マイクロサービスの依存数にもとづいたアラートの重大度による緊急性の高いタスクの割り当て

近藤 悠斗[1]　平尾 真斗[1]　串田 高幸[1]

概要：EC サイトはマイクロサービスを使用して構築される．マイクロサービスは監視ソフトウェアを使用して監視する．監視のアラートには重大度を設定する．課題は，重大度を決めることができないことである．重大度が決定できないと，アラートの修正順序が決定できない．その結果サービスの復旧時間の増加につながる．提案は，対象となるサービスがいくつのサービスに依存されいるかを数え重要度を決定する．すべてのサービスの依存に比べて多くのサービスに依存されているサービスは重要度を高く設定する．それによって，アラートの解決に優先順位をつけることができる．評価は，Sock Shop に対して，Locust を使用して負荷を与えた．ユーザ数は 300 人で，1 ユーザは毎秒 1 回リクエストを行う．1000 秒間の負荷を与え，10 秒に 1 回監視を行った．シナリオは，まずフロントページにアクセスし，ログインを行う．次にカタログの中からランダムに商品をカートに入れて，最後に購入を行う．すべてのアラートをそのまま出力したものと，提案ソフトウェアによって削減したアラートを比較した．既存手法では 262 件のアラートが発生し，提案手法では 201 件のアラートが発生した．また発生したアラートの 72 %は重大度 4 のアラートで，28 %は重大度 3 のアラートだった．重大度の設定によりアラートを 22%削減し，重大なアラートが多く発生していることがわかった．

1. はじめに

背景

マイクロサービスとは，ビジネス機能を実行するためにシステムを複数のサービスに分割するアーキテクチャである [1]．例えば，Amazon や Netflix はマイクロサービスを使用して EC サイトを構築している [2–4]．マイクロサービスは 1 つの機能を複数のサービスを連携させて実現する．そのため，複数のサービスから依存されているサービスが停止した場合，そのサービスに依存しているサービスも停止し，カスケード障害が発生する [5]．カスケード障害とは，対象のサービスが停止すると他のサービスも停止する障害である．

EC サイトはサーバ上に構築される．サーバの運用を継続的に監視するために，監視ソフトウェアが導入される．監視ソフトウェアには異常が発生したときに管理者に通知するためにアラートの設定を行う．アラートが設定されていない場合，管理者は異常が起こっていることを発見するのに時間がかかる．そのためアラートの設定は必要である [6]．管理者はソフトウェアや機器に合わせたアラートを設定することで，障害が発生していることに気付くことができる．

マイクロサービスではレイテンシの監視を行っている [6]．アラートを作成できる監視ソフトウェアとして Prometheus や Nagios が挙げられる [7, 8]．アラートには重大度が設定できる．重大度は管理者がサービスの構成や障害が発生したときの影響を元にして決定する．重大度の設定は人によって異なる． 重大度を設定することで，アラートが発生した際，問題を修正する順番を決定することができる [9]．例えばすべてのアラートの重大度を Critical に設定すると，複数のアラートが同時に発生したときにどれから修正をすればいいのか分からない．

課題

課題は，アラートに対してどの程度の重大度をつければいいか決めれないことである．例えば Prometheus では AlertManager を使用することでアラートを作成することができる．AlertManager の設定ファイルでレイテンシの監視を行う設定を設定ファイル 1 に示す．1 行目では app latency という名前のアラートを定義している．2-7 行目ではアラートが発生する条件を定義している．ここでは 1 分間の http リクエストの平均レイテンシが 100ms 以上の時にアラートが発生する条件になっている．8，9 行目にはアラートの重大度を定義している．重大度は自由な文字列，

[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

設定ファイル 1 アラートマネージャの設定ファイルの例

```
- alert: app latency
  expr: |
    rate(
      request_duration_seconds_sum[1m]
    ) / rate(
      request_duration_seconds_count[1m]
    ) >= 100
  labels:
    severity: critical
```

数値で定義することができる．例えば hoge，fuga，page，critical, error, warning, info，1，2，3，4 である[*1]．この場合，重大度に critical が設定されている．

