障害時のアラート名とホスト名の重複判定を用いたグルーピングによるチケット件数の削減

山崎 拓海[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研究室である, Cloud and Distributed Systems Laboratory（CDSL）では，研究室内で運用する物理サーバや，ネットワーク機器が稼働しているか確認するため監視を行っている．監視システムは設定されたメトリクスの値が閾値を超えた場合にアラートを通知する．通知されたアラートはチケットシステムに登録される．課題は，通知されたアラート 1 件 1 件を全てチケットとして登録しているため，次に同じアラートが通知された際に，重複してチケットが登録されてしまうことである．提案では，通知されたアラートからアラート名と，アラート内の監視対象の IP アドレスまたはホスト名を表す文字列である instance を抽出し，すでに登録されているチケット内に 同じアラート名または同じ instance が含まれているかを判定してグルーピング処理を行う．同じアラート名で異なる instance を含むアラートが通知された場合は，そのアラート名の共通の親チケットを作成し，その子チケットとして登録する．同じアラート名と instance のアラートが通知された場合は最初に作成されたチケットにコメントという形で追加する．評価では，2025 年 10 月 21 日から 11 月 1 日の間で，全てのアラートをチケットとして登録している場合と提案ソフトウェアを適用した場合のチケット件数を比較した．提案手法は，最後に通知されてから 4 時間を区切りとし，それ以降に通知されたアラートは新たに親チケットとして登録する．チケット件数を比較した結果，通知されたアラートを全てチケットとして登録する場合は 342 件となり，提案ソフトウェアでは，135 件となった．提案ソフトウェアは通知された全てのアラートをチケットとして登録する場合と比較して約 61%チケット件数を削減した．また，提案手法で削減された 135 件のうち，アラート名が異なる場合かつ instance が異なる場合でチケットをグルーピングしていることはなかった．一方で，135 件のうち，同じアラート名かつ同じ instance だが，直近で 4 時間以内にそのアラートが通知されていなかったため，まとめきれなかったチケットが 62 件存在した．

1. はじめに

背景

EC サイトシステムは，24 時間 365 日，絶え間なく安定して稼働し続けることが求められており，その運用には高度な監視体制が不可欠となっている．これを実現するために，サーバやネットワーク機器，各種アプリケーションの状態を常時監視し，異常を早期に検知と対応するために，監視システムと連携したアラート通知機能を導入している [1].

アラートはチケットに記録や追跡できる形式に変換されることで，運用対応の可視化と体系化を実現している．この運用方法を採用することで，運用担当者が誰でも対応履歴を参照できるようにし，属人化の回避，対応漏れの防止，障害対応のスピード向上の効果をもたらしてくれる [2,3]. さらに，アラートチケットはシステム障害のパターン分析や，インシデントの傾向把握にも活用できる．特に，同じホストや同じサービスに関して繰り返し発生する問題を可視化し，根本原因の分析につなげるための基礎データとしても機能する [4]．運用現場においては，発生したアラートを単発で処理するのではなく，チケットとして管理し体系的に扱うことが，長期的な安定運用の鍵となる．特に，大規模なシステム環境ではアラートの頻度が高く，1 つの障害から多数のアラートが同時に発生することもある．体系的に管理することで，対応の重複や漏れを防ぎ，根本原因に対する集中的な対応を実現する．また，インシデント対応の標準化した基盤でも，アラートチケットは運用の信頼性と効率性を支える基盤要素となる [5,6].

東京工科大学コンピュータサイエンス学部の研究室である，Cloud and Distributed Systems Laboratory（以下，CDSL）では，物理マシンが合計で 10 台稼働している．物

[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
〒 192–0982 東京都八王子市片倉町 1404-1

理マシンには，VMware が提供しているハイパーバイザーである VMware ESXi を導入している．VMware ESXi は，物理マシンにインストールして仮想マシンを実行できるソフトウェアで，1 つの物理マシン上で複数の仮想マシンを運用することができる．VMware ESXi により，他の仮想マシンを実行するためのソフトウェアに比べて少ないリソースで運用でき，物理マシンのリソースを効率的に使用できる．CDSL 内では，所属している学生が実験に使用するマシンが 7 台動いている．また，DNS や DHCP，STNS，踏み台サーバを管理している基幹用マシンが 2 台，外部公開サイト用のマシンが 1 台ある．その他に NAS が 2 台稼働している．図 1 は CDSL 内の監視システムの構成図を示す．

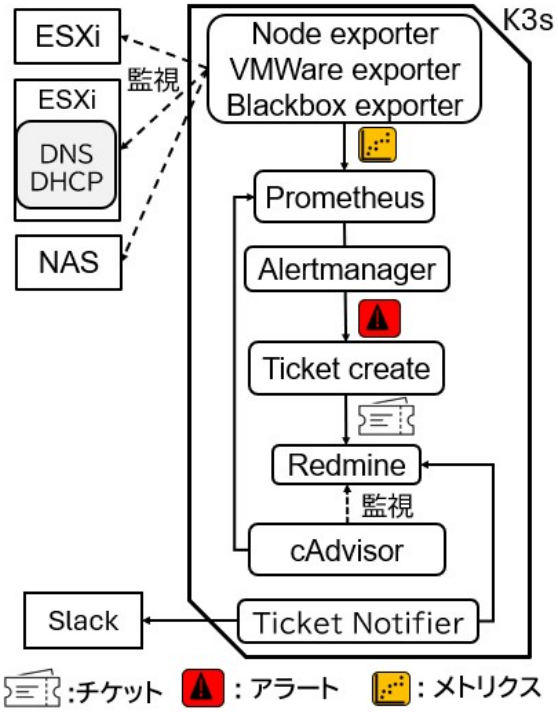

図 1: CDSL 内の監視システム構成

ESXi は複数の OS を実行可能にするハイパーバイザーで，各物理マシンに導入されている．CDSL の研究室の学生が実験で使用する開発環境用に 7 台，外部にサイトを公開し運用する本番環境用が 1 台，DNS．DHCP を動かす基幹用のサーバが 2 台配置されている．基幹サーバ上で動作する DNS は，DHCP が登録した IP アドレスとホスト名を紐付ける．DHCP は，MAC アドレスと IP アドレスの紐付けを行う．NAS は各システムや VM のデータのバックアップを保存している．Prometheus オープンソースのシステム監視ソフトウェアで [7]，各 exporter が収集したデータは Prometheus が取得し，Grafana でデータを可視化をしている．[8,9]．Node exporter は，Linux システムのハードウェアおよび OS レベルのメトリクス（CPU，メモリ，ディスク I/O）を収集している．Blackbox exporter は ICMP，SSH，HTTP のプロトコルで監視対象に疎通ができるかを確認する．VMware exporter は，VMware ESXi 上の仮想マシンのパフォーマンスメトリクス（CPU 使用率，メモリ使用量，ディスク I/O）を収集する．cAdvisor は Google が開発したオープンソースのコンテナ監視ツールである．実行中のコンテナのリソース使用状況やパフォーマンス特性を収集している．Alertmanager は，Prometheus に付随するアラート管理ツールで，Prometheus からのアラートを受信し，通知するツールである．Ticket create は，Alertmanager から通知されたアラートをもとにチケットを作成する．Redmine は，オープンソースのプロジェクト管理および課題追跡ツールで，Ticket create で作成されたチケットを登録している．Ticket Notifier は Redmine にチケットが登録されると，Slack に登録されたことをポストする．

