HTTP ステータスコードを含むログとメトリクスの対応関係を用いたアラート件数の削減

坂井 萌桜[1]　平尾 真斗[2]　串田 高幸[1]

概要：東京工科大学コンピュータサイエンス学部の Cloud and Distributed Systems Laboratory(以下 CDSL と呼ぶ) では，インターネットに公開しているサイトである Doktor を，Prometheus で監視し，ログを Elasticsearch に保存している．Prometheus は Doktor の HTTP ステータスコードのメトリクスを取得し，その値が 200 以外になった場合にアラートを発行する．Alertmanager は Prometheus で発行されたアラートを Redmine に送信し，Redmine でチケットが作成される．課題は，Alertmanager から Redmine に送信されるアラートの中に，Prometheus の誤検知が含まれていることである．提案では Redmine にチケットが作成された際，Prometheus が取得した対象の HTTP ステータスコードのメトリクスの値と，その時刻の前後に Elasticsearch に記録された HTTP リクエストのログのステータスコードを 1 つのデータセットとして保存する．調査者はチケットの作成時刻と，その前後の時間に記録されたログを確認し，データセットに対して「要確認」か「問題なし」かのラベル付けを行う．チケット作成時には，新たに作成されたデータセットに含まれる Prometheus が取得した HTTP ステータスコードのメトリクス値と，Elasticsearch に記録された HTTP リクエストのログのステータスコードを過去のデータセットに含まれる同じ値の組み合わせと比較する．一致する場合には，過去のデータセットに付けられた「要確認」か「問題なし」のラベルを新しいセットに付ける．一致しない場合は「要確認」のラベルをつけアラート通知を行う．「問題なし」の場合にはアラート通知は行わない．実験では，提案手法の適用により削減できるアラート通知の数を評価する．2025 年 11 月 24 日から 11 月 26 日において作成された 55 件のチケットに対応するデータセットを実験の対象とする．そのうち 11 月 24 日に作成された 25 件のデータセットについては調査者がログを確認しラベル付けを行い，提案手法が使用する過去の複数のデータセットとした．調査者は CDSL に所属する学生 4 人である．ラベル付けの結果，全員が「問題なし」と判断したデータセットは 25 件中 7 件で，全員が「要確認」と判断したデータセットはなかった．また，4 人中 1 人が「要確認」と判断したデータセットが 6 件，2 人が「要確認」と判断したデータセットが 5 件，3 人が「要確認」と判断したデータセットが 7 件だった．全員が「問題なし」と判断した過去のデータセットのみを「問題なし」のデータセットとして使用し，11 月 25 日から 11 月 26 日に作成された 30 件のデータセットを評価対象として提案手法を適用した．その結果，30 件中 6 件が「問題なし」とラベル付けされた過去のデータセットと一致したため，アラート通知の対象外となることが確認でき，残りの 24 件は「要確認」となりアラート通知の対象となることが確認できた．

1. はじめに

背景

産業や社会活動は IT システムに依存しており，これらを安定的に運用することの重要性が高まっている [1]．安定的なシステムの運用を維持するためには，システムの稼働状況を継続的に監視し，異常を早期に発見し，的確に通知する仕組みが不可欠である [2]．システム監視ではメトリクスやログが使用され，これらに異常があった場合，必要に応じてシステムの管理者にアラート通知を行う．メトリクスとは，CPU 使用率やメモリ使用量，レスポンス時間，エラー率を例とする，システム内で継続的に発生する，時系列データとして扱われる数値データのことである [2]．ログとは，システムの活動を時系列で記録したものである [3]．ログには，タイムスタンプ，イベント内容，エラーの有無が含まれている．エンジニアはログを閲覧することでシステムの動作や，発生したイベントを把握することができる [4,5]．

東京工科大学コンピュータサイエンス学部の CDSL で

[1] 東京工科大学コンピュータサイエンス学部
〒 192–0982 東京都八王子市片倉町 1404-1
[2] 東京工科大学大学院バイオ・情報メディア研究科コンピュータサイエンス専攻
〒 192–0982 東京都八王子市片倉町 1404-1

は，Doktor というサイトを運用している[*1]．監視ツールとして Prometheus を使用しており，ログについては ELK Stack と Filebeat，Elastalert を使用して管理，収集，アラート通知を行っている．また，通知されたアラートをチケットとして管理するために Redmine をもちいている．Prometheus とは，メトリクスをシステムやアプリケーションから収集するオープンソースのソフトウェアである[*2]．Prometheus は，監視対象のメトリクスに対して管理者が任意の条件でアラートルールを定義できる仕組みを提供している．設定した条件に沿って外部の通知システムへアラートを送信するように構成することができる．Grafana とは Prometheus で収集したメトリクスの数値を可視化することができるオープンソースのデータ可視化プラットフォームである．使用することでメトリクスの変動をグラフとして表示することができる [6]．ELK Stack とは Elasticsearch，Logstash，Kibana の 3 つのプロジェクトで構成されるスタックである [7]．Filebeat とは，Logstash にログを転送するためのログジッパーである [8]．ログの収集は Filebeat が行い，収集したログの構造化は Logstash が行う．ログの管理は Elasticsearch が行い，Elasticsearch で管理しているログの可視化を Kibana が行う [9, 10]．Elastalert とは，Elasticsearch で管理しているログを検索し，アラートルールとして設定した条件に一致した場合にアラートを通知する，オープンソースのツールである．Redmine は，タスクや問題の進捗をチケットとして管理する，オープンソースのプロジェクト管理ツールである．Prometheus と Elastalert で検知されたアラート内容を Redmine のチケットとして作成する構成としている．

CDSL で運用している Doktor を監視対象としている．Doktor は CDSL に所属している学生が作成したテクニカルレポートをインターネットに公開するためのサイトである．Doktor と監視システム，ログサーバの構成を図 1に示す．

Doktor は 1 台のマスターノードと 3 台のワーカーノードから構成される Kubernetes クラスタ上に構築されている．4 台のノードは ESXi 上をインストールした物理サーバ上に配置されている．クラスタ内では複数のサービスが稼働している．例えば front，stats，author，paper がある．各サービスは 1 つ以上の Pod から構成されており，Pod 内にはアプリケーション本体のコンテナと，通信を中継・制御するサイドカーコンテナである Istio-proxy が含まれている．Doktor の利用者からのアクセスは，Google Cloud Platform 上の Nginx を経由して VPN でクラスタ内の Istio Ingress Gateway に到達する．Istio Ingress Gateway に到達したリクエストは，front に送られ，front から各サービスに通信が行われる．

[*1] https://doktor.tak-cslab.org/(参照 2025/10/30)
[*2] https://prometheus.io/ (参照 2025/10/29)