重大度は管理者が自分で設定する．重大度決定の課題を図 1 に示す．これは Sock Shop のサービスの一部である．Sock Shop はマイクロサービスでの EC サイトのデモアプリケーションである [10]．Frontend は Web サイトのページを表示するサービスである．Carts は購入予定の商品を保管するカートを管理するサービスである．Orders 購入予定の商品に対して購入処理を行うサービスである．具体的にどの重大度をつけるか決めるには他の複数のサービスとの関係性が必要になる．例えば，そのマイクロサービスにアクセスするための窓口となるサービスがあった場合には，そのサービスが停止すると，すべてのマイクロサービスが使用できなくなる．図 1 では Frontend が停止すると，Carts と Orders が停止していなくても，商品をカートに追加することと商品を購入することができなくなる．そのため，Frontend を Carts と Orders よりも先に修正する必要がある．管理者は重大度にもとづいて修正作業の順序と作業タイミングを決める [9]．そのため重大度が適切に設定されていないと，以下の 2 つの問題が発生する．

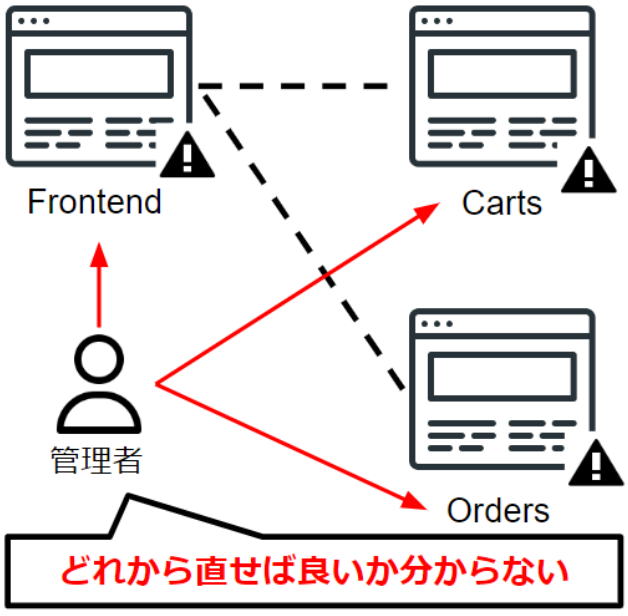

図 1 重大度決定の課題

(1) **修正順序の誤り**: 複数のアラートが発生したとき，アラートの修正順序を誤って決めてしまう．誤った順序で修正を行うと，サービス復旧までの時間の増加につながる．例えば，Frontend と Carts でアラートが発生した際に Carts から修正作業を行っても，Frontend が修正されていないため，ユーザは Carts の機能を使用することができない．

(2) **順序決定の工数の増加**: 複数のアラートが同時に，表示されると，そのすべてに修正作業の順番を付ける必要があるため，修正作業の順番決定の工数が増える．例えば，Frontend と Carts で発生したアラートに同様の重大度が設定されていた場合，それぞれのサービスがどのような依存関係を持っているのか確認する必要がある．工数が増えると，サービス復旧までの時間の増加につながる．

各章の概要

2 章では, アラートの重大度決定について関連する既存研究を紹介する．3 章では, マイクロサービスの依存数をもとにしてアラートの重大度を決定する手法を提案する．4 章では, 実装したソフトウェアについて説明する．5 章では, アラートの発生数，重大度ごとのアラート数の割合を比較する．6 章では, 依存数の決定式について議論し，改善案を提示する．最後に 7 章で本稿のまとめを行う．

2. 関連研究

ディープラーニングを使用してアラートを設定する手法がある [6]．この研究ではアラートの設定に，サービスの稼働状況のデータを用いる．稼働状況のデータを取得するためには，実際にサービスを稼働させる必要がある．そのため，新しく作成された直後のサービスは対象としていない．

DevOps によって生成されたフィードバックを元にしてアラートを修正する手法がある [11]．この研究ではサービス運用で発生したアラートを元にして，アラートの条件を修正している．そのため，新しく作成された直後のサービスは対象としていない．

HPC 環境での Shasta のアラートに関する研究がある [12]．この研究では事前に設定したルールを元にして，アラートの優先順位を決定している．これは事前に定義された固定のルールであるため，構造が柔軟に変化するマイクロサービスには適用できない．

