間は tsc (time stamp counter) の値を取得してで計測した.
- メモリ : プロセスが物理メモリを確保/解放するプリミティブ関数 (page_fault()) にフックを設定した. page_fault() はプロセスがメモリをページングして利用する関数である. 低レベルな物理ページを監視対象としているので, malloc() しても, 実際には copy_on_write されていないメモリ量は対象にならない.
- ネットワーク : プロセスが参照したソケットで利用したバッファ(送信+受信 bytes) 量を集計した. プロセスが新規にソケットを作成する際に, 監視対象のプロセスのソケットを登録し, 利用したバッファ量を周期ごとに加算した.

カーネル内のトレースポイントはロギングシステムである LTTng (LTT Next Generation)[6] のインターフェイスを拡張して利用した.

3. 機械学習による異常検知

3.1 隠れマルコフモデルによる機械学習

本システムでは, カーネルより得られた情報を入力値として, 機械学習を用いてプロセスごとに正常状態モデルを作成する. 運用時には, 予め作成してある正常状態モデルとの比較を行い, 一致が見られない場合には, 異常として警告を出すものとした.

機械学習で利用するカーネル内の資源統計情報は, CPU, メモリにネットワークの利用率を追加した 3 つの統計値を用いた. またプロセスごとの資源利用率の変化をモデル化する手法として, HMM (Hidden Markov Model) を用いた. 時系列データを元に予測を行う方

Vol.2010-SLDM-144 No.62
Vol.2010-EMB-16 No.62
Vol.2010-MBL-53 No.62
Vol.2010-UBI-25 No.62
2010/3/28

法の中でも，MM (Markov Model) は未来の予測値が直近の観測値以外の過去の観測値に対して独立という仮定を置くことで，モデルをシンプルにする事ができるメリットがある．特に HMM は，変数を離散値として扱う事ができるため, 我々が今回行うある標本周期で取得する資源情報を扱いやすい利点があった．また, HMM の状態遷移数の計算爆発が生じないように，予め類似度の高いデータを予めクラスタリングし，その値を元にプロセスごとの資源利用率の遷移状態を HMM で運用前に学習させるものとした．検知時には，正常状態モデルと入力情報を比較し，正常状態への適合率が低いケースを異常と判定する．

3.2 資源情報の取得

異常検知のためのデータとして, プロセス毎の各資源（CPU, メモリ，ネットワーク）の使用率の時系列データを取得する．各資源の使用率は周期 T 毎に以下の計算式によって求める．CPU については, あるプロセス p が, 周期 T で利用した CPU の使用率 U_{cpu} は, 式（1）で表せる．ここで T_p は，T 周期の中で p が使用した時間を示す.

$$U^P_{cpu} = T_p/T \tag{1}$$

メモリは, システム全体のメモリ量を M, プロセス p がある周期 T で使用したメモリの量を m とすると, メモリの利用率 U^P_{mem} は式（2）で表せる.

$$U^P_{mem} = \Sigma^n_{i=0} m_p/M \tag{2}$$

ネットワークは，周期 T 内でのプロセス p のデータ転送量を P_d, ネットワークの平均帯域を N_a とすると, ネットワーク使用率 U^P_{net} は式（3）で表せる.

$$U^P_{net} = P_d/N_a \tag{3}$$

時系列データは, 標本周期 T ごとに (1), (2), (3) の計算により統計値を取得する．この時の資源利用率はそれぞれ $U_{cpu}, U_{mem}, U_{net}$ とする．これらを資源セット ($u = \{U_{cpu}, U_{mem}, U_{net}\}$) とし, 周期ごとに取得した時系列情報としてベクトル化する ($vec_i = (U_{i,cpu}, U_{i,mem}, U_{i,net})$).

ベクトルデータをさらに，k-means 法でクラスタリングした上で HMM で学習させる．クラスタリングは類似度の高い部分集合に分割する手法であり，本研究ではその中でも分割最適化クラスタリングの代表的手法である k-means 手法を用いた[7].

3.3 HMM による学習と判定

クラスタリングによって得られた時系列データを, 観測時系列 V とする．V はクラスタリングにより作成した時系列データ $V = \{v_0, v_1, ..., v_k, ..., v_{K-1}\}$ である．出力の時系列 $\{V(t)\}^T_{t=0}$，状態の種類は $S = \{s_0, s_1, ..., s_n, ..., s_{N-1}\}$ とする．ここで，状態遷移確率 $A = \{a_{ij}\}, a_{ij} = P(S(t+1) = j|S(t) = i)$, 出力の出現確率 $B = \{b_i(k)\}, b_i(k) = P(V(t) = v_k|S(t) = i)$, 初期状態の確率 $\Pi = \{\pi_i\}, \pi_i = P(S(0) = s_i)$ とし，モデルは $\theta \equiv (\Pi, A, B)$ で表す.