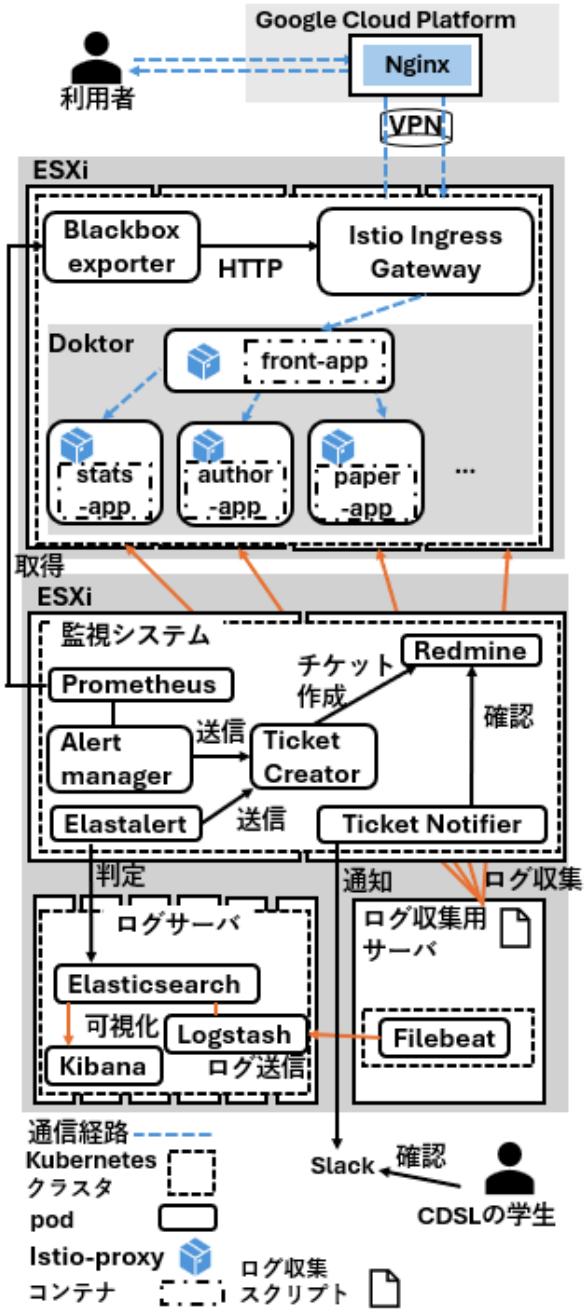

図 1: Doktor と監視システムとログサーバの構成

監視システムは 1 台のマスターノードと 1 台のワーカーノードから構成される Kubernetes クラスタ上に構築されている．クラスタ内には Prometheus，Alertmanager，Redmine, Ticket Creator, Ticket Notifier がある．Prometheus が Doktor の HTTP ステータスコードのメトリクスを取得する際には，Blackbox exporter を使用する．Blackbox exporter は HTTP リクエストを送信し，Doktor にアクセスすることでステータスコードを監視したメトリクスを取得する．Prometheus は Blackbox exporter が取得したステータスコードのメトリクスを時系列データとして保存する．Prometheus は設定したアラートルールの条件に一致した場合，Alertmanager にアラートを送信する．具

体的には Blackbox exporter が Doktor に送信した HTTP リクエストに対し，返された HTTP ステータスコードが 200 以外である場合に，そのメトリクスを Prometheus が取得し，Alertmanager にアラートを送信する．Elastalert は Elasticsearch 上に保存されている Doktor のログを監視し，アラートルールの条件に一致するログがあるかを判定し，アラートを通知する．Ticket Creator は Alertmanager から送信されたアラート，Elastalert から送信されたアラートを受け取り，Redmine 上にチケットを作成する．Ticket Notifier は，Redmine にチケットが作成された時に Slack へ通知を行う．CDSL の学生は Slack でアラート通知を受け取り，内容を確認する．

ログ収集用サーバは 1 台のノードから構成される Kubernetes クラスタ上に構築されている．クラスタ内には Filebeat があり，ノードではログ収集スクリプトが 1 分間隔で実行されている．スクリプトは，Doktor を構成する 4 台のノードの/var/log/containers にあるログを取得する．取得したログは，ログ収集用サーバのノードに保存され，Filebeat はこれをログサーバ上にある Logstash に送信する．

ログサーバは 1 台のマスターノードと 5 台のワーカーノードから構成される Kubernetes クラスタ上に構築されている．これらのノードは，監視システム，ログ収集用サーバと同じ ESXi 上に配置されている．クラスタ内には Elasticsearch，Logstash，Kibana がある．Logstash は Filebeat から送信されたログを構造化し，Elasticsearch に送信する．Kibana は，Elasticsearch に蓄積されたログを可視化している．

運用しているアラートルール

運用しているアラートルールは, Prometheus と Elastalert がある．Prometheus は Blackbox exporter が監視した HTTP のステータスコードのメトリクスが設定された閾値の値を超えた場合にアラートを作成する．また, 作成されたアラートは，Alertmanager が通知を担当する．Elastalert では Elasticsearch に蓄積された Doktor のログをもとにアラートを作成している．CDSL では，Prometheus が収集したメトリクスを元にしたアラートルールが 119 個，Elastalert のアラートルールを 12 個設定している．

例として対象とするアラートルールの内容と，運用の目的について説明する．Prometheus では，Blackbox exporter を使用して Doktor に対する HTTP リクエストを送信し，取得した HTTP ステータスコードを監視している．監視対象となる URL はノード IP 直下，ノード IP/paper/UUID，ノード IP/author/UUID であり，これらはいずれも監視対象アプリケーションのエンドポイントである．HTTP ステータスコード 200 以外が返された場合にアラートを作成するルールを設定している．このアラートルールの運用目的は，Doktor のシステムにおける一時的な HTTP ステータスコードの変動を検知することである．HTTP ステータスコードの変動をシステムが不安定になる兆候として捉え，検知する設定にしている．