[*1] https://datatracker.ietf.org/doc/html/rfc5424#section-6.2.1

3. 提案

提案方式

本稿では，サービスの依存数をもとにして，アラートの重大度を決定し重大度をもとにアラートを削減して出力する方法を提案する．提案する手法の流れを図 2 に示す．管理者は設定ファイルに監視項目を記述する．提案ソフトウェアは，サービスから取得した経路情報をもとに重大度を決定し，設定ファイルに適用する．監視システムは設定ファイルの内容に従ってサービスの監視を行う．新たにサービスが追加された時には提案ソフトウェアが再度経路情報から重大度を計算し，設定ファイルを更新する．

提案ソフトウェアの詳細を図 3 に示す．まず，サービス間の経路情報を取得する．これによって，各サービスが他のどのサービスに依存されているか取得する．次に，依存されているサービスが更に他のどのサービスに依存されているかを取得する．これによって，そのサービスが停止した時に間接的なものも含めたいくつのサービスが利用できなくなるか分かる．最後に重大度の計算式を使用して，4 段階の重大度を導く．リッカート尺度では 4 段階を推している [13]．理由は，4 段階が人間が選択肢の内容を認識しやすい数だからである．アラートでも重大度の度合いを認識しやすくする必要がある．またリッケルト型スケールと呼ばれる重要性を評価する指標がある [13]．これは重大度

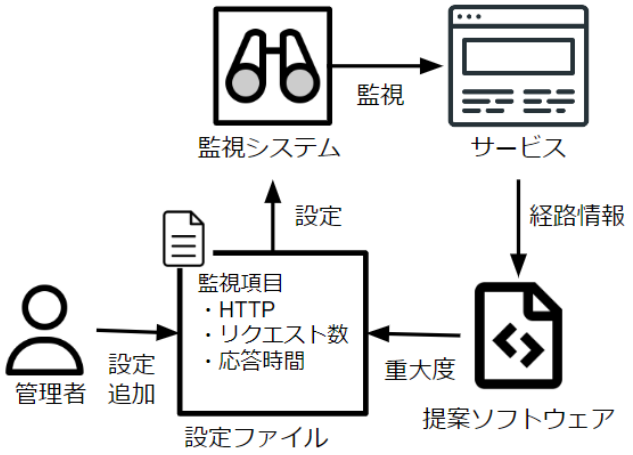

図 2 提案手法の流れ

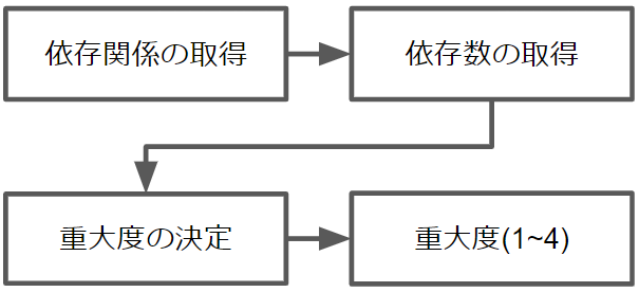

図 3 提案ソフトウェアの詳細

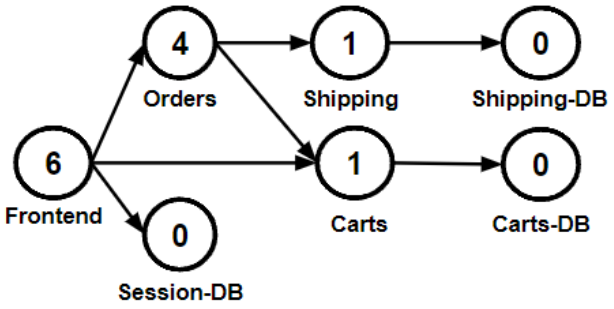

図 4 提案の例

と同様に，対象が全体に対してどれだけの影響を持つかの指標として優れている．そのため 4 段階に分類する．重大度の計算式を式 (1) に示す．

$$D\_Range = D\_Max - D\_Min \tag{1}$$

依存数はそのサービスが他のサービスに依存されている数を示す．D_Max は最大依存数で全サービスの中で最も多く依存されている数を示す．D_Min は最小依存数で全サービスの中で最も少なく依存されている数を示す．D_Range は依存範囲でサービスの依存数の範囲を示す．依存範囲は最大依存数から最小依存数を引いて導く．重大度は依存範囲を 4 分割し，どの範囲に含まれるかによって決定する．重大度は 1 が最も重大ではなく，4 が最も重大となる．