次に HMM により正常状態モデルを学習する.HMM 出力シンボル系列から, Baum-Welch アルゴリズムにより状態遷移確率等のパラメータの最尤推定値を求める．HMM のモデルでは任意の状態へ遷移可能なエルゴード性を仮定する．マルコフモデルの状態遷移確率 a と, 各状態におけるシンボル出力確率 b が得られる．状態遷移確率 a_{ij} の推定値 $\hat{a}_{ij}$ は式（4）, 状態 s_j において出力シンボルが v_k である確率の推定値 $\hat{b}_j(k)$ は式（5）で求められる.

$$\hat{a}_{ij} = \frac{\sum^{T-1}_{t=0} \gamma_{ij}(t)}{\sum^{T-1}_{t=0} \sum_j \gamma_{ij}(t)} = \frac{\sum^{T-1}_{t=0} \gamma_{ij}(t)}{\sum^{T-1}_{t=0} \gamma_i(t)} \tag{4}$$

$$\hat{b}_j = \frac{\sum_{t \in V(t)=v_k} \gamma_j(t)}{\sum^{T}_{t=0} \gamma_j(t)} \tag{5}$$

ここで，初期状態の推定は式 (6) となる.

$$\hat{\pi}_i = \gamma_i(0) \tag{6}$$

学習によって, 観測系列 $\{V(t)\}^T_{t=0}$ が与えられた時の状態遷移確率の推定値 $\hat{a}_{ij}$ と，各状態におけるシンボル出力確率 $\hat{b}_j$ が得られる. ここで a_{ji} は, 状態 j から状態 i へ遷移する確率, $b_i(k)$ は状態 i でシンボル k を出力する確率をしめす．最終的に $\theta_{new} = (\hat{\pi}_i, \hat{a}_{ij}, \hat{b}_j(k))$ を新たなパラメータとして $P(V|\theta_{new}) \geq P(V|\theta_{old})$ を繰り返し，最適なパラメータを計算する．最適なパラメータが求められた時点で確率 $P(V|\theta)$ を求める．別の出力シンボルの列が $\{V(t)\}^T_{t=0}$ も同様に計算を行い, 学習時の $P(V|\theta_{learn})$ を求める．判定においては式 (7) によりモデルの一致，不一致を判定する．ω は適合率の閾値である.

$$P(V|\theta_{train})/P(V|\theta_{learn}) > \omega \tag{7}$$

機械学習による異常検知の詳細については[7] に詳細を述べた.

4. 評　　価

4.1 実験の目的と概要

本実験では，異常の学習による検知と，正常学習による異常の検知の両方の学習の精度を検証する.

実験の題材として CPU の資源占有を生じる顔認識アプリケーションと，メモリリーク障害を発生させるアプリケーションを用いて，異常状態と正常状態を学習させ，本手法による異常の検知結果から有効性を検討する.

4.2 実験対象の障害

実験対象の障害として，組込みシステムで発生しう

Vol.2010-SLDM-144 No.62
Vol.2010-EMB-16 No.62
Vol.2010-MBL-53 No.62
Vol.2010-UBI-25 No.62
2010/3/28

表 1 実験環境

CPU	Intel(R) Core(TM)2 Duo CPU 2.40GHz
Memory	1,017,128 kB
Kernel	2.6.24 Ayaka (Unip)
Distribution	Ubuntu 9.04

表 2 実験環境

学習データの取得	正常/異常プログラムの動作を学習
検知時間	120 秒
判定回数	50 - 100 回
データ取得周期	100ms, 500ms, 1000ms
適合率の閾値	0.01, 0.1, 1
状態遷移数	8
HMM モデル	Ergodic
クラスタセントロイド数	14

る障害を実験対象とした．組込みシステムでの課題として取り上げたプログラムのダウンロードや設計ミスなどによる過負荷，アプリケーションの資源占有を含む 3 つの障害を主な障害として調査を行うものとした．