課題

課題は，監視システムによって通知されるアラートの中に，システム運用においては異常とみなさないものが含まれていることである．課題の概要を図 2に示す．

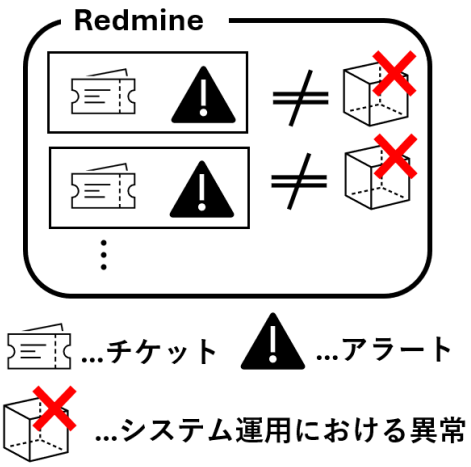

図 2: 課題の概要

2025 年 10 月 19 日 4 時 13 分に確認されたケースを例として取り上げる．この時刻において，Blackbox exporter がノード IP 直下に対する HTTP リクエストでステータスコード 503 を観測し，Prometheus によりアラートが作成された．前後の時刻における Blackbox exporter 監視対象へのアクセスログにも HTTP ステータスコード 503 が確認された．しかし，アプリケーション本体のログには HTTP ステータスコード 503 が記録されておらず，エラーを示すログも記録されていなかった．このようなケースはシステム運用においては異常とみなさないことが多いが，このようなケースのアラートを抑制するためにアラートの感度を下げることは，通知されるアラートの件数を減らすことができるが，システム運用における異常を見逃すケースが増加する．Doktor に対する，HTTP ステータスコード 200 以外になった場合にアラートを作成するアラートルールは，システムが不安定になる兆候を捉えることを目的として設定されており，一時的な異常も検出する．このようにアラートの感度が高い場合はアラート通知数が増加する．システムが通常稼働している場合にもアラートが通知されるようになり，システム運用において真の異常を見逃すリスクと運用のコストが増加する [11]．システム運用における異常の検出と削減はトレードオフの関係にある [12, 13].

各章の概要

第 2 章の関連研究では，関連する既存研究について述べる．第 3 章では課題を解決する提案方式と，ユースケースシナリオについて述べる．第 4 章の実装では，実装方法について述べる．第 5 章の評価実験では，実験環境，実験結果とその分析について述べる．第 6 章では提案方式の議論を行う．第 7 章は本稿におけるまとめである．

2. 関連研究

システムのアラート管理において，手動で設定された静的な閾値が誤検知や異常の見逃しを引き起こすという課題に対し，システムの運用ログやメトリクスを分析し，動的に閾値を設定する仕組みを提案している研究がある [14]．システムの通常状態を過去のメトリクスからモデル化し，変化点分析や分類回帰木をもちいることでシステム状態の変動に応じた柔軟なアラート条件を自動的に生成している．また，複数の関連アラートを統合する手法や，将来の異常を予測するモデルを導入することで，誤検知の削減と予防的な監視を可能にしている．この研究では，通知されたアラートが監視システム側の誤検知によるものか，実際の監視対象システムの異常によるものかを識別する点については扱われていない．

アラート通知の精度向上や有用なアラートの抽出を目的として，ログに内在するパターンの変化に着目し，エントロピーを指標として注目すべきログを抽出・分類する手法を提案している研究がある [15]．ログ内に含まれる単語や構文を統計的に分析し，出現頻度の変化をもとに対数エントロピーを算出することで，システムの異常な動作を検知している．そのため，事前の学習データを必要とせずにログ内の文章の変化から異常を特定できる．この研究では，検出された異常の原因の切り分けまでは扱われていない．

クラウド環境におけるシステム運用時の異常検知を目的として，ログとメトリクスを組み合わせた相関分析による異常検知手法を提案している研究がある [16]．アプリケーションログと監視メトリクスを時系列的に対応づけ，それぞれの変動パターンの相関を分析することで，単一のデータソースでは検知が困難な異常を発見する．この研究は，検出された異常が監視システム側の誤検知によるものか，対象システムそのものの障害によるものかどうかの原因の識別については扱われていない．

ネットワーク運用におけるアラート管理の効率化を目的として，管理者の業務負荷や対応可能時間を考慮したアラートの制御を行う手法を提案している研究がある [17]．ネットワーク機器から得られる時系列メトリクスから，システムの異常を自動的に検知し，運用スケジュールに応じてアラートの通知タイミングや通知優先度を動的に調整している．これにより，アラート担当者が不在の時間帯や過剰な通知による運用負荷の増大を抑制することができる．この研究では，アラートルールの閾値の設定やメトリクスの異常値の定義については検討されていない．

3. 提案方式

本提案では，監視システムで監視対象の HTTP ステータスコードが 200 以外が確認された場合に，その前後の時間における監視システムから監視対象へのリクエストを記録したログを取得し，メトリクスと合わせて 1 つのデータセットとして保存する．保存されたデータセットに対して，調査者が「要確認」か「問題なし」のラベル付けを行い，これを過去のデータセットとして蓄積する．新たにアラートが作成された際には，取得したログとメトリクスのデータセットを過去のデータセットと比較し，同じデータセットがある場合には，過去のラベルを参照して「要確認」か「問題なし」かの判断を行う．過去に同様のデータセットがない場合，調査者にラベル付けを行うための通知を行う．提案手法の具体的な適用例として CDSL で運用している Doktor をもちいる．提案の概要を図 3に示す．

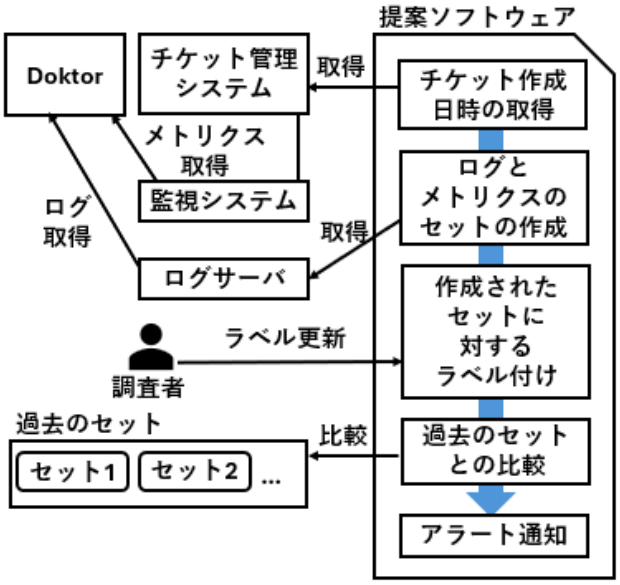

図 3: 提案の概要

監視システムとログサーバは監視対象からそれぞれログとメトリクスを取得している．監視システムで設定されているアラートルールの閾値を超えた場合，チケット管理システムでチケットが作成される．提案ソフトウェアはチケット作成日時，メトリクス，ログをそれぞれ取得，データセットを作成し保存する．新しく作成されたデータセットは過去のデータセットとの比較を行い，提案ソフトウェアがラベル付けを行う．過去に同様のデータセットがない場合は「要確認」のラベルを付ける．ラベルが「要確認」の場合はアラートがチケット管理システムに送信され，チケットが作成される．調査者はチケットを調査して，新し

く作成されたデータセットに対して「要確認」か「問題なし」かのラベル付けを行う．「要確認」のラベルが付いたデータセットは再度アラート通知を行うデータセットとして残しておく．

チケット作成日時の取得

監視対象に対して設定しているアラートルールでは，監視システムで取得された HTTP ステータスコードが 200 以外の場合にチケット管理システムでチケットを作成し，アラートを通知する．そのため，チケットの作成日時は異常が検出された時刻を表す．