例として，図 4 の Sock Shop の一部のサービスに提案を使用して重大度を決定する．矢印は通信経路を示している．例えば Frontend は Orders，Session-DB に依存されている．更に Orders は Shipping，Carts に依存されている．そのため，Frontend は Shipping と Carts にも依存されていると言える．同様に全ての依存関係を確認すると Frontend はサービス Orders，Shession-DB，Shipping，Carts，Shipping-DB，Carts-DB に依存されていると言える．同様に Orders なら Shipping,Shipping-DB,Carts,Carts-DB，Shipping ならば Shipping-DB に依存されていると言える．ここからいくつのサービスに依存されているかを数える．例えば Frontend ならば 6 つ，Orders ならば 4 つとなる．次に依存範囲を導く．最大依存数は Frontend の 6，最小依存数は Carts-DB の 0 であるため依存範囲は 6 になる．最後に依存範囲をもとにして，重大度を決定する．重大度が 4 となるのは依存範囲の 3/4 の数値以上の依存数のものとなる．つまり依存範囲 6 では 4.5 となり，依存数 5 以上のサービスは重大度が 4 となる．よって Frontend は重大度 4 になる．同様に，Orders は重大度 3，その他は重大度が 1 になる．

重大度を元にしてアラートを通知する頻度を変更することによって，一度に表示されるアラートの数を減らすことができる．

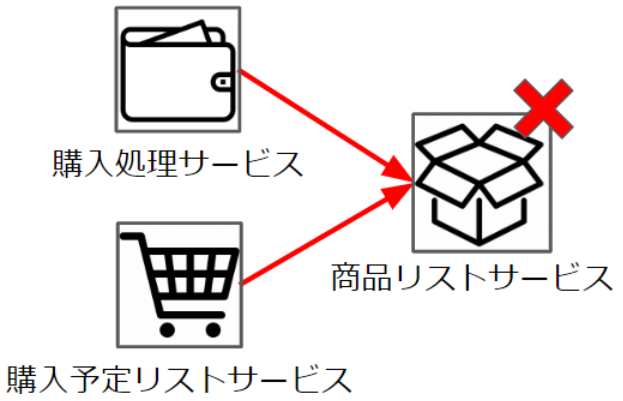

図 5 ユースケース

ユースケース・シナリオ

ユースケースとして EC サイトを想定している．例えばサイト内に新たにユーザが作成できる購入予定商品のリスト機能を追加した状況である．図 5 にユースケースを示す．このとき，購入予定リストサービスは，商品リストサービスに依存している．商品リストサービスは購入処理サービスからも参照されている．機能追加前は，商品リストサービスが停止すると，購入処理サービス 1 つが停止する状況だった．機能追加後は，購入予定リストサービスを含む 2 つのサービスが停止してしまう．本提案を使用すると，商品リストサービスのアラートの重大度が上がり，実際にサービスが停止する前に修正を要求するため，2 つのサービスが停止する状況を回避できる．

4. 実装

提案ソフトウェアは設定部分とアラート部分で構成されている．設定部分は重大度を計算する level_calculator.py と重大度からアラート設定を行う level_set.py，重大度が記録される level_data.csv が含まれる．アラート部分は，アラートを行う alert.py とアラートの設定を記述した alert.yaml が含まれる．