- 資源占有
- エージング（メモリリーク）

プログラムの資源占有は，プログラムがなんらかの欠陥をもつことで資源を占有する場合と，欠陥がない場合であっても，動作条件により発生する．特に未検証のアプリケーションは，設計が十分であるとは限らず，こうしたプログラムがシステムに追加される場合などに発生しやすい．

メモリリークは，プログラムが永続的な欠陥を持つことによるバグの一種である．プログラムが確保したメモリの一部，または全部の開放し忘れ等の単純なミスやプログラムの論理的欠陥によって発生する．特に組込みシステムでは，メモリ資源に制限があることが多く，プログラムのメモリリークにより，たとえ仮想メモリを利用していたとしてもシステムのページングやスラッシングの頻発や仮想メモリの枯渇によりシステムが停止するなどの深刻な問題になる．

4.3 実験方法

実験方法としては，障害を発生するプログラムと，正常状態のまま動作する二つのプログラムをランダムに実行し，提案システムにより検知が正しく行われるかを調査した．実験では次の二つのケースを調査した．

- 正常アプリケーションの動作を学習し，異なる状態を異常として検知する
- 異常アプリケーションの動作を学習し，異常として検知する

4.3.1 実験環境

実験環境を表 1 に示した．

4.3.2 基本パラメータ

- HMM パラメータ：学習および検知にあたり，HMM の動作に影響を与えるパラメータを変更可能とし，それぞれ検知結果にどのように影響を与えるかを示す．下記に変更を行うパラメータを示す．
  - データ取得周期：100ms, 500ms, 1000ms
  - 適合率の閾値：0.01, 0.1, 1
  - 状態遷移数：8
  - HMM モデル：Ergodic
  - クラスタセントロイド数：14

また，実験では 0.01, 0.1, 1 の 3 つの適合率の閾値を比較した．0.01 は適合率が 1%である事を示し，低い適合率であってもモデルが一致したとみなす．逆に 1 はモデルが 100%一致していることを示す．学習時と検知時の適合率の組み合わせを変えて実験を行った．

4.3.3 判定方法

本研究においては，実験の手順に述べたように，正常と，異常の二つを学習対象とする．学習対象となるアプリケーションが，正常と異常の二つあり，検知は正常，異常のアプリケーションで行う．

評価は正常を正常と正しく判定 (TN)，異常を異常と正しく判定 (TP), 見逃し (FN)，誤検知 (FN) の 4 つとし，それぞれ表 3 をもとに検知結果を精度，再現率，正確さにより評価する．

表 3 分割表から得られる指標

指標	式
精度 (precision)	$\frac{TP}{TP+FP}$
再現率 (recall)	$\frac{TP}{TP+FN}$
正確さ (accuracy)	$\frac{TP+TN}{TP+TN+FP+FN}$

4.4 障害の学習

CPU 資源占有の障害は OpenCV の顔認識プログラム[8),9)] を用いた．顔認識アプリケーションはバグを含まない．しかし，顔認識数が異常に増加した場合や，性能が低いマシンで動作させるなど仕様に異なる環境で動作させた場合に CPU 資源占有を発生する．実験では，CPU を使い尽くす状態を再現した顔認識アプリケーションを異常動作アプリケーションとし，使い尽くさずに作するアプリケーションを正常動作アプリケーションとして用いた．正常動作では探索ウィンドウのスケールを 10 倍にし，処理負荷を軽減させた状態を維持する改変を行った．

メモリリークを示すプログラムは，正常なメモリ処

Vol.2010-SLDM-144 No.62
Vol.2010-EMB-16 No.62
Vol.2010-MBL-53 No.62
Vol.2010-UBI-25 No.62
2010/3/28

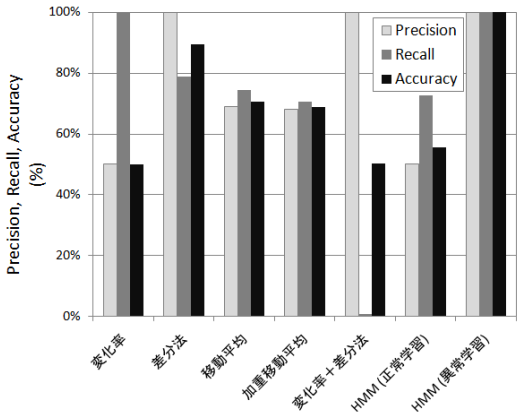

図 4 複数手法の比較

理を行うプログラムと，メモリリークを発生させるプログラムの二つを作成した．

正常なメモリ処理を行うプログラムでは，4,096K の malloc を 100 回繰り返し，メモリの開放処理では確保した領域を開放する処理を行う．異常動作の場合は，100 回確保した領域に対してメモリを開放する領域は 90 回分とした．10 回分確保した領域が開放漏れを起こすものとした．