CDSL では，Alertmanager からアラートが通知されチケットが作成されると，原因調査や対応のためにその時間の監視担当者が定められた手順と Runbook に沿って調査を実施する．原因の特定や障害が解消されなかった場合はエスカレーションし，2 次チケット，3 次チケットまで作成される仕組みをとっている．2 次調査では，対象となるシステムの管理を担当しているメンバーが調査することになっている．

課題

課題は，CDSL で運用されている監視システムにおいて，Alertmanager からアラートが通知された際に，全てのアラートをが親のチケットとして登録されてしまっていることである．図 2 は，同じアラート名と監視対象のチケットが重複して作成されている様子を示す．Alertmanager はアラートが発生したことを通知する．4 つのアラートは短時間で同じアラートが発生していることを表している．Ticket create はアラートをもとにチケットを作成する．Redmine は Ticket create で作成されたチケットを登録する．Redmine 内のチケットは同じアラート名と監視対象を示すアラートから登録されたものである．

Alertmanager から短時間（数分間隔）で同じアラートが通知されても，Ticket Create では全ての通知をチケット化するため，Redmine には同じ原因と思われるアラートがいくつも並ぶことになる．そのため，対応しなければいけないチケットの見直しや，チケットを監視作業者がまとめる作業が発生する．

表 1 は，2025 年の 10 月 21 日，22 日の 2 日間に Redmine に登録されたチケットの一部分を抜粋してきたものである．発生したアラートは「Internal Host CPU UsageHigh-Usage100%」と「External_Clematis_Node_ICMP_Check」の 2 種類のみである．External Clematis Node ICMP Check

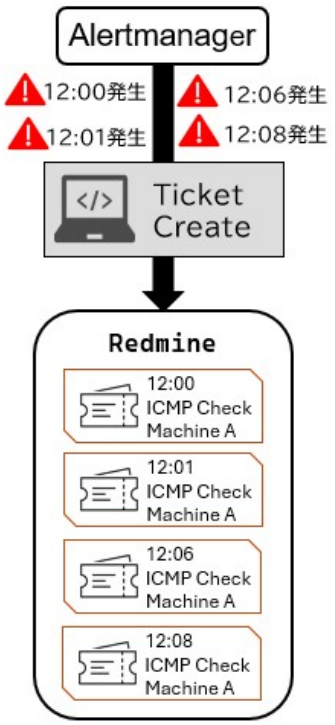

図 2: 同じアラート名と監視対象のチケットが重複して作成される

は CDSL の本番環境上で動作する物理環境上にある，4 つの Node に対して ICMP を用いた疎通確認ができなくなったことを示すアラートである．Internal Host CPU UsageHigh-Usage100%については，CDSL の開発環境で動作する基幹用の物理サーバのうち 1 台の CPU 使用率が 100%を超えた際にアラートが通知される．その原因は STNS サーバを動かすコンテナ内の shell スクリプトが 1 秒ごとに git config を呼び出していることである．そのため，12 件のアラートについては，同じ原因によるものであるといえる．以上のことから，同じ障害が原因でアラートが通知された場合，チケットとして登録されてしまう．このように同じチケットが重複して登録されると．チケットをまとめる時間や障害の原因を示すチケットを見逃すことにつながる．

図 3 は重複した同じチケットがアラートから登録され，1 次調査を違う対応者が担当してしまっている例を示す．Alertmanager は Prometheus 上で監視ルールが閾値を超えた際にアラートを通知する．1 次チケットは Alertmanager からの通知で登録されたチケットである．1 次対応者は，その時間の監視業務を担当している人物である．2 次チケットは 1 次対応者が，調査を行いエスカレーションするために作成するチケットである．2 次対応者はチケットのアラート発生元のシステムの管理を担当している人物である．アラートの，調査，対処を行う際に，同じアラートのチケットが登録されていると，同じ調査を違う担当者が別々に行ってしまう．それによって監視運用の効率の低下を招く．

表 1: アラートチケット抜粋

アラート名	作成日
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-23 12:12
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-23 12:08
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-23 07:12
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-22 18:08
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-22 16:48
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-22 15:44
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-22 15:15
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-22 15:12
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-22 14:28
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-22 14:18
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-22 13:54
Internal Host CPU UsageHigh-Usage100% (monitoring-master-ml:32708)	10-22 13:38
External_Clematis_Node_ICMP_Check (192.168.201.8)	10-21 17:11
External_Clematis_Node_ICMP_Check (192.168.201.10)	10-21 17:11
External_Clematis_Node_ICMP_Check (192.168.201.9)	10-21 17:11
External_Clematis_Node_ICMP_Check (192.168.201.11)	10-21 17:11
External_Clematis_Node_ICMP_Check (192.168.201.10)	10-21 16:47
External_Clematis_Node_ICMP_Check (192.168.201.8)	10-21 16:47
External_Clematis_Node_ICMP_Check (192.168.201.11)	10-21 16:47
External_Clematis_Node_ICMP_Check (192.168.201.9)	10-21 16:47
External_Clematis_Node_ICMP_Check (192.168.201.8)	10-21 16:41
External_Clematis_Node_ICMP_Check (192.168.201.10)	10-21 16:41
External_Clematis_Node_ICMP_Check (192.168.201.11)	10-21 16:41
External_Clematis_Node_ICMP_Check (192.168.201.9)	10-21 16:41