ログとメトリクスのデータセットの作成

チケットの作成日時を起点として，その前後の時間における監視システムによる GET リクエストに対応するアクセスログを取得する．作成されるデータセットの内容を図 4に示す．

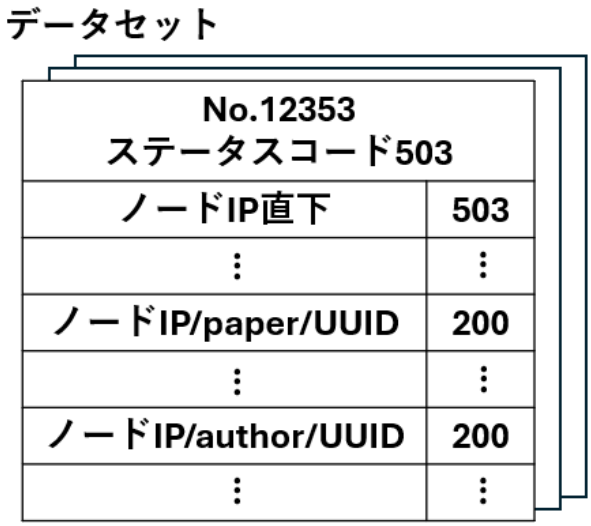

図 4: 作成されるデータセットの内容

監視システムによる監視の対象となる URL は，ノード IP 直下，ノード IP/paper/UUID，ノード IP/author/UUID の 3 種であり，これらは監視対象のアプリケーションのエンドポイントである．取得したログに含まれる HTTP ステータスコードと，異常を検出したターゲット URL の組をパターンとして扱う．これらをメトリクスと合わせて 1 つのデータセットとして保存する．

ログを取得する際の時間幅の決定

チケットの作成日時を起点として，その前後の時間における監視システムによる GET リクエストに対応するアクセスログを取得する．この時，チケット作成日時を t とし，t を中心とした前後の時間幅をそれぞれ ΔT_1，ΔT_2 と定義する．チケット作成日時 t と時間幅 ΔT_1，ΔT_2 の関係について図 5および式 1に示す．

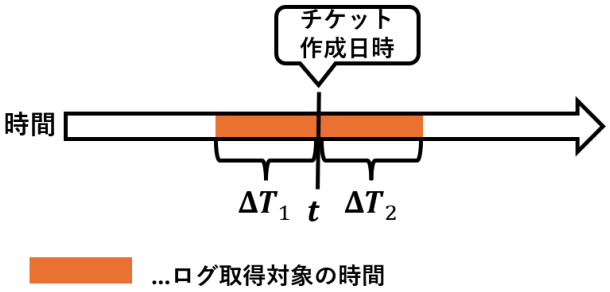

図 5: チケット作成日時 t と前後の時間幅 ΔT_1，ΔT_2 の関係

$$\text{ログ取得対象の時間} = [t - \Delta T_1,\ t + \Delta T_2] \tag{1}$$

$t - \Delta T_1$ から $t + \Delta T_2$ の時間に記録された監視システムによる GET リクエストに対応するアクセスログを取得の対象とする．取得対象の時間幅を短く設定すると不要なログの取得を避けられる一方で，監視システムによる GET リクエストに対応するログを取得できないケースが発生する．そのため，ログの取得漏れを避けつつ，不要なログは取得しないことを目的として ΔT_1，ΔT_2 を設定する．

作成されたデータセットに対するラベル付け

作成されたデータセットに対して調査者が内容を確認し，そのデータセットが実際に調査が必要なアラートであるかを判断してラベル付けを行う．調査をが必要であると判断した場合は「要確認」，調査の結果，運用上問題がないと判断された場合は「問題なし」としてラベル付けする．調査者は，データセットに含まれるチケット作成日時 t を基準とし，$t - \Delta T_1$ から $t + \Delta T_2$ の範囲に記録されたログを確認する．確認するログは，監視システムによる GET リクエストに対応するアクセスログおよび，関連するアプリケーションログである．

過去のデータセットとの比較

新たに作成されたデータセットに対して，過去に保存されたデータセットと新規に作成されたログとメトリクスのデータセットが一致するものを探す．パターンの一致とはアクセスログ一致する過去のデータセットが存在する場合には，そのラベルを参照することで異常の有無を判定する．一致するパターンが存在しない場合は，人手によるラベル付けが必要なため「要確認」のラベルを付ける．

アラート通知

過去のデータセットとのパターンの確認により「要確認」と判断された場合のみ，通知を行う．過去に同様のパターンが「問題なし」として蓄積されている場合には，アラート通知を行わない．

ユースケース・シナリオ

本稿では，ユースケースとして CDSL で運用しているサイトである Doktor を使用する．Doktor は Kubernetes クラスタ上に構築されているサイトである．利用者は web ブラウザを使用してページへアクセスすることで Doktor 上のコンテンツを閲覧することができる．図 6にユースケース・シナリオを示す．

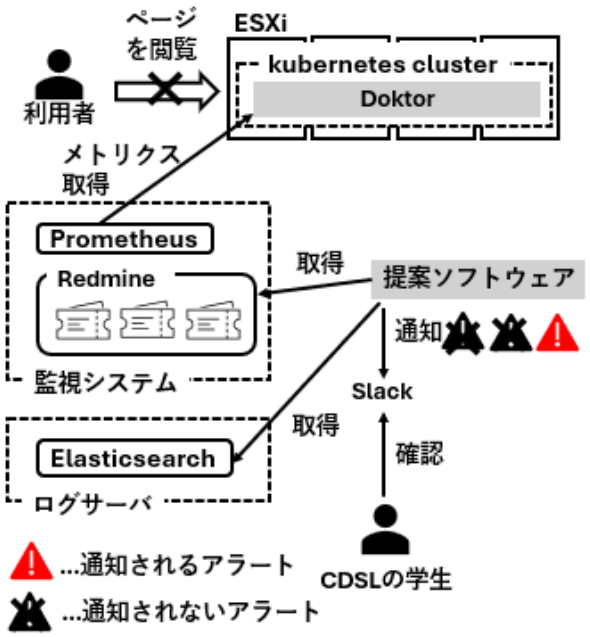

図 6: ユースケース・シナリオ

利用者が Doktor にアクセスし，コンテンツを閲覧しようとしたがアクセスできなかった．同時刻，Doktor を監視している Prometheus は HTTP ステータスコードが 200 以外のメトリクスを取得し，Redmine ではチケットが作成される．提案ソフトウェアはチケットから作成日時とステータスコードを取得し，その時刻を起点として Elasticsearch から前後 1 分間の Blackbox exporter による GET リクエストに対応するアクセスログを取得する．これらをもとにログとメトリクスから 1 つのデータセットを作成し，過去に蓄積されたデータセットとパターンが一致するものを探す．パターンの一致したデータセットのラベルが「問題なし」の場合，そのチケットのアラート通知は行わない．提案適用前は，Doktor を監視している Prometheus は Doktor の HTTP ステータスコードが 200 以外になった場合，全てをアラートとして通知している状態であった．本提案により，既に「問題なし」と判断されているチケットのアラートを抑制できる．