設定部分の動作の流れを図 6 に示す．初めに level_set.py

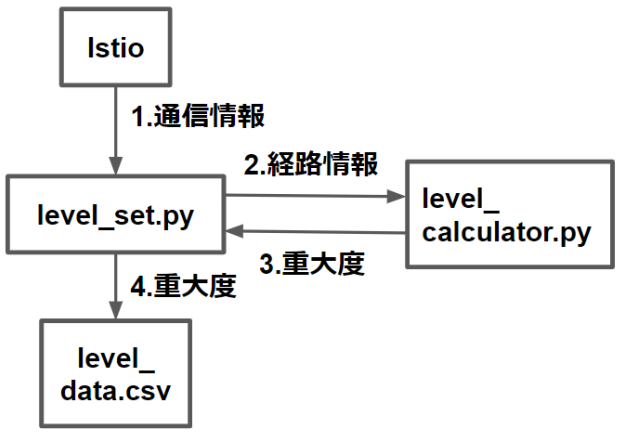

図 6 設定部分の動作の流れ

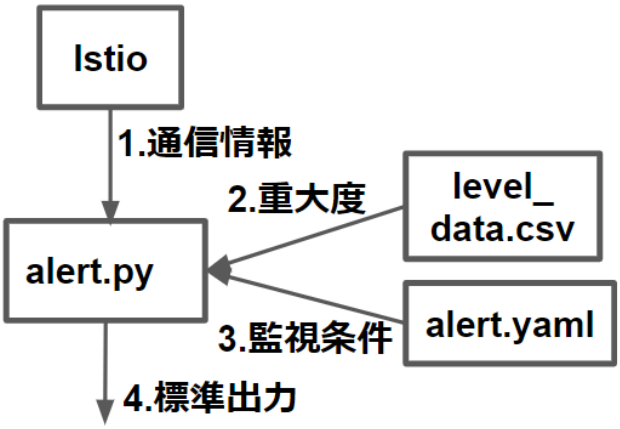

図 7 アラート部分の動作の流れ

が Istio から Pod とサービスの ID と名称，Pod とサービス間の通信情報を取得する．その後 level_calculator.py に ID と経路情報を渡す．level_calculator.py では受け取った経路情報を元に計算を適用し，ID と重大度のリストを返す．level_set.py は受け取った重大度を level_data.csv に出力する．

アラート部分の動作の流れを図 7 に示す．alert.py は alert.yaml に記述された内容を元に監視を行う．データは Istio から取得する．alert.yaml に記載された監視条件を超えた場合には，level_data.csv から重大度を取り出しアラート内容に重大度を付け標準出力を行う．

5. 評価実験

実験として Sock Shop に対して Locust を使用して負荷を与え，提案ソフトウェアからレイテンシのアラートを発生させる．初めに Sock Shop の各サービスにアラートを設定する．アラートの設定内容を設定ファイル 2 に示す．監視項目は HTTP 通信のレイテンシである．Pod を作成してから，各サービスに対して 1 度ずつアクセスをした時のレイテンシの 2 倍をアラート条件とした．queue-master, payment, user, shipping, catalogue は 10ms 以上のレイテ

設定ファイル 2 アラートの設定内容

```
{
    "queue-master": 10,
    "payment": 10,
    "orders": 600,
    "user": 60,
    "front-end": 2000,
    "carts": 250,
    "shipping": 10,
    "catalogue": 80
}
```

ンシが発生したときにアラートが出る．carts は 250ms 以上のレイテンシが発生したときにアラートが出る．orders は 600ms 以上のレイテンシが発生したときにアラートが出る．front-end は 2000ms 以上のレイテンシが発生したときにアラートが出る．次に locust を使用してユーザ数を 300 人で，1 ユーザが毎秒 1 回リクエストを行う負荷を 10 分間与える．負荷試験のテストシナリオとして SockShop のテストシナリオを用いる．シナリオの内容は以下の通りとなる．

(1) フロントページにアクセスする．
(2) ログインする．
(3) カタログの中からランダムに商品をカートに入れる．
(4) 購入する．

次に重大度を設定するために level_set.py を実行する．最後に 10 秒に 1 回の監視を 100 回行う．アラートの発生後は，設定した重大度を元に通知送信する．例えば Prometheus では，for という設定によって，条件がどれだけの期間続いたら，アラートが発生するかを設定することができる[*2]．本実験ではそれに従い，重大度ごとにアラート発生までの回数を設定した．重大度が 4 のアラートは毎回通知を送信する．重大度が 3 のアラートは 2 回連続でアラートが発生したときに通知を送信する．同様に重大度が 2 のアラートは 3 回，重大度が 1 のアラートは 4 回となる．