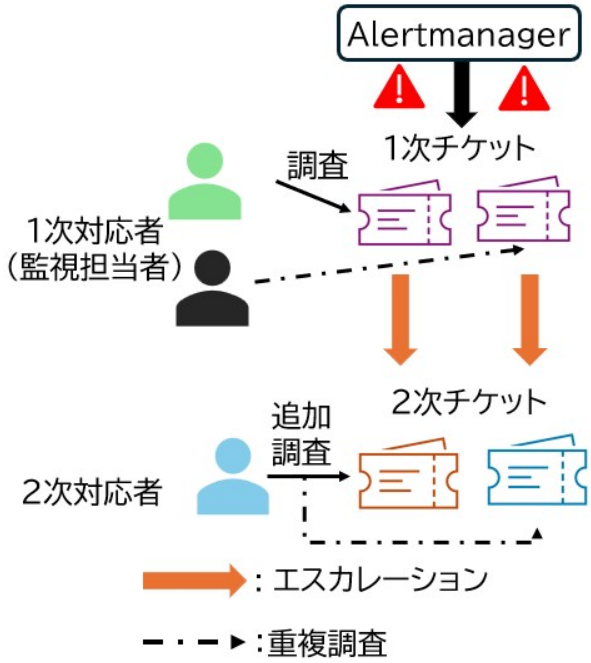

図 3: 重複した同じチケットがアラートから登録され，1 次調査を違う対応者が担当してしまっている例

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 章の提案では，課題を解決する提案とユースケース・シナリオについて述べる．第 4 章の実装では，提案をもとに作成したソフトウェアの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，実験結果と分析について述べる．第 6 章の議論では，提案の議論を述べる．第 7 章では，全体のまとめを述べる．

2. 関連研究

企業の大規模 IT インフラを対象に，アラートおよびインシデントチケットを自動的にクラスタリングする手法を提案している論文がある [10]．数百万件規模のアラートデータを対象に，テキスト情報をもとにした階層的およびグラフベースのクラスタリングを行い，同種の障害や再発パターンを抽出する．クラスタ化することで，頻発するアラート群を一つの問題単位として扱うことが可能になり，運用者が重複チケットに個別対応する負担を削減している．同手法では主に文面上の類似性に依存しており，異なる表現ながら同じ原因に由来するアラートの識別には限界があり，根本原因を特定するための因果関係分析やシステム依存性の考慮は行われていない．

Microsoft Azure におけるチケットの重複統合を目的とし，アラート情報とチケット情報を多層リンク構造（alert–alert および alert–ticket）で結合する手法を提案している研究がある [11]．この研究では，1 つのインシデントに起因する複数のチケットを，テキスト表現が異なる場合でも統一的にグルーピングできる点に特徴がある．さらに，Attention 機構を用いたニューラルネットワークモデルにより，各チケットを適切なインシデントグループに分類し，重複チケットを高精度（F1 スコア 0.87〜0.93）での検出を可能にしている．iPACK と呼ばれているこの手法では，クラウドサービス事業者の運用データを利用することで，実運用環境に即した高精度なチケット統合を実現している．その反面，クラウドプラットフォーム固有のインシデントログに依存しており，オンプレミス環境や，Microsoft Azure 以外のクラウドプラットフォームを使用した際のログ構造の異なるシステムへの汎用性は限定的となっている．

大規模言語モデル（LLM）を活用し，アラートの内容と発生パターンの因果関係を解析する手法を提案している論文がある [12]．この手法では，まず時系列および空間的近接性にもとづいてアラートをクラスタリングし，その後，LLM によってアラート間の意味的，因果的関係を推論する二段階構成をとっている．これにより，単なる類似性ではなく，根本原因にもとづいてアラートを統合することが可能となり，短時間に大量のアラートが連続して発生するアラートストームが発生した際にも，関連する通知を 1 つのインシデント単位に要約できるとしている．実験結果では，既存の頻出パターン抽出法や密度クラスタリング法に比べて F1 スコアが大幅に向上しており，従来のルールベース手法を凌駕する性能を示している．一方で，LLM を活用するためには高い計算コストが伴い，リアルタイム性の確保や，未知の障害パターンに対する汎化性能の検証が今後の課題となっている．

3. 提案

本稿での提案は，監視ソフトウェアから通知を受信すると通知内容を解析し，条件に合わせてチケット管理ソフトウェアに登録，更新する．チケット間の関連性を自動判定し，親チケットと子チケットを用いて階層構造化することで，アラート原因単位での管理を目指す．提案では下記の 3 点を柱としている．

（1） 同じアラート名の重複チケット発行の抑制

（2） 異なるホストから発生した同じアラートの統合管理

（3） 対応状況に応じた親子チケットの階層構造化

図 4 は提案ソフトウェアの処理の流れを示している．システム監視ソフトは，各システムのメトリクスを集めアラートを通知する．チケットクリエイトはアラートを受信し，グルーピングの判定を行いアラートを処理する．チケット管理ソフトは，チケットクリエイトで作成されたチケットを登録，管理する．システム監視ソフトウェアから通知されたアラートをチケットクリエイトが受信すると，チケット管理ソフトに対して，登録済みのチケットについ

て検索をかけ，登録状況を取得する．それをもとにグルーピング判定を行い，チケットを作成または，既存のチケットへのコメントとして，アラートを処理する．

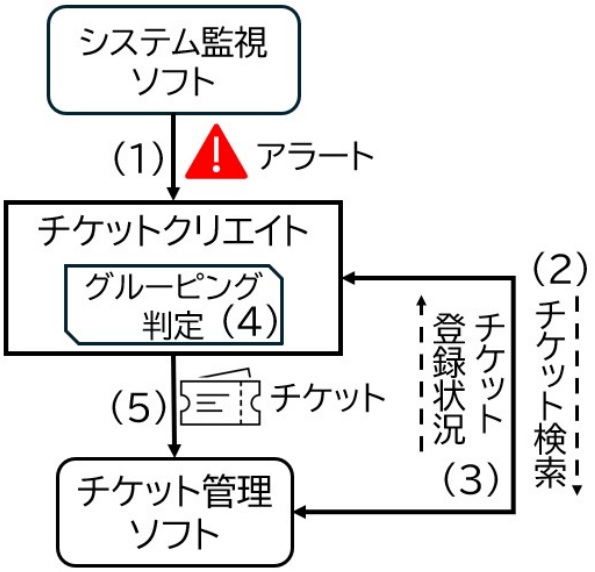

図 4: 提案ソフト概要

本提案システムは，アラート内容の完全な因果分析は行わず，短期間に 1 つのアラートが同じまたは複数ホストから通知された場合は，同じ原因によるものと推定．構造的にグルーピングするヒューリスティック [13] 方式を採用する．これにより，「短時間で同種のアラートが集中する現象」を同じ事象として扱う．