4. 実装

Redmine のチケットから作成日時を取得し，その時刻を起点として監視システムで取得したメトリクスとログサーバで取得したログを 1 つのデータセットとして保存する処理と，保存されたデータセットをもちいてアラート通知の必要性を判定し，Slack に通知する処理について説明する．実装の処理の流れを図 7に示す．

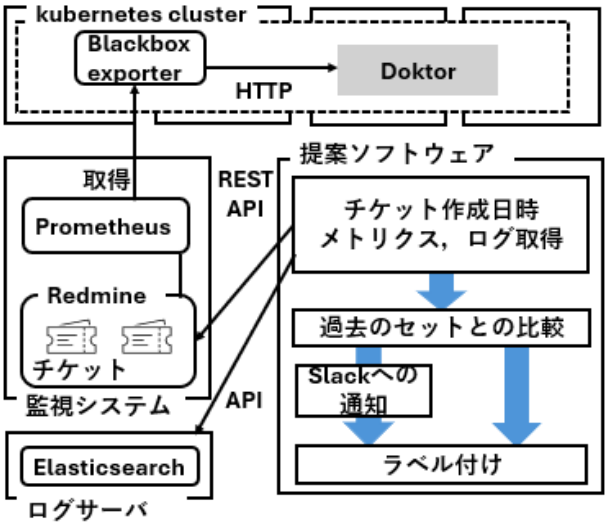

図 7: 実装の処理の流れ

チケット作成日時とメトリクスの取得

チケット作成日時の取得には，Redmine の REST API をもちいる．1 分間隔で API を実行し，新たに作成されたチケットのチケット番号とその作成日時，Prometheus で取得された HTTP ステータスコードのメトリクスを取得する．

ログの取得

ログの取得には Elasticsearch API をもちいる．取得したチケット作成日時から前後 1 分を対象とする時間を設定し，Elasticsearch に対するアクセスログの検索クエリを作成する．検索対象とするログは，Blackbox exporter が監視対象に対して送信した HTTP リクエストに対応するものであり，ノード IP 直下，ノード IP/paper/UUID，ノード IP/author/UUID への GET リクエストの 3 種類のエンドポイントに対するアクセスログを取得する．これにより，1 つのチケットに対して，チケット作成時刻前後 1 分間における 3 種類のエンドポイントへのアクセスログを取得する．

データセットの保存形式

取得したメトリクスとアクセスログは，チケット 1 件ごとに 1 つのデータセットとして保存する．データセットは CSV 形式であり，チケット番号，Prometheus が取得した HTTP ステータスコード，ラベル，Blackbox exporter が監視対象に対して送信したノード IP 直下，ノード IP/paper/UUID，ノード IP/author/UUID への GET リクエストによって出力されたログの HTTP ステータスコードを記録する．

過去のデータセットとの比較

新たに作成されたデータセットに対して，過去のデータセットとの比較を行う．比較では Prometheus が取得した HTTP ステータスコード，Blackbox exporter が監視対象に対して送信したノード IP 直下，ノード IP/paper/UUID，ノード IP/author/UUID への GET リクエストによって出力されたログの HTTP ステータスコードの出力パターンを比較する．過去のデータセットに同じ出力パターンのものがある場合，新しいデータセットに対して過去のデータセットと同じラベルを付ける．過去のデータセットに同じ出力パターンのものがない場合，「要確認」ラベルを付ける．

作成されたデータセットに対するラベル付け

「要確認」のラベル付きのデータセットは調査を行う．調査者は保存されたデータセットから，どのリクエストによって出力されたログの HTTP ステータスコードによってチケットが作成されたのかを確認する．チケットの作成日時とデータセットの内容を使用して Elasticsearch でログの検索を行う．運用上問題がないと判断された場合にはラベルを「問題なし」として，データセットに記録し調査を終了する．

Slack への通知

過去のデータセットとの比較により「要確認」と判断された際，そのチケットを Slack の Incoming Webhook API をもちいて通知する．過去のデータセットに存在しない場合にも通知を行う．

5. 評価実験

評価実験では，監視システムで監視対象の HTTP ステータスステータスコードが 200 以外になり，Redmine でチケットが作成された時間と，監視システムから監視対象へのリクエストを記録したログの時間の差を確認した．また，提案を適用した際に削減できるアラート通知数の確認を行った．

実験環境

実験環境を図 8に示す．主な構成は図 1で示した Doktor と監視システムとログサーバの構成と同じである．

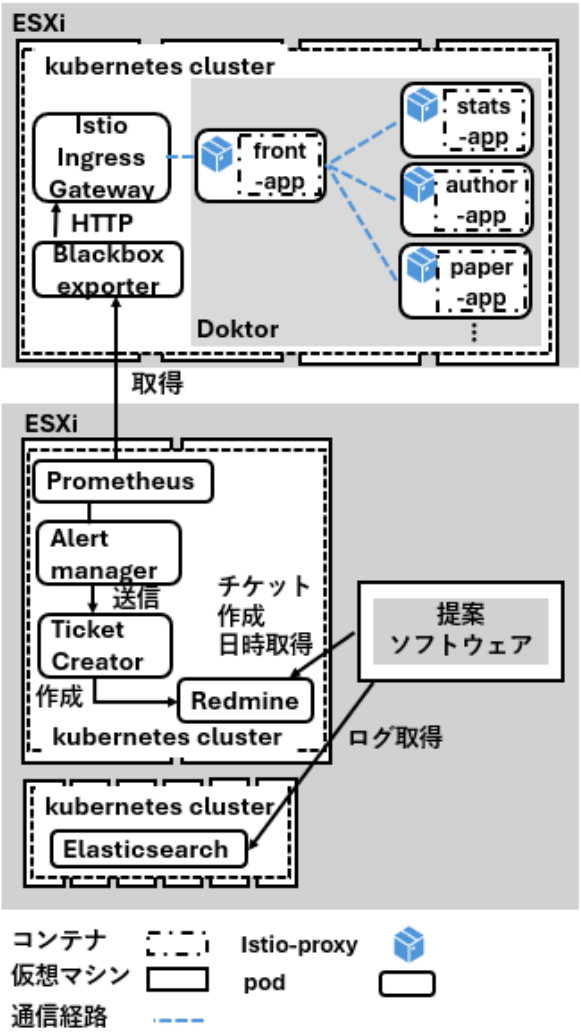

図 8: 実験環境

Doktor は，ESXi 上に構築されたノード群で構成される Kubernetes クラスタ上で稼働している．クラスタは 1 台のマスターノードと 3 台のワーカーノードから構成されており，これらのノードはそれぞれ仮想マシンとして ESXi 上に配置されている．各仮想マシンには Ubuntu22.04 がインストールされており，vCPU 8core，RAM 8GB，SSD 40GB である．また，分散ストレージ用の SSD が 40GB ずつ割り当てられている．Doktor が配置されているクラスタ上には，front，author，paper，stats をはじめとする複数のサービスが稼働している．各サービスの Pod はアプリケーションコンテナとサイドカーコンテナである Istio-proxy から構成されており，Istio Ingress Gateway を通じて外部からのアクセスを受け付ける構成である．