4.5 実験結果

4.3.2 節の手順に沿って学習と検知を行った結果を集計した．

4.5.1 CPU 資源占有アプリケーションの異常検知

異常学習による異常検知，正常学習による異常検知の有効性（未知の異常への対応可能性）を確認するために，閾値，サンプリング周期の 2 つのパラメータの組み合わせを変更して検知テストを行い，図 2, 図 3 に結果 (紙面の制限上，100ms, 1000ms のみを) 掲載した．

4.5.2 (1) 閾値変更の結果比較

精度，再現率，正確さの 3 つの評価で 100%の結果であったのは，異常学習時の周期 100ms では学習時閾値 1, 検知時閾値 0.1, 閾値 0.01，1000ms では学習時閾値 0.01 では，0.1, 学習時閾値 0.1 では 0.01 と 0.1, 学習時閾値 1 では 0.01,0.1,1 と 3 つともで 100%の検知精度となった．この事から (1) 正常学習と異常学習では，異常学習の検知精度，再現率，正確さいづれも高い (2) 周期は 1 秒の方が検知精度が良い (2) 学習時閾値が高い方が検知精度が高いことがいえる．それぞれ (1) 正常よりも異常の方が特徴抽出がしやすいこと (2) CPU の異常のように，長い周期で変化が現れるものについては 1 秒の周期の方が特徴が抽出しやすかったこと (3) 学習時に作成するモデルの適合率を厳しく設定することで，モデル判定時に基準となる厳しいモデルが作成できることなどがあげられる．今回の HMM では正常学習による正常検知，異常学習による異常検知など同じパターンの検出は比較的高い精度で行えたが，逆のパターンを検知することは難しいことがわかった．

4.5.3 (2) 統計アプローチとの比較

図 4 に，他の複数手法との比較結果を示した．HMM の結果は最も指標の総合値が高かった異常学習の結果と正常学習の結果を比較として用いた．HMM の異常学習に関しては他の統計手法と比較し，精度，再現率，正確さとも 100%で結果は他のどの統計手法よりも優れている．ただし，正常学習時の異常検知は最も悪い手法から数えて二番目である．しかし，正常学習は，未知の障害に対応できる可能性をもっている．この方法が既存の手法と同程度の結果が得られたことは，今後の未知障害の検知の可能性という意味では期待が持てる結果といえる．

4.6 メモリリーク障害アプリケーションの異常検知

4.6.1 (1) 周期，閾値の比較

メモリリーク障害の検知結果を図 5 に示した．検知結果では，サンプリング周期が 1000ms, 500ms では全く検知が行われず，サンプリング周期が 100ms のみ検知がなされている．ただし，正常学習による異常検知はできていない．100ms などサンプリング周期が短い場合には，より詳細にデータを取るため，メモリ操作を行っていない時間が確率的に多くなる．このために，モデルを作成することができていると考えられる．

4.6.2 (2) 検知時間の延長

メモリリーク障害の検知時間を 300 秒に変化した結果を図 6 に示した．結果より，サンプリング周期 500ms においても検知がなされるなど 180 秒よりは改善していた．しかし，精度はいずれの結果も 50%から 10%と低く，誤検出が多いことを示している．再現率は精度に比べて高いといえるため見逃し数は少ないといえるが，実用に十分な検知結果とはいえない．

4.6.3 (3) メモリの入力パラメータのスケーリング

メモリリークを検知するためには，メモリリーク自体の変化量をより明確にモデルに反映させることが有利に働くと考え，メモリの値として取得した値のスケールを 10 倍，100 倍，1000 倍と増加させて，検知精度に変化がないか実験を行った．サンプリング周期については 100ms 固定として，10 倍，100 倍，1000 倍にメモリの値のスケールを変更し結果を図 7 に示

Vol.2010-SLDM-144 No.62
Vol.2010-EMB-16 No.62
Vol.2010-MBL-53 No.62
Vol.2010-UBI-25 No.62
2010/3/28