既存の手法として，発生したすべてのアラートを時系列順で送信するものを用意する．こちらは重大度が設定されていないため，提案手法のようなアラートの発生回数までの判断は行っていない．提案手法と既存手法で，修正作業の順番決定の工数を比較する．

実験環境

実験環境を図 8 に示す．研究室内のサーバに仮想マシン 1,2 を構築する．仮想マシン 1 には Ubuntu22.04 をインストールした. 仮想マシンの台数は 15 台である. 仮想マシンは同じ性能の (vCPU: 3[core], RAM:3[GB],SSD:25[GB]) で作成した. また仮想マシン上に K3s による Kubernetes [14] クラスタを用意しマスター 1 台, ワーカ 14 台の構成とした．仮想マシン 1 には Sock Shop を構築し，提案ソフトウェアも導入する．仮想マシン 2 には負荷試験のため Locust[*3]を導入する．提案ソフトウェアは SockShop を対象とし Discord にアラートを送信するように設定する．Locust は SockShop のテストシナリオを使用し，SockShop に負荷を与えるように設定する．

実験結果と分析

level_set.py によって設定された重大度を表 1 に示す．種類はそのリソースが Pod か Service かを表している．サービス名ではリソースの名称を表している．重大度は各リソースに設定された重大度を表している．

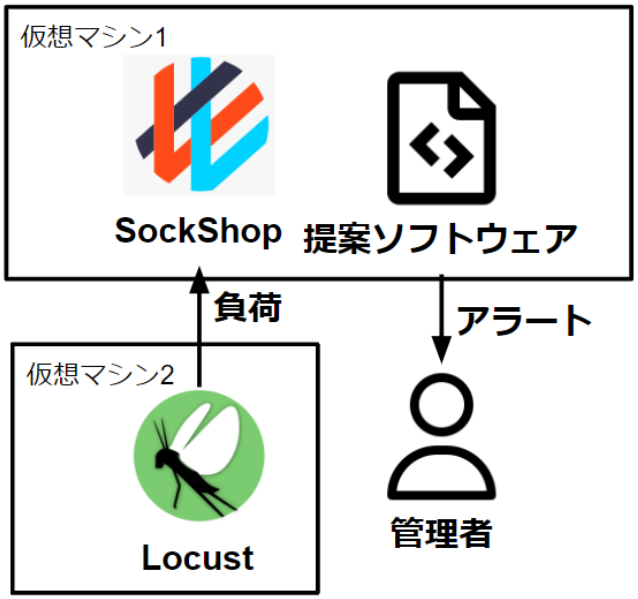

図 8　実験環境

表 1　作成された重大度

種類	サービス名	重大度
service	queue-master	3
service	payment	1
pod	orders	4
service	user-db	1
service	rabbitmq	1
pod	catalogue	2
service	orders	4
pod	shipping	2
service	users	3
pod	front-end	4
service	front-end	4
service	carts	3
service	catalogue-db	1
pod	queue-master	2
service	session-db	1
app	carts	2
service	shipping	3
pod	user	3
service	catalogue	3
service	carts-db	1
service	orders-db	2

発生したアラートの個数を図 9 に示す．既存手法では合計 262 件のアラートが発生した．対して提案手法では合計 201 件のアラートが発生した．front-end のアラートは既存手法では 100 件，提案手法では 99 件発生した．carts のアラートは既存手法では 97 件，提案手法では，48 件発生した．orders のアラートは既存手法では 47 件，提案手法では 46 件発生した．catalogue のアラートは既存手法で

[*2] https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/
[*3] https://developers.play.jp/entry/2023/02/03/143619