監視システムは，Doktor とは異なる ESXi 上に構築された仮想マシン群で構成される Kubernetes クラスタ上で稼働している．クラスタは 1 台のマスターノードと 1

台のワーカーノードから構成されており，これらのノードはそれぞれ仮想マシンとして ESXi 上に配置されている．各仮想マシンには Ubuntu24.04.2 がインストールされており，vCPU 4core，RAM 8GB，SSD 40GB である．監視には Prometheus と Blackbox exporter を使用する．Prometheus は Exporter を介して監視対象のメトリクスを収集する構成であり，Blackbox exporter は Doktor に対して HTTP リクエストを送信して応答を確認する．返された HTTP ステータスコードをもとに監視を行っている．

提案ソフトウェアは，監視システムと同じ ESXi 上に構築された仮想マシン上にある．仮想マシンには Ubuntu24.04.3 がインストールされており，vCPU 2core，RAM 4GB，SSD 25GB である．Redmine の API を使用してチケット作成日時を取得する．

実験結果と分析

本提案では，監視システムで監視対象の HTTP ステータスコードが 200 以外が確認された場合に，その前後の時間における監視システムから監視対象へのリクエストを記録したログを取得する．まず，ログを取得する際の時間幅 $\Delta T_1, \Delta T_2$ を決定するために，監視システムで監視対象の HTTP ステータスステータスコードが 200 以外になり，Redmine でチケットが作成された時間と，監視システムから監視対象へのリクエストを記録したログの時間の差を確認した．提案では，チケット作成日時を t とし，t を中心とした前後の時間幅をそれぞれΔ T1，Δ T2 と定義している．ログを取得する際の時間幅は短いほど不要なログが減り，望ましいが，短く設定した場合には取得すべきログを取得できないケースが発生する．そのため，実験で確認した時間の差の最大値をそれぞれΔ T1，Δ T2 とする．

実験には，2025 年 11 月 23 日において作成されたチケット 12 件を使用した．まず，各チケットの作成時刻を取得した．次に，Elasticsearch に保存されている，チケット作成時刻の前後に記録された Blackbox exporter から Istio-proxy へのリクエストログのタイムスタンプを取得した．基礎実験では，ログ内に記録されているリクエスト発生時刻ではなく，Elasticsearch に登録された時刻であるタイムスタンプをもちいた．これは，提案ソフトウェアが Elasticsearch を API で検索する際，タイムスタンプに記録されている時刻をもとにログを取得するためである．時間の差は，ログのタイムスタンプからチケットの作成時刻を引いたものとする．チケット番号とその作成時刻，対応するログのタイムスタンプと時間の差を，表 1に示す．

表 1は，チケット番号とそのチケットが作成された時刻，チケット作成時刻の前後に記録された Blackbox exporter から Istio-proxy へのリクエストログのタイムスタンプと，その時刻の差を示している．チケットの作成時刻とログのタイムスタンプの時間の差の最小値は 3 秒，最大値は 11 秒だった．ログのタイムスタンプは，チケット作成時刻よりも早い時間に記録されたものはなかった．そのため，この実験結果から，$\Delta T_1 = 0, \Delta T_2 = 11$ と設定するのが妥当であると考えられる．

表 1: チケット番号とその作成時刻，対応するログのタイムスタンプと時間の差

チケット番号	チケット作成時刻	ログのタイムスタンプ	時間の差
15834	01:39:00	01:39:05	00:05
15909	02:54:00	02:54:10	00:10
16058	05:44:00	05:44:11	00:11
16194	08:31:00	08:31:03	00:03
16329	11:06:00	11:06:10	00:10
16418	12:37:00	12:37:09	00:09
16703	17:31:00	17:31:09	00:09
16836	19:55:00	19:55:07	00:07
16849	20:04:00	20:04:11	00:11
16885	20:34:00	20:34:05	00:05
16978	22:14:00	22:14:09	00:09
17073	23:51:00	23:51:07	00:07

次に，提案手法の適用前後のアラート通知数を比較することで，提案手法によってアラート通知が削減されるかを確認した．実験には，2025 年 11 月 24 日から 2025 年 11 月 26 日において作成されたチケット 55 件を使用した．このうち，2025 年 11 月 24 日に作成された 25 件のチケットに対応するデータセットを，提案手法が使用する過去のデータセット群として使用するために，CDSL に所属している学生 4 名が調査者としてラベル付けを行った．調査者は，有田 海斗，岡田 京太郎，佐藤 健斗，月森 陽太である．ラベル付けの結果，全員が「問題なし」のラベルを付けたデータセットは 25 件中 7 件で，全員が「要調査」のラベルを付けたデータセットはなかった．残りの 18 件については調査者によって判断にばらつきがあった．調査者の判断が一致しなかったデータセットに対するラベル付けの結果を表 2に示す．

表 2は調査者の判断が一致しなかったデータセットに対するラベル付けの結果を示している．各行は 1 つのデータセットを表しており，そのデータセットと対応するチケット番号と，4 人の調査者によるラベル付けの結果を示している．4 人中 1 人が「要確認」と判断したデータセットはチケット番号 18725，18845，18896，19164，19378，19394 の 6 件だった．チケット番号 18725，18845，18896，19378 は佐藤 健斗のみが「要確認」と判断していて，チケット番号 19164，19394 は岡田 京太郎のみが「要確認」と判断していた．4 人中 2 人が「要確認」と判断したデータセットはチケット番号 18633，18770，19147，19275，19392 の 5

表 2: 調査者の判断が一致しなかったデータセットに対するラベル付けの結果

チケット番号	有田 海斗	岡田 京太郎	佐藤 健斗	月森 陽太
18633	問題なし	要確認	要確認	問題なし
18725	問題なし	問題なし	要確認	問題なし
18770	問題なし	要確認	要確認	問題なし
18832	問題なし	要確認	要確認	要確認
18845	問題なし	問題なし	要確認	問題なし
18896	問題なし	問題なし	要確認	問題なし
18915	問題なし	要確認	要確認	要確認
19147	問題なし	問題なし	要確認	要確認
19164	問題なし	要確認	問題なし	問題なし
19239	問題なし	要確認	要確認	要確認
19275	問題なし	問題なし	要確認	要確認
19353	問題なし	要確認	要確認	要確認
19364	問題なし	要確認	要確認	要確認
19378	問題なし	問題なし	要確認	問題なし
19380	問題なし	要確認	要確認	要確認
19386	問題なし	要確認	要確認	要確認
19392	問題なし	問題なし	要確認	要確認
19394	問題なし	要確認	問題なし	問題なし