同じアラート名のルート化

図 5 は同じアラートで異なるホストの場合のチケットの構造を示す．Root ticket は，同じアラートで異なる複数のホストからアラートが発生すると，集約先となる親チケットである．子チケットには，同じアラートだが，ホストが異なるアラートのチケットである．同じのアラートが異なるホストからアラートが通知された場合は，共通の親チケット（Root チケット）を作成して集約する．これにより，「システム全体の障害」や「ネットワーク共通問題」の複数ホストにまたがる現象を一括で管理できる．また，同じホストからのアラートについては，前項と同じようにコメントに追記していくことで，発生した事実を残す．

同じアラート名かつホストも同じ場合

図 6 は同じアラート名，なおかつ対象ホストも同じアラートが再発した際に，コメント追記で集約している様子を表している．最初に登録されたチケットを親とし，コメントにアラートが再発（再通知）した日次を記録している．この場合は，最初に作成されたチケット内にコメント追記として集約する．これにより，通知頻度が高いホストに対

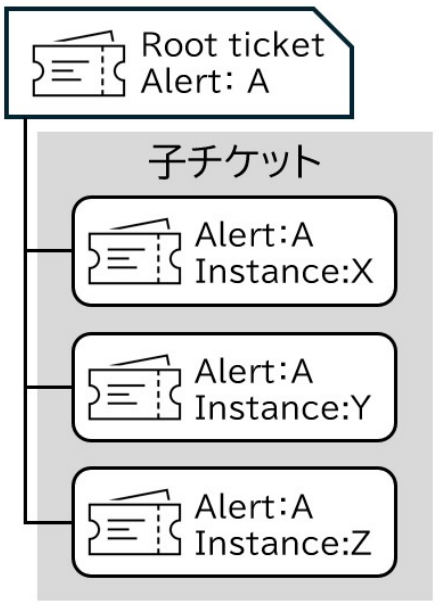

図 5: 同じアラートだがホストは異なる場合

するチケット件数の増加を防ぎつつ，時系列的な再発履歴を残すことができる．

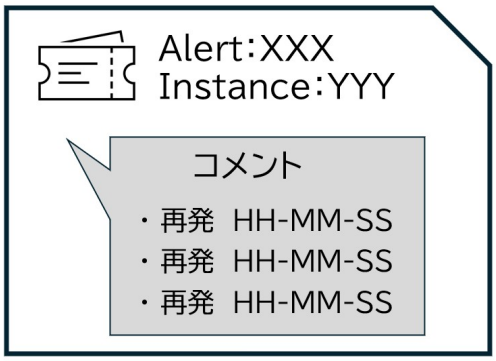

図 6: 同じアラート名で，なおかつホストも同じ場合

ルートチケットの子チケットに登録された場合の再発時も，その子チケットのコメントに追記することで再発履歴を残す．また作成されたチケットについては，24 時間でチケットを整理するために，持ち越し処理を行い，「持ち越し」ステータスへと変更される．これにより，監視担当者は翌日の業務開始時点で未完了チケットを明確に把握でき，対応の優先度の判断や進捗の確認を容易することができる．

ユースケース・シナリオ

CDSL のシステム Prometheus，Alertmanager からアラートが通知され，提案ソフトウェアによりチケットが作成，集約され，Redmine に登録される状況を想定する．図 7 にユースケースシナリオを示す．Prometheus，Alertmanager はシステム監視システムを構成するソフトウェアである．ticketcreate_evo は提案するソフトウェアである．

アラートを受信する Fast API と Redmine から情報を取得する Rest API と，チケットのグルーピングを処理するチケットクリエイトの 3 つで構成されてる．Redmine は処理されたチケットを登録．監視担当者がそのチケットの調査を行う．

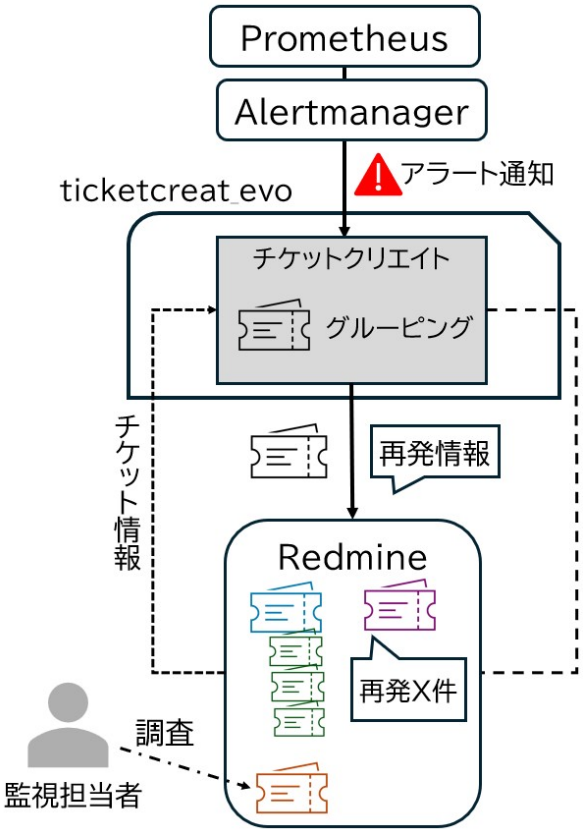

図 7: ユースケースシナリオ

監視対象の物理サーバや仮想基盤，ネットワーク機器から収集されたメトリクスは Prometheus に集められ，しきい値やヘルスチェックにもとづいて Alertmanager からアラートが通知される．チケットクリエイトが通知を受け取ると，通知情報からアラート名，instance，アラートの概要を記載している description を抽出する．その後，既存のチケット状況について REST API を通じて Redmine に対して検索をかける．まず同アラートを束ねる Root チケットの有無を確認し，存在すればそのチケットの子チケットとして登録し，存在しなければ新たに Root チケットを作成する．続いて，同じアラート名と instance の組み合わせで既存のチケットがあるかを調べる．同じホストでの再発であれば当該チケットのコメントに追記して再発履歴を残す．通知されたアラートはその日の内に対処をすることが望ましいが，対処されなかった場合は「持越し」ステータスに変更され，翌日以降，どのアラートが未対処であるかの判別をしやすくする．

4. 実装

提案方式をもとに作成した，チケット作成ソフトウェアである「ticketcreat_evo」について説明する．ソフトウェアは，Python3（ver. 3.12.3）で作成した．使用したパッケージ，ライブラリは以下の通りである．