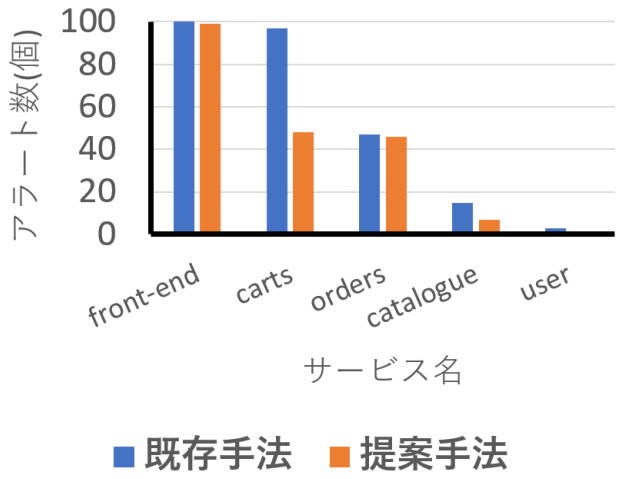

図 9 発生したアラートの個数

は 15 件，提案手法では 7 件発生した．user のアラートは既存手法では 3 件，提案手法では 1 件発生した．carts と catalogue は重大度が 3 である．そのためアラートの個数が約半分に減少している事がわかる．これにより，全体のアラートは 22%削減された．

発生したアラートでの重大度の割合を表 2 に示す．この表は提案手法で発生した 201 件のアラートの重大度の割合を示している．発生したアラートのうち 72%は重大度が 4 であり，28%は重大度が 3 である．重大度が 2 と 1 のアラートは発生しなかった．全アラートのうち約 3/4 が重大度 4 のアラートになっているため，重大なアラートを管理者に知覚させやすくなっている．

6. 議論

本稿ではサービスの依存数を元にしてアラートの重大度を決定した．本提案では重大度を 4 分割して決定したが，実際にはアラートの重大さは均等に分布していないと考えられる．そのためより正確に重大度を決定するためには，分割方法を変える必要がある．例えば，依存数の偏差をもとにして分割を行う方法が考えられる．この方法ならば，極端に多く依存しているサービスには他のサービスよりも強い重大度が設定されると考えられる．例えば今回の実験では Frontend の依存数が 26，Orders の依存数が 18 であり，他のサービスはすべて 4 以下だった．そのため，偏差を利用すると Frontend と Orders だけが重大度 4 に割り振られる．これは，重大なアラートの近くの向上につながる．またサービスの規模や特性によって別の分割数に変更する必要がある．例えばサービス数が多い場合には分割数が 4 のままでは 1 つの項目に含まれるサービスが増える．これは重大なアラートの知覚の阻害になる．そのため分割数を増やす．

表 2 アラートの重大度ごとの割合

重大度	4	3	2	1
既存手法のアラート数	147 件	115 件	0 件	0 件
既存手法の割合	56%	44%	0%	0%
提案手法のアラート数	145 件	56 件	0 件	0 件
提案手法の割合	72%	28%	0%	0%

現在の方法では依存数を元に数値を決定しているため，フロントエンドの重大度が高くなる．相対的にデータベースの重大度は低くなる．しかしデータベースのアラートも重要である．依存している数を元にして重大度を決定すると，データベースの重大度を高くすることができる．しかし，今度はフロントエンドの重大度が低くなる．そのため，監視の条件に合わせて重大度を依存されいる数で数えるか依存している数で数えるかを判断できるようにする必要がある．例えば，レイテンシは多くのサービスに影響があるため依存されている数をもとに決定する．対してサービスが一瞬でも停止していないかなど，それ自体が動作しているかを重視する監視項目は依存している数をもとにして重大度を決定できる．

本提案では依存関係があるかどうかを基準にして重大度を決定している．しかし．実際にはトラフィックが発生する頻度も考慮する必要がある．例えば 1 日 1 回流れる通信経路は，1 年に 1 度しか流れない通信経路よりも依存が強いと言える．そのため依存の強い経路を含むサービスの重大度は高くなる必要がある．依存の強弱を重大度に反映するため，依存数の計算を秒間リクエスト数を指標に決定する．

評価では実際にアラートの出力を行った．今回の実験では，重大度が 4 と 3 のアラートしか発生しなかった．これは，監視項目がレイテンシであり，実験の方法が負荷試験であったからと考えられる．本提案ではアラートの条件を 1 度流したトラフィックの 2 倍に指定している．しかし実際には定常状態を元にしてアラートの条件を指定する．またリクエストも 300 人のアクセスが常にかかるわけではない．よりユースケースに即した実験を行うため，実際に研究室のサーバで実際のトラフィックを使用した実験を行う．