件だった．チケット番号 18633，18770 は岡田 京太郎と佐藤 健斗の 2 人が「要確認」と判断していて，チケット番号 19147，19275，19392 は佐藤 健斗と月森 陽太の 2 人が「要確認」と判断していた．4 人中 3 人が「要確認」と判断したデータセットはチケット番号 18832，18915，19239，19353，19364，19380，19386 の 7 件だった．岡田 京太郎と佐藤 健斗と月森 陽太の 3 人がこれらの 7 件に対して「要確認」と判断していた．

2025 年 11 月 24 日に作成された 25 件のチケットに対応するデータセットについて，1 人以上の調査者が「要確認」と判断した場合は要確認」，全員が「問題なし」と判断した場合のみ「問題なし」とラベル付けを行った．これら 25 件のデータセットを提案手法が使用する過去のデータセットとした．2025 年 11 月 25 日から 2025 年 11 月 26 日に作成された 30 件のチケットについて，対応するデータセットを生成し，過去のデータセットとの一致を確認することで通知対象となるチケットの件数を比較した．提案適用前と提案適用後における通知対象となるチケットの件数の比較を図 9に示す．

提案適用前は 30 件のチケット全てが通知の対象であったが，提案の適用により通知の対象が 24 件となって 6 件の通知を削減でき，削減率は 20 %であることが確認できた．これは過去の 25 件のデータセットのうち「問題なし」と判断されたデータセットと一致するデータセットが評価対象の 30 件の中に 6 件存在したためである．

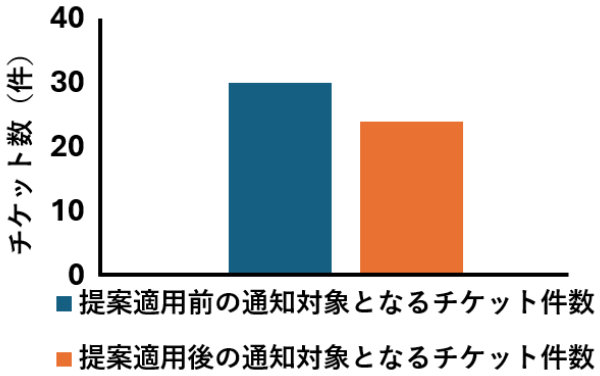

図 9: 提案適用前と提案適用後における通知対象となるチケットの件数の比較

6. 議論

本提案の評価では，どのアラートが実際に通知が必要であるかを定義していないため，提案手法によるアラート通知削減の適合率を確認することができていない．適合率を求めることで，提案手法が削減したアラート通知が実際に通知が不要なアラートであったかを評価できる．これに対し，アラートが送信されチケットが作成された後，提案手法によりデータセットを作成した上で，一定時間にわたり，Prometheus に記録された HTTP ステータスコードのメトリクスの値，Elasticsearch に記録された Blackbox exporter による HTTP リクエストのログのステータスコード，対応する pod とノードのステータスを確認する．これらの値においてアラートが送信された際の状態が継続している場合に，実際に通知が必要であるアラートと定義する．具体的には，Prometheus に記録される HTTP ステータスコードが 200 以外の値を継続して取得している状態が挙げられる．一方，継続していない場合は通知が不要なアラートとみなす．これは，後続の Blackbox exporter による Doktor への HTTP リクエストで正常な応答が得られており，Doktor が通常どおりアクセスでき状態であることを意味しているためである．このように，通知が必要なアラートと不要なアラートの基準を定義することで提案手法の適合率を評価することができるようになる．

本提案方式では，アラートを示すチケットが作成された際に，監視対象におけるログとメトリクスの対応関係のパターンをデータセットとして記録する．CDSL の学生が「問題なし」と判断しラベル付けを行ったデータセットのログとメトリクスの対応関係のパターンと一致する場合にはアラートは抑制し，アラートの通知数を削減できる．しかし，アラート送信時の監視対象におけるログとメトリクスのパターンは非常に多様である．そのため，初めて記録されたログとメトリクスの対応関係のパターンをもつデータセットが必ず通知対象となり，人手によるラベル付けが

必要となる本手法は，実運用では大きな負担となるという課題がある．この課題を解決するために，初めて記録されたログとメトリクスの対応関係のパターンをもつデータセットに対しても「問題なし」と判断できる仕組みを検討する．具体的には失敗したリクエストだけでなく，直後の成功したリクエストについて，ログ，メトリクス，ノードの状態，Pod の状態を確認し，パターンとして記録する．失敗パターンと，失敗パターンの発生直後に取得された成功パターンを比較し，失敗パターンの時に見られた異常が直後に解消しているかを確認することで，失敗が一時的なものであるかの判断を行う．これにより，当該チケットに対応するデータセットに対して「問題なし」のラベル付けを行うことができ，人手によるラベル付けの負担を減らすことができる．

本提案の実験環境では，実際に Doktor から出力されるログを参考に Logstash のフィルタ設定を記述し，Elasticsearch がログから HTTP ステータスコードを抽出できるようにすることで，API による HTTP ステータスコーの取得を行っている．しかし，参考にしたログ構造に当てはまらない構造のログについては，ログからステータスコードを抽出できない．ログの構造化において，全てのログ構造を網羅し完全に構造化することは現実的に難しい [18–20]．Logstash のフィルタ設定において，構造化に失敗したログにその旨を示すタグを付与することで，構造化されなかったログを管理者が Elasticsearch で検索できるようにし，フィルタ設定の更新を支援することで改善することができる．