- fast api（0.101.0）
- requests（2.21.0）
- uvicorn（0.27.1）

図 8 では，ticketcreate_evo.py の概要を示している．Prometheus，Alertmanager はシステム監視ソフトでアラートを通知する．webhook エンドポイントは通知されたアラートを受信する Fast API である．Rest API は，Redmine に登録済みチケットを取得する．チケットクリエイトでチケットのグルーピング判定と作成を行う．Redmine はチケットクリエイトで作成されたチケットを登録する．

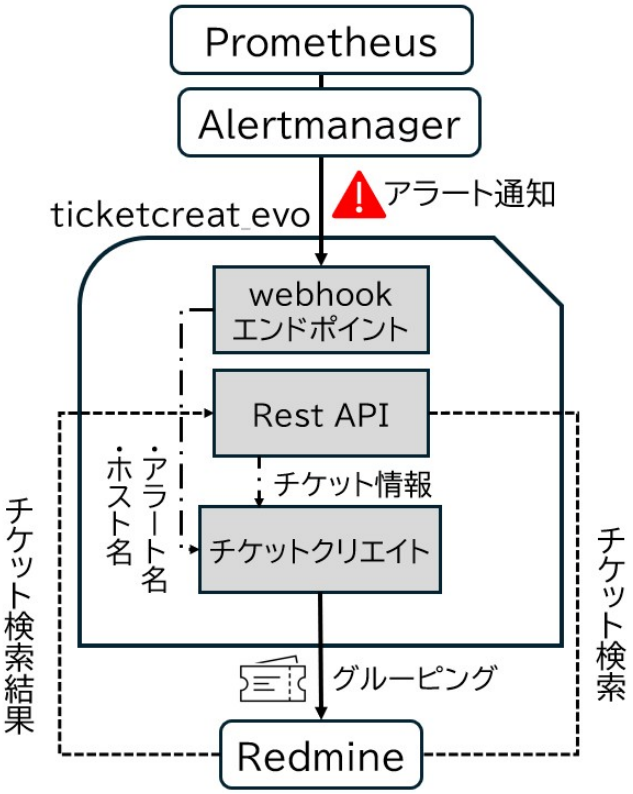

図 8: ticketcreate_evo の概要

まず，ticketcreat_evo 内の Fast API で動いているエンドポイントが Alertmanager からの通知を受け取ると，「アラート名」「instance」「description（説明）」を抽出する．次に，Redmine に対して，Rest API を通じて既存のチケットについて検索をかける．

最初に Root チケット（親チケット）を探す．題名が「[Root] [Alert] アラート名」の Issue を取得し，存在すれ

ば最初の 1 件を返す．続いて同じアラートかつホストのも同じ Issue が存在するかを確認する．存在すれば，そのチケットに再発した履歴を残すためにコメントに追記する．アラートは同じだがホストが異なる場合は Root チケットの子チケットとして登録する．

なおチケットの集約は，最後にチケットが登録，子チケット追加，コメント追記がされてから 4 時間経過するまでに再度アクションが起こらなければそのチケットについてはクローズする [14]．再度同じアラートがでた際は，新規のチケットとして登録して集約していく．この区切り方は，この提案がヒューリスティックにもとづいているからである．システム障害や負荷増大による再発は，アプリケーションリトライやジョブ再実行が運用サイクルの影響を受ける．これらは数時間の範囲で収束するため，「4 時間以内に再発したアラートは同じ原因である可能性が高い」としている．また，長い間一つに集約すると異なる原因によるアラートまでもが統合されてしまう．

5. 評価実験

評価実験では，全てのアラートをチケット化する場合と，提案ソフトウェアによりチケットを集約することで，Redmine 上に登録されるチケット件数の削減率を調べた．

実験環境

図 9 に実験を行う環境の構成図を示す．

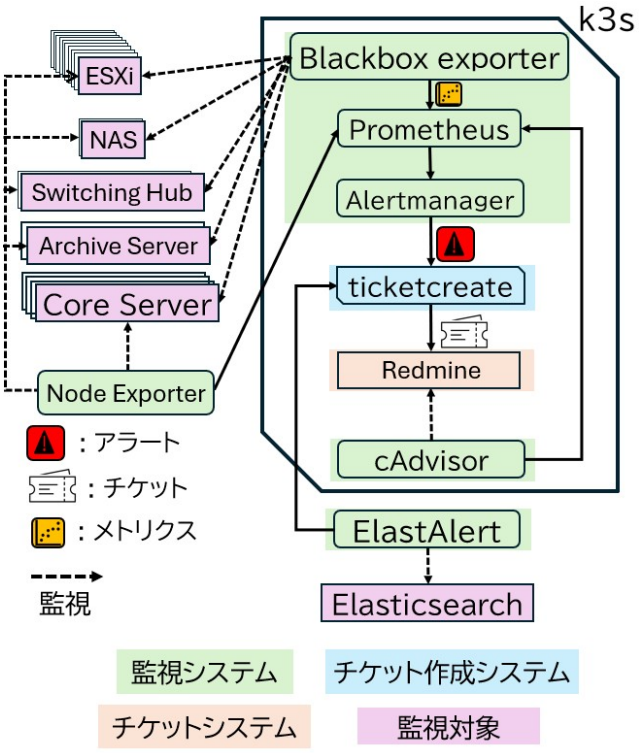

図 9: 実験環境の構成図

実験を行う環境として，監視システム，チケット作成システム，チケット管理システムを用意する．提案ソフトウェアを動かす VM のスペックは，vCPU が 2 コア，メモリが 2GB，ストレージが 25GB となっている．監視システムを動かすマシンのスペックは，vCPU が 4 コア，メモリが 8GB，ストレージが 40GB となっている．監視対象は，物理マシンのほかに，ネットワーク機器と物理マシンにインストールされた VMware ESXi に配置された VM がある．監視システムは，監視ソフトウェアに Prometheus，監視エージェントは Blackbox exporter，Node exporter，VMware exporter，cAdvisor，Metrics Server を使用する．アラートの通知には Alertmanager を使用する．Alertmanager から通知されたアラートは，ticketcreate がアラートチケットを作成し，Redmine に登録していく．またアラートが通知されると，Ticket Notifier が CDSL の Slack にアラートが通知されたことをポストする．

VMware ESXi をインストールした物理マシンは 10 台ある．その内訳は，研究室で実験を行う用途の物理マシンが 7 台，DNS や DHCP，Jump サーバを運用する基幹用の物理マシンが 2 台，外部にサイトを公開するための用途の物理マシンが 1 台となっている．NAS は 2 台あり，実験用の VMware ESXi の VM データを保存するストレージとして使用する．VMware ESXi の電源管理のために SwitchBot を導入している．1GB の Swichinghub は VMware ESXi に対して通信を行うためのネットワークを経由する目的で使用する．10GB の Swichinghub は VMware ESXi と NAS を用いたストレージ間のネットワークを経由する目的で使用する．実験と本番環境間のファイアウォールは実験用の環境と本番環境で疎通を行う際に経由する．LiDAR 送信用 Raspberry Pi は LiDAR が取得したデータを本番環境上の VMware ESXi である Iris 上の VM に送信するために使用される．アーカイブ用の Ubuntu は実験環境で使用している VM のデータの中で登録が解除されたデータをアーカイブする目的で使用している．