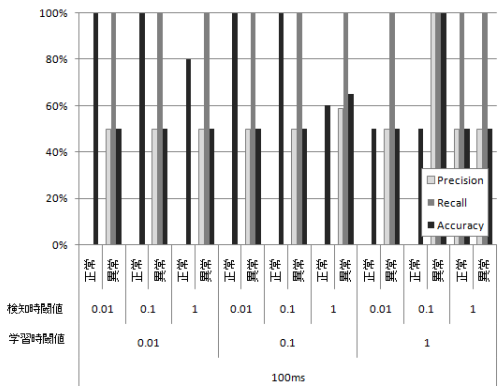

図 2 精度，再現率，正確さ (100ms)

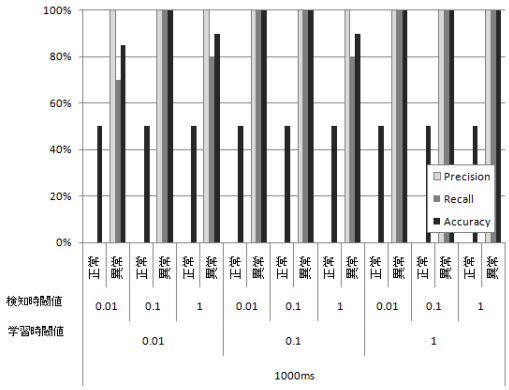

図 3 精度，再現率，正確さ (1000ms)

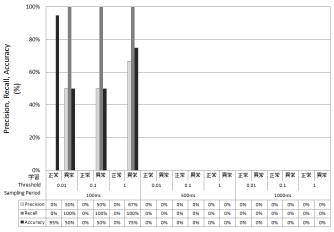

図 5 メモリリーク検知結果 (120sec)

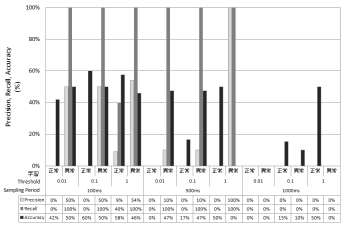

図 6 メモリリーク検知結果 (300sec)

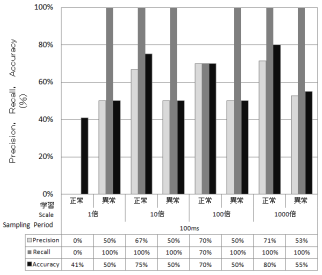

図 7 メモリリーク (スケールの変更)

した．結果より，10 倍，100 倍，1000 倍の異常学習，異常検知の精度と正確さについては効果が確認できなかった．しかし，正常学習，異常検知に関しては，10 倍，100 倍，1000 倍については精度，再現率，正確さいずれの結果も改善された．周期が長い場合には検知がなされていないことからも，焦点を当てる特徴量の変化をスケールして判別しやすくすることで，特徴抽出ができるといえよう．

4.7 オーバーヘッド調査

監視によるオーバーヘッドの評価を行った．初めにシステム全体にかかるオーバーヘッドを計測した．計測方法は，モニタリングシステムを動作しない状態 (Normal)，学習中 (Training)，運用中 (Runtime) 時に，それぞれバックグラウンドで動かしたプログラムがかかった時間を スタンドアロンマシン上で計測した．プログラムは単純な行列計算で，実行時間を CPU のサイクルカウンターで計測し，1,000 回実行したときの平均値，最大値，最小値，分散値をそれぞれ示した．さらにモニタリングシステムを動作させていない場合を 1 とした場合の比率を (A) に示し，1 から (A) の値をマイナスした値を (B) とした．(B) はオーバーヘッドの比率を表す値として用いることができる．(B) の結果から，学習時のオーバーヘッドは 1.1%，動作中のオーバーヘッドは 1.2%となった．

表 4 システムオーバーヘッド
Table 4 System Overhead

実行時間 (second)	モニタリング無	学習時	運用時
平均実行時間	1.902	1.923	1.925
最大	1.950	1.978	1.970
最小	1.899	1.922	1.922
(A) モニタリング有/無	1.000	1.011	1.012
(B) {(A) - 1 }%	-	1.10%	1.20%

5. 関 連 研 究

異常検出システムにおいては，予めアプリケーションの正常状態をモデル化し，そのモデルへの適合を動作を監視するとい方法が一般的である．モデル化には大きく分けて二つの方法がある．一つは論理性を保証する方法，もう一つはイベント発生手順をヒューリスティックに記述する方法である．論理性保証の代表に形式手法がある[10][11]．[10] では DoS アタックを，処理ステップごとの形式手法によりモデル化する手法を示している．[11] では SPIN のモデルチェッカー[12] を用いて記述し，その結果を分析した内容を示している．しかし，フォーマルメソッドは通常のプログラムと文