7. おわりに

課題は，アラートに対してどの程度の重大度をつければいいか決めれないことである．そのため重大度が適切に設定されていないと，修正順序の誤りと順序決定の工数の増加が発生することをあげた．本稿では，サービスの依存数をもとにして，アラートの重大度を決定し重大度をもとにアラートを削減して出力する方法を提案した．その際，アラートを発生させるソフトウェアは独自で作成した．評価実験では，レイテンシに関する監視を 10 秒に 1 回行い，それを 100 回行った結果をアラートの件数と重大度ごとのアラートの割合のそれぞれで比較した．結果，重大度 3 のアラートが半数になることによって，アラートの件数が，22%削減され，また発生したアラートの 72%が重大度 4 と

なった.

参考文献

[1] Alshuqayran, N., Ali, N. and Evans, R.: A Systematic Mapping Study in Microservice Architecture, *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, pp. 44–51 (online), DOI: 10.1109/SOCA.2016.15 (2016).

[2] Hasselbring, W. and Steinacker, G.: Microservice Architectures for Scalability, Agility and Reliability in E-Commerce, *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, pp. 243–246 (online), DOI: 10.1109/ICSAW.2017.11 (2017).

[3] Wolff, E.: *Microservices: flexible software architecture*, Addison-Wesley Professional (2016).

[4] Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R. and Gil, S.: Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud, *2015 10th Computing Colombian Conference (10CCC)*, pp. 583–590 (online), DOI: 10.1109/ColumbianCC.2015.7333476 (2015).

[5] Soldani, J. and Brogi, A.: Anomaly Detection and Failure Root Cause Analysis in (Micro) Service-Based Cloud Applications: A Survey, *ACM Comput. Surv.*, Vol. 55, No. 3 (online), DOI: 10.1145/3501297 (2022).

[6] Hrusto, A., Engström, E. and Runeson, P.: Optimization of Anomaly Detection in a Microservice System through Continuous Feedback from Development, *Proceedings of the 10th IEEE/ACM International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems*, SESoS '22, New York, NY, USA, Association for Computing Machinery, p. 13–20 (online), DOI: 10.1145/3528229.3529382 (2022).

[7] Turnbull, J.: *Monitoring with Prometheus*, Turnbull Press (2018).

[8] bin Mohd Shuhaimi, M. A. A., binti Roslan, I., binti Anawar, S. et al.: The new services in Nagios: Network bandwidth utility, email notification and sms alert in improving the network performance, *2011 7th International Conference on Information Assurance and Security (IAS)*, IEEE, pp. 86–91 (2011).

[9] Sukhija, N., Bautista, E., James, O., Gens, D., Deng, S., Lam, Y., Quan, T. and Lalli, B.: Event Management and Monitoring Framework for HPC Environments Using ServiceNow and Prometheus, *Proceedings of the 12th International Conference on Management of Digital EcoSystems*, MEDES '20, New York, NY, USA, Association for Computing Machinery, p. 149–156 (online), DOI: 10.1145/3415958.3433046 (2020).

[10] Rahman, J. and Lama, P.: Predicting the End-to-End Tail Latency of Containerized Microservices in the Cloud, *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 200–210 (online), DOI: 10.1109/IC2E.2019.00034 (2019).

[11] Hrusto, A., Runeson, P. and Engström, E.: Closing the feedback loop in devops through autonomous monitors in operations, *SN Computer Science*, Vol. 2, No. 6, p. 447 (2021).

[12] Bautista, E., Sukhija, N. and Deng, S.: Shasta Log Aggregation, Monitoring and Alerting in HPC Environments with Grafana Loki and ServiceNow, *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 602–610 (online), DOI: 10.1109/CLUSTER51413.2022.00079 (2022).

[13] Harpe, S. E.: How to analyze Likert and other rating scale data, *Currents in Pharmacy Teaching and Learning*, Vol. 7, No. 6, pp. 836–850 (online), DOI: https://doi.org/10.1016/j.cptl.2015.08.001 (2015).

[14] Wu, Q., Yu, J., Lu, L., Qian, S. and Xue, G.: Dynamically Adjusting Scale of a Kubernetes Cluster under QoS Guarantee, *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, pp. 193–200 (online), DOI: 10.1109/ICPADS47876.2019.00037 (2019).