実験結果と分析

チケット削減率

削減率の評価では，アラート名と instance をキーとし，再発間隔が 4 時間未満であれば同じ事象として束ね，4 時間以上であれば新規として分割するという提案手法と比較対象を用いた．

図 10 は，10 月 21 日〜11 月 1 日の期間のチケットについて，全てのアラートをチケット化した場合と，提案ソフトウェアを適用した場合のチケット数を比較している．Redmine の Web UI 上での登録件数は，未集約場合，チケットが 342 件であったのに対し，提案ソフトウェアによる集約後は 135 件となり，約 61%削減できたことを確認した．特に，10 月 26 日は 96 件と，他の日と比べても多

くのチケットが登録されていたが，提案ソフトウェアにより 31 件まで集約でき，約 67%削減することができた．グルーピングされた 135 件の内，異なるアラート名かつ異なる instance のアラートを 1 つのチケットにグルーピングしていることはなかった．一方で，同じアラート名かつ同じ instance のチケットであるが，直近 4 時間以内にそのアラートが通知されていなかったため，グルーピングされずに独立した状態のチケットが 62 件存在した．

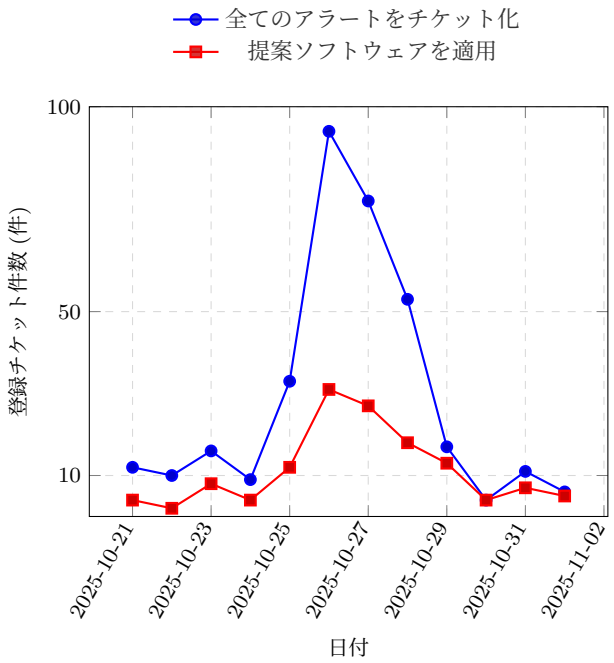

図 10: 未集約と集約後のチケット数比較

集約ウィンドウの変化による統計量の比較

前項において，チケットをグルーピングする集約ウィンドウを 4 時間としていた．本項では，集約ウィンドウを変更した際の各統計量の変化を比較する．評価には，2025 年 10 月 21 日から 11 月 26 日までの期間に，Redmine に登録された 1101 件のチケットを使用する．表 2 は，集約ウィンドウごとのチケット数と統計量をまとめたものである．

表 2: 集約ウィンドウごとのチケット数と統計量

Win [h]	Tickets	Alerts / ticket		Duration / ticket [min]	
		Avg	P95	Avg	P95
1	770	1.43	3.00	8.7	51.5
2	691	1.59	3.40	19.4	104.8
4	621	1.77	3.00	41.5	214.0
6	575	1.91	4.00	70.2	356.0
8	542	2.03	4.00	99.3	607.5
12	412	2.67	5.00	345.0	1470.5

「Win [h]」は，グルーピングする時間の長さを表す．「Tickets」は，グルーピングを行った後のチケット数を表す．「Alerts / ticket – Avg」は，グルーピングにより，1 件の親チケットにまとめられたチケットの平均数を表す．「Alerts / ticket – P95」は，下位 95%が 1 件の親チケットがにまとめられているチケットが，その値以下であることを表す．「Duration / ticket [min] – Avg」は，1 件の親チケットが，平均して何分間オープンの状態になっているかを表す．「Duration / ticket [min] – P95」は，は，下位 95%が親チケットが何分以内にクローズしているかを表す．

チケット数の比較

ここでは，集約ウィンドウを 1, 2, 4, 6, 8, 12 時間の 6 パターンに変更して，チケット数を比較する．各ウィンドウでグルーピングを行い，登録されたチケットの件数をカウントした．表 3 は集約ウィンドウごとのチケット件数を示している．

表 3: 集約ウィンドウごとのチケット件数

集約時間	チケット件数
1 時間	770
2 時間	691
4 時間	621
6 時間	575
8 時間	542
12 時間	412

1，2，4 時間では上記のように減少し，1 時間から 2 時間，2 時間から 4 時間に変化させると，それぞれ約 10%と，同じ割合でチケット件数を減らせていることが分かる．これは，短い間隔で連続的に発生するアラートが，4 時間程度の窓であれば適切に 1 つの問題としてまとめられている．

一方で，グルーピング時間を 6，8，12 時間からと伸ばした場合は，4 時間から 6 時間では約 7%，6 時間から 8 時間では約 6%の減少率となった．1 時間から 4 時間では約 10%減少していたが，6，8，12 時間では減少率が鈍くなっているのがわかる．この傾向に反して，8 時間から 12 時間では約 24%チケット数が減り，これは逆にまとめ過ぎているといえる．

以上より，1 時間から 4 時間の区間では集約ウィンドウを延長することによってチケット数の大幅な削減効果が得られる一方で，4 時間を超えて 6 時間，8 時間とウィンドウを伸ばしたとしても，追加で得られる削減効果は限定的である．チケット数の削減という観点では，「4 時間まで伸ばすとメリットが大きく，それ以降は効果が見込めない」というエルボーポイントが存在している．

1 件の親チケットあたりのアラート件数，継続時間の比較

各集約ウィンドウにおける「1 件の親チケットあたりのアラート件数と継続時間（最初のアラート発生時刻から最後のアラート発生時刻まで)」を比較する．まず，1 件の親チケットに含まれるアラート件数（コメント件数）に着目し，平均値の推移を見る．表 5 は集約ウィンドウごとの 1 件の親チケットに含まれるアラート件数を示す．