Vol.2010-SLDM-144 No.62
Vol.2010-EMB-16 No.62
Vol.2010-MBL-53 No.62
Vol.2010-UBI-25 No.62
2010/3/28

法と異なり様式での記述が必要なうえ，検査プログラムには，実際のプログラムをベースに抽出したモデルが求められるため，汎用的な問題に対応させる事が難しい．

異常検出に確率やクラスタリング手法を取り入れ．ランタイム検出を行う試みが，Cohen らによって行われている[13]．Cohen らは，故障解析の手法として Tree-Augumented Naive Baysian network (TAN)[14] をもちいた手法を提案した．Cohen らは正常，異常状態の判定を行う Service-level Objective (SLO) を定義し，SLO により異常と判定されたデータをさらに，クラスタリングすることで異常の特徴をより詳細に特定し，故障を特定する方法を示した．この手法は特徴抽出にクラスタリングを活用することで異常の種類を明確にし，既知の異常かどうか状態遷移も含めて把握できる．しかし，Cohen らは並列分散計算を前提としているため，全ての分散ノードでのマシン構成，アプリケーションが同一てあることが前提とされている．こうした前提は我々の要求とは異なる．Cohen が行った各統計量の 2 値化，すなわちリアルタイムに学習と異常検出を行う手法は効率的であるため, 我々も採り入れた．しかし，異常，正常の基準を最終的には SLO に依存しているため，想定しない問題があった場合であっても最初の SLO で見逃してしまった場合には見逃しが発生する問題がある．我々は正常状態はあくまでもテスト段階で作成することが必要であると考え，異常, 正常判定を事前に与えることは避けた．

6. 結　　論

本論文では機械学習を用いたオンライン監視による異常検知手法を提案した．オンライン監視ではオペレーティングシステム内で取得するプロセスごとの資源情報を利用した機械学習により (1) 自律性 (2) 汎用性 (3) 低オーバーヘッドな監視システムを実現した．また，評価結果より，高い検知精度とていオーバーヘッドを実現することを示した．また，閾値やサンプリング周期のパラメータによって, 検知精度が異なることがわかった．今後，より検知精度を高めるように，最適なパラメータを調査する．また，未知障害の検知精度を上げることに取り組む必要があると考える．

参 考 文 献

1) 経済産業省. 2007 馬版 組込みソフトテデヂ孜業実態調査報告書. 2007.
2) Ganglia monitoring system. http://ganglia.sourceforge.net/.
3) Nagios. http://www.nagios.org/.
4) 大山恵弘 神山貴幸．ビールベタッハ情報を利用したモデル分割に基づく異常検知プベテム．**情報処理学会論文誌，コンピューティングシステム**, 2(1):12–22, March 2009.
5) High resolution timer. http://lwn.net/Articles/167897/.
6) M.Desnoyers and M.R. Dagenais. The lttng tracer: a low impact performance and behavior monitor for gnu/linux. In *Proceedings of the Linux Symposium*, volume1, pages 209–224, 2006.
7) Tatsuo Nakajima Midori Sugaya, Yuki Ohno. Lightweight anomaly detection system with hmm resource modeling. *International Journal of Security and Its Applications*, 3(3):to be published, 2009.
8) Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.
9) Face Recognition using OpenCV. http://opencv.willowgarage.com/wiki/FaceRecognition.
10) Catherine Meadows. A formal framework and evaluation method for network denial of service. In *In Proceedings of the 1999 IEEE Computer Security Foundations Workshop*, pages 4–13. Society Press, 1999.
11) Daigo Tomioka, Shinya Nishizaki, and Ritsuya Ikeda. A Cost Estimation Calculus for Analyzing the Resistance to Denial-of-Service Attack. *In Software Security Theories and Systems*, 3233 of Lecture Notes in Computer Science:25–44, 2004.
12) Gerard J. Holzmann. The model checker SPIN. *Software Engineering*, 23(5):279–295, 1997.
13) Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons, and JeffreyS. Chase. Correlating instrumentation data to system states: a building block for automated diagnosis and control. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 16–16, 2004.
14) Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. In P.Smyth G.Provan, P.Langley, editor, *Machine Learning*, number 29, pages 131–163. Kluwer Academic Publishers, 1997.