7. おわりに

CDSL では，インターネットに公開しているサイトである Doktor を，Prometheus で監視し，ログを Elasticsearch に保存している．Prometheus は Doktor の HTTP ステータスコードのメトリクスを取得し，その値が 200 以外になった場合にアラートを発行する．Alertmanager は Prometheus で発行されたアラートを Redmine に送信し，Redmine でチケットが作成される．課題は，Alertmanager から Redmine に送信されるアラートの中に，Prometheus の誤検知が含まれていることである．提案では Redmine にチケットが作成された際，Prometheus が取得した対象の HTTP ステータスコードのメトリクスの値と，その時刻の前後に Elasticsearch に記録された HTTP リクエストのログのステータスコードを 1 つのデータセットとして保存する．調査者はチケットの作成時刻と，その前後の時間に記録されたログを確認し，データセットに対して「要確認」か「問題なし」かのラベル付けを行う．チケット作成時には，新たに作成されたデータセットに含まれる Prometheus が取得した HTTP ステータスコードのメトリクス値と，Elasticsearch に記録された HTTP リクエストのログのステータスコードを過去のデータセットに含まれる同じ値の組み合わせと比較する．一致する場合には，過去のデータセットに付けられた「要確認」か「問題なし」のラベルを新しいセットに付ける．一致しない場合は「要確認」のラベルをつけアラート通知を行う．「問題なし」の場合にはアラート通知は行わない．実験では，提案手法の適用により削減できるアラート通知の数を評価する．2025 年 11 月 24 日から 11 月 26 日において作成された 55 件のチケットに対応するデータセットを実験の対象とする．そのうち 11 月 24 日に作成された 25 件のデータセットについては調査者がログを確認しラベル付けを行い，提案手法が使用する過去の複数のデータセットとした．調査者は CDSL に所属する学生 4 人である．ラベル付けの結果，全員が「問題なし」と判断したデータセットは 25 件中 7 件で，全員が「要確認」と判断したデータセットはなかった．また，4 人中 1 人が「要確認」と判断したデータセットが 6 件，2 人が「要確認」と判断したデータセットが 5 件，3 人が「要確認」と判断したデータセットが 7 件だった．全員が「問題なし」と判断した過去のデータセットのみを「問題なし」のデータセットとして使用し，11 月 25 日から 11 月 26 日に作成された 30 件のデータセットを評価対象として提案手法を適用した．その結果，30 件中 6 件が「問題なし」とラベル付けされた過去のデータセットと一致したため，アラート通知の対象外となることが確認でき，残りの 24 件は「要確認」となりアラート通知の対象となることが確認できた．

参考文献

[1] Burton, S., McDermid, J., Garnett, P. and Weaver, R.: Safer Complex Systems : An Initial Framework (2021).

[2] Rivera, C. and Martinez, A.: Enhancing Reliability Through Effective System Monitoring, *Science and Technology*, Vol. 8 (2024).

[3] Cao, Q., Qiao, Y. and Lyu, Z.: Machine learning to detect anomalies in web log analysis, *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, pp. 519–523 (online), DOI: 10.1109/CompComm.2017.8322600 (2017).

[4] Cândido, J. B., Aniche, M. F. and van Deursen, A.: Log-based software monitoring: a systematic mapping study (2021).

[5] Teixeira, C., de Vasconcelos, J. B. and Pestana, G.: A knowledge management system for analysis of organisational log files, *2018 13th Iberian Conference on Information Systems and Technologies (CISTI)*, pp. 1–4 (online), DOI: 10.23919/CISTI.2018.8399229 (2018).

[6] Jani, Y.: Unified Monitoring for Microservices: Implementing Prometheus and Grafana for Scalable Solutions, `https://www.theseus.fi/handle/10024/512860` (2022). Bachelor's thesis.

[7] Son, S. J. and Kwon, Y.: Performance of ELK stack and commercial system in security log analysis, *2017 IEEE 13th Malaysia International Conference on Communications (MICC)*, pp. 187–190 (online), DOI: 10.1109/MICC.2017.8311756 (2017).

[8] Lertwuthikarn, T., Barroso, V. C. and Akkarajitsakul,

K.: Resource Optimization for Log Shipper and Preprocessing Pipeline in a Large-Scale Logging System, *2022 IEEE 5th International Conference on Knowledge Innovation and Invention (ICKII)*, pp. 196–200 (online), DOI: 10.1109/ICKII55100.2022.9983590 (2022).

[9] Langi, P. P. I., Widyawan, Najib, W. and Aji, T. B.: An evaluation of Twitter river and Logstash performances as elasticsearch inputs for social media analysis of Twitter, *2015 International Conference on Information Communication Technology and Systems (ICTS)*, pp. 181–186 (online), DOI: 10.1109/ICTS.2015.7379895 (2015).

[10] Wei, B., Dai, J., Deng, L. and Huang, H.: An Optimization Method for Elasticsearch Index Shard Number, *2020 16th International Conference on Computational Intelligence and Security (CIS)*, pp. 191–195 (online), DOI: 10.1109/CIS52066.2020.00048 (2020).

[11] Ghafouri, A., Abbas, W., Laszka, A., Vorobeychik, Y. and Koutsoukos, X.: Optimal Thresholds for Anomaly-Based Intrusion Detection in Dynamical Environments (2017).

[12] Zohrevand, Z. and Glässer, U.: Should I Raise The Red Flag? A comprehensive survey of anomaly scoring methods toward mitigating false alarms (2020).

[13] Correia, L., Goos, J.-C., Klein, P., Bäck, T. and Kononova, A. V.: Online model-based anomaly detection in multivariate time series: Taxonomy, survey, research challenges and future directions, *Engineering Applications of Artificial Intelligence*, Vol. 138, p. 109323 (online), DOI: https://doi.org/10.1016/j.engappai.2024.109323 (2024).

[14] Venkateswaran, P., Malapati, A., Natu, M. and Sadaphal, V.: Towards next-generation alert management of data centers, *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, pp. 1–2 (online), DOI: 10.1109/COMSNETS.2016.7440016 (2016).

[15] Cinque, M., Corte, R. D. and Pecchia, A.: Entropy-Based Security Analytics: Measurements from a Critical Information System, *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 379–390 (online), DOI: 10.1109/DSN.2017.39 (2017).

[16] Viola, L., Ronchieri, E. and Cavallaro, C.: Combining Log Files and Monitoring Data to Detect Anomaly Patterns in a Data Center, *Computers*, Vol. 11, No. 8 (online), available from ⟨https://www.mdpi.com/2073-431X/11/8/117⟩ (2022).

[17] Perdices, D., García-Dorado, J. L., Ramos, J., De Pool, R. and Aracil, J.: Towards the Automatic and Schedule-Aware Alerting of Internetwork Time Series, *IEEE Access*, Vol. 9, pp. 61346–61358 (online), DOI: 10.1109/ACCESS.2021.3073598 (2021).

[18] Khan, Z. A., Shin, D., Bianculli, D. and Briand, L. C.: Impact of log parsing on deep learning-based anomaly detection, *Empirical Software Engineering*, Vol. 29, No. 6 (online), DOI: 10.1007/s10664-024-10533-w (2024).

[19] Fu, Y., Yan, M., Xu, Z., Xia, X., Zhang, X. and Yang, D.: An empirical study of the impact of log parsers on the performance of log-based anomaly detection, *Empirical Softw. Engg.*, Vol. 28, No. 1 (online), DOI: 10.1007/s10664-022-10214-6 (2022).

[20] Beck, V., Landauer, M., Wurzenberger, M., Skopik, F. and Rauber, A.: System Log Parsing with Large Language Models: A Review (2025).