表 4: 親チケットあたりの平均件数

時間	平均件数
1 時間	1.43 件
2 時間	1.59 件
4 時間	1.77 件
6 時間	2.06 件
8 時間	2.03 件
12 時間	2.67 件

集約ウィンドウを長くするに従い，親チケットあたりに含まれるアラート件数は緩やかに増加している．中央値はいずれのウィンドウでも 2 件前後にとどまっており，多くのチケットは少数件のアラートから構成されていることが分かる．しかし，95 パーセンタイル値（上位 5%に位置する「多いチケット」の件数）を見ると，1 から 4 時間幅では約 3 から 4 件だが，12 時間では 5 件前後に増加した．これにより，ウィンドウを極端に長くすると「ごく一部だが，非常に多くのアラートを抱えるチケット」が増える傾向を確認することができた．1 件の親チケットに含まれるアラート件数が多すぎると，原因や影響範囲が異なる事象が一つのチケットに混在したり，タイムラインが長くなり，対応履歴が追いづらくなる問題が生じることになる．4 時間のウィンドウでは，平均，中央値ともに少数件の範囲に収まっており，95 パーセンタイルでも数件程度であるため，1 件のチケットの大きさとしては比較的扱いやすい件数に収まっているといえる．

次に，チケット継続時間の分布を比較する．表 6 は，集約時間ごとのチケット継続時間を示している．

特に 8 時間および 12 時間の設定では，95 パーセンタイル値が 10 時間〜1 日規模となっており，少なくない数のチケットが「1 日近く継続している」とみなされることが分かる．この場合，日をまたいだ長大なチケットが増えることになり，障害の切り分けや収束判定を困難にする要因となりうる．それに対して 4 時間で集約する場合の平均継続時間は 1 時間未満で，95%のチケットが約 3 から 4 時間以内に収まるという結果になっている．これより，多くのチケットが 1 回の対応の中で完結する規模に留まっている．このことから，4 時間のグルーピングは，過度に長期のチケットを生み出すことなく，タイムラインを保てていると評価できる．

表 5: 集約時間ごとのチケット継続時間

時間	平均継続時間	上位 5%の継続時間
1 時間	8〜10 分	1 時間弱以内
2 時間	40 分前後	2〜3 時間程度
4 時間	40〜50 分	3〜4 時間程度
8 時間	300 分超	10 時間前後
12 時間	300〜350 分	16〜24 時間前後

1 種のアラートに着目した時の集約ウィンドウの評価

「Internal Host CPU UsageHigh-Usage100%」というアラートのみに着目した際の適した集約ウィンドウを検証する．このアラートを選択した理由は，2025 年 10 月 21 日から 11 月 26 日の期間に登録された 1101 件のチケット中で最も多い 356 件であったからである．

図 11 は，アラートの発生間隔分布を示したものである．横軸は，アラートの発生間隔を表し，縦軸は発生間隔ごとの件数を表している．

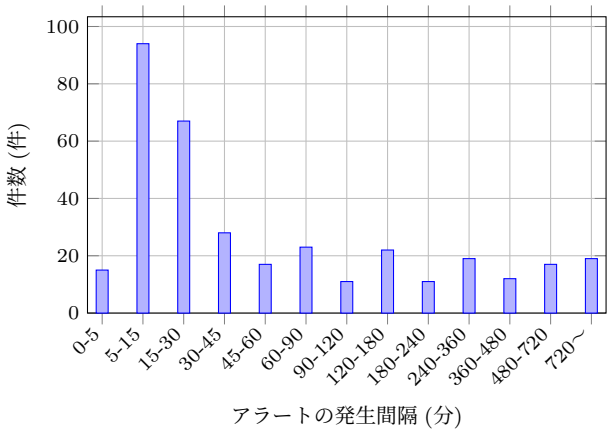

図 11: アラート発生間隔の分布

アラートの発生間隔で分けて分布を集計したところ，0 から 5 分が 15 件，5 から 15 分が 94 件，15 から 30 分が 67 件，30 から 45 分が 28 件，45 から 60 分が 17 件，60 から 90 分が 23 件，90 から 120 分が 11 件，120 から 180 分が 22 件，180 から 240 分が 11 件，240 から 360 分が 190 件，360 から 480 分が 12 件，720 分以上が 19 件であった．0 分から 30 分の 3 区画で，約 49.6%の 176 件を占めており，発生間隔のほぼ半数が 30 分以内の短時間での再発であることが分かる．

表 6 は，発生間隔に対するアラート発生割合を示している．表より，全発生間隔の約半分は 30 分以内，約 6 割は 60 分以内に収まっており，残りは数時間から半日以上空いてから再発していることがわかる．本アラートの障害進行中では 30 分前後の間隔で連続的に発生し，収束した後は数時間から十数時間にわたりアラートがない期間が続くというフェーズを繰り返している．

表 6: 発生間隔ごとのアラート件数の割合

区間（分）	割合
30 以下	約 49.6%
30–60	約 12.7%
60–120	約 9.6%
120–240	約 9.3%
240–720	約 13.5%
720 ～	約 5.4%

30 分を集約ウィンドウとした場合，0 から 30 分の発生間隔は 1 件に集約できるが，30 から 60 分のアラートをまとめることはできない．この区間は，ピークからしばらく時間が空いた後の再発であり，この間も CPU の負荷は依然として高い状態であると考えられるため，1 件にまとめるのが適切であると考える．以上より，0 から 30 分の短時間での再発を確実に取り込みつつ，30 から 60 分の少し離れた再発も同一の障害として扱うことができるため，「Internal Host CPU UsageHigh-Usage100%」では 60 分の集約ウィンドウが適していると考える．

6. 議論

提案したアラートのグルーピングは，主にアラート名および instance をキーとした単純な一致判定を基準としている．一方で，異なるアラート名であっても，同じ原因による事象である可能性も存在する．例を挙げると，設定しているアラートの中に，「Internal Core Server Cluster Node OOM Kill」と「Internal Monitoring Cluster Pod Memory Usage High」がある．OOM は，Out of Memory の略称で，システムのメモリが物理的に不足した際に，Linux カーネルが特定のプロセスを強制終了する機能である．つまり，OOM Kill と Memory Usage High のアラートは同時に発生するアラートであるといえる．以上のことからノードと Pod の関係を定義し学習させることで，異なるアラートでも集約させることができる．

本稿では，アラートを一定時間幅で集約して 1 件のインシデントとして扱う手法を導入したが，解析の結果，アラート種別ごとに適切な集約ウィンドウの長さが大きく異なることが明らかになった．評価の項でも述べたが，「Internal Host CPU UsageHigh-Usage100%」に適切な集約ウィンドウは，60 分が適しているという結論になった．単一の固定ウィンドウを全アラート種別に一律に適用する設計では不十分であるといえる．そのため，直近のインターバル分布や 1 時間あたりの発生件数などから，集約ウィンドウを動的に伸縮させるアルゴリズムを導入することも有効であると考える．

また，本稿での手法は，アラート名と instance，および短期間での再発を根拠としたヒューリスティックな集約手法となっており，明示的なメトリクス解析やモデル学習は行っていない．今後，より自動的かつ高精度なアラート統合を目指す場合には，機械学習を用いた手法が有効であると考えられる．メトリクス系列やアラート文面，発生間隔，物理，論理トポロジ特徴を組み合わせた学習アプローチにより，因果構造の自動検出を目指すことができる．特に，クラスタリングやグラフベース手法を用いることで，アラート同士の「時空間的，意味的距離」を定量化し，人手によるルール設計を最小化できる．

7. おわりに

東京工科大学コンピュータサイエンス学部の研究室である Cloud and Distributed Systems Laboratory（CDSL）は，各種監視エージェントと Prometheus を利用した監視システムを運用している．課題は CDSL の監視システムにおいて，各アラートに対して，1 件のアラートチケットを作成していることである．同じ原因によるものが継続，または断続的に再発している場合でも，チケット単位では分断されて記録されるため，対応しなければいけないチケットの見直しや，チケットを監視作業者がまとめる作業が発生する．また，アラートに対して，調査，対処を行うにあたり，同じアラートのチケットが登録されていると，複数の監視担当者が同じ調査を繰り返すことで，人的，時間的リソースの無駄を生んでいる．この課題に対して，監視ソフトから通知を受信すると通知内容を解析し，条件に合わせてチケット管理ソフトに登録，更新する．チケット間の関連性を自動判定し，親チケットと子チケットを用いて階層構造化することで，アラート原因単位での管理を提案．原因単位で集約することで，同じ障害に対する重複対応を防ぐ．また，対応履歴が一元化され，運用負荷と確認漏れを削減できる．評価では 2025 年 10 月 21 日～11 月 1 日に作成された 342 件のチケットを使用して，全てのアラートを登録していた場合と，提案のグルーピングを適用した場合を比較した．その結果，全てのアラートをチケットとして登録していた場合は 342 件あったが，グルーピングすることで，135 件までまとめられ，約 61%削減することができた．特にチケット件数が多かった 10 月 26 日は 96 件登録から 31 件まで集約でき，約 67%削減した．グルーピン

グした 135 件の内，異なるアラート名かつ異なる instance のアラートを 1 つのグルーピングしていることはなかった．一方で，同じアラート名かつ同じ instance のチケットだが，グルーピングされずに独立した状態のものが 62 件存在した．

参考文献

[1] Ling, C., Xiaoming, W., Ping, X., Yong, Z., Xuezhi, Z., Xianzhou, C., Mengyao, F., Yueqi, G. and Shuang, X.: Based on Zabbix-Prometheus Group Classification Alarm System, *Journal of Physics: Conference Series*, Vol. 2665, p. 012010 (online), DOI: 10.1088/1742-6596/2665/1/012010 (2023).

[2] Aglibar, K., Alegre, G., del Mundo, G., Duro, K. and Rodelas, N.: Ticketing System: A Descriptive Research on the Use of Ticketing System for Project Management and Issue Tracking in IT Companies (2022).

[3] Aglibar, K. and Rodelas, N.: Impact of Critical and Auto Ticket: Analysis for Management and Workers Productivity in using a Ticketing System, *International Journal of Computing Sciences Research*, Vol. 6, pp. 988–1004 (online), DOI: 10.25147/ijcsr.2017.001.1.84 (2022).

[4] Montgomery, L. and Damian, D.: What do Support Analysts Know About Their Customers? On the Study and Prediction of Support Ticket Escalations in Large Software Organizations, *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pp. 362–371 (online), DOI: 10.1109/RE.2017.61 (2017).

[5] Salah, S., Maciá-Fernández, G. and Díaz-Verdejo, J. E.: Fusing information from tickets and alerts to improve the incident resolution process, *Information Fusion*, Vol. 45, pp. 38–52 (online), DOI: https://doi.org/10.1016/j.inffus.2018.01.011 (2019).

[6] Montgomery, L. and Damian, D.: What do support analysts know about their customers? on the study and prediction of support ticket escalations in large software organizations, *2017 IEEE 25th international requirements engineering conference (RE)*, IEEE, pp. 362–371 (2017).

[7] Jani, Y.: Unified Monitoring for Microservices: Implementing Prometheus and Grafana for Scalable Solutions, *Journal of Artificial Intelligence, Machine Learning and Data Science*, Vol. 2, pp. 848–852 (online), DOI: 10.51219/JAIMLD/yash-jani/206 (2024).

[8] B.C., P., Maddirala, H. and M., S.: Implementing an effective Infrastructure Monitoring Solution with Prometheus and Grafana, *International Journal of Computer Applications*, Vol. 186, pp. 7–15 (online), DOI: 10.5120/ijca2024923873 (2024).

[9] Jani, Y.: Unified monitoring for microservices: Implementing prometheus and grafana for scalable solutions, *J Artif Intell Mach Learn & Data Sci*, Vol. 2, No. 1, pp. 848–852 (2024).

[10] Lin, D., Raghu, R., Ramamurthy, V., Yu, J., Radhakrishnan, R. and Fernandez, J.: Unveiling clusters of events for alert and incident management in large-scale enterprise it, *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (online), DOI: 10.1145/2623330.2623360 (2014).

[11] Liu, Z., Benge, C. and Jiang, S.: Ticket-BERT: Labeling Incident Management Tickets with Language Models (2023).

[12] Zha, J., Shan, X., Lu, J., Zhu, J. and Liu, Z.: Leveraging Large Language Models for Efficient Alert Aggregation in AIOPs, *Electronics*, Vol. 13, No. 22, p. 4425 (2024).

[13] Salah, S., Maciá-Fernández, G. and Díaz-Verdejo, J.: Fusing Information from Tickets and Alerts to Improve the Incident Resolution Process, *Information Fusion*, Vol. 45 (online), DOI: 10.1016/j.inffus.2018.01.011 (2018).

[14] Kabiri, P. and Ghorbani, A. A.: A Rule-based Temporal Alert Correlation System., *Int. J. Netw. Secur.*, Vol. 5, No. 1, pp. 66–72 (2007).