情報処理学会研究報告
IPSJ SIG Technical Report

イベントとリソース定義から作成した依存グラフを用いた連鎖障害の調査時間の短縮

小山 智之[1,2,a)]　串田 高幸[1,b)]　生野 壮一郎[1,c)]

概要：マイクロサービスアーキテクチャで構成される Web サービスで障害が発生すると，システム管理者は障害の原因調査を行う．障害の原因調査ではログやトレース，メトリクスを調査する．障害の 1 つに連鎖障害がある．連鎖障害では，あるマイクロサービスの障害にともない別のマイクロサービスで障害が発生する．連鎖障害の調査は，単一のマイクロサービスでの障害に比べ調査すべき対象のログやトレース，メトリクスが増えるため，障害の原因調査に時間がかかる．本稿では連鎖障害の原因調査にかかる時間を短縮するために，アプリケーションから出力されるトレースと分散コンピューティング基盤から取得したリソース定義をもとに依存グラフを作成する．依存グラフと分散コンピューティング基盤から取得したイベントからイベントの件数を依存グラフのエンドツーエンドのパスごとに集計する．スコアを計算し，スコアの高いリソースを障害原因の候補リストとして出力する．障害原因の候補リストを提案ソフトウェアが出力することで，手動での原因調査にかかる時間の短縮を実現する．マイクロサービスで構成される論文検索の Web サービスを対象に 2 種類の障害を再現した．1 つ目は Kubernetes クラスタのノードが応答しない障害である．2 つ目は OOM kill に伴うボリューム割り当て時のリソース競合である．評価対象には提案手法とベースライン手法として MicroRCA を使用する．評価指標には，障害原因の候補リストの内容を評価するために MRR と Hits@k を使用する．また，オーバーヘッドを評価するために提案手法のソフトウェアの実行時間を計測する．提案手法の MRR はベースライン手法に比べて平均で 28.0%高い結果を示した．提案手法の Hits@k はベースライン手法に比べて k=5 において平均で 0.88 高い結果を示した．提案手法のソフトウェアの実行時間はベースライン手法に比べて平均で 2.18 秒短い結果を示した．

キーワード：システム障害，根本原因の分析，障害箇所の特定，連鎖障害

1. はじめに

クラウド・分散システム研究室 (以降，CDSL とする) では論文検索の Web サービスである Doktor を運用している．ユーザは Doktor にアクセスし論文の検索やダウンロードを行う．Doktor はマイクロサービスアーキテクチャで設計されており，6 つのマイクロサービスから構成される．

2025 年 6 月 21 日に Doktor でシステム障害が発生した．図 1 に Doktor で発生したシステム障害の概要を示す．Kubernetes クラスタは Doktor のアプリケーションが動作するための分散コンピューティング基盤である．author MS は author マイクロサービスを表す．author マイクロサービスは Doktor を構成するマイクロサービスの 1 つである．アプリ Pod はアプリケーションサーバが動作する Pod[*1]である．MongoDB Pod はアプリケーションサーバがレコードを保存するためのデータベースサーバである．Persistent Volume は MongoDB Pod のレコードを保存するための Kubernetes の永続ストレージである．Rook Ceph[*2]は分散ストレージシステムである．OSD Pod は Rook Ceph を構成するソフトウェアの 1 つである．アプリ Pod や MongoDB Pod, OSD Pod からはログやトレース，メトリクスが収集され，監視サーバ (SigNoz)[*3]に保存される．監視サーバはログやトレース，メトリクスの保存と検索を提供するデータベースである．

Doktor で発生したシステム障害では，Kubernetes クラスタを構成する 1 台のノードが応答しなくなった．この問題に対処するために，当該ノードを手動で再起動した．再起動を行った後に OSD Pod が CrashLoopBackOff の

[1] 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
[2] 株式会社メルカリ
[a)] d212400159@edu.teu.ac.jp
[b)] kushida@acm.org
[c)] ikuno@stf.teu.ac.jp

[*1] https://kubernetes.io/ja/docs/concepts/workloads/pods/
[*2] https://rook.io/
[*3] https://signoz.io/

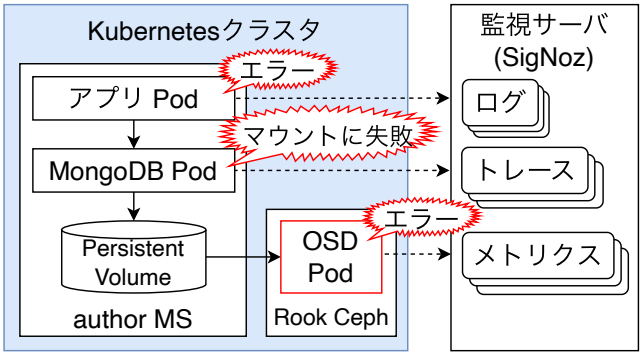

図 1 Doktor で発生したシステム障害の概要

表 1 論文をもとに作成した連鎖障害での連鎖パターンの上位 3 件

順位	連鎖元	連鎖先	件数
1	ストレージ	アプリケーション	73
2	ネットワーク	アプリケーション	65
3	ミドルウェア	アプリケーション	45

状態になった．CrashLoopBackOff は Kubernetes の Pod 内のコンテナが異常終了し，コンテナの再起動が繰り返される状態である．OSD Pod が CrashLoopBackOff になると，MongoDB Pod は OSD Pod を参照する Persistent Volume のマウントに失敗した．その結果，MongoDB Pod が Pending 状態になり，起動に失敗した．アプリ Pod が MongoDB Pod へアクセスを試みると，MongoDB Pod が起動していないため，アクセスに失敗しデータベース接続のエラーが発生した．一連の過程では，OSD Pod でのエラーが MongoDB Pod の Pending 状態を発生させ，最終的にアプリ Pod のデータベース接続のエラーを発生させた．

一箇所のエラーが別の箇所のエラーを発生させる障害は連鎖障害と呼ばれる．ある実証研究によると全体の障害のうち 52%以上は，障害が最初に観測された箇所と根本原因の箇所が異なる連鎖障害であった [1]．連鎖障害の代表的なパターンは，ミドルウェアやインフラストラクチャからアプリケーションへの連鎖である [2]．Microsoft 社での調査によるとミドルウェアとインフラストラクチャの障害は全体の 26.3%であった [3]．表 1 に論文をもとに作成した連鎖障害での連鎖パターンの上位 3 件を示す [4]．連鎖障害は全体で 460 件あり，そのうち約 15.9%の 73 件がストレージからアプリケーションへの連鎖障害である．約 14.1%にあたる 65 件がネットワークからアプリケーションへの連鎖障害である．約 9.8%にあたる 45 件がミドルウェアからアプリケーションへの連鎖障害である．クラウドコンピューティングの台頭によりストレージやネットワークは，ハードウェア製品からソフトウェアに置き換えられている．ストレージやネットワークはミドルウェアとして実装され動作しており，ミドルウェアからアプリケーションへの連鎖障害の原因調査が必要とされている．

課題

手動での原因調査はシステムを構成するコンポーネント数が増えるほどに，時間がかかる作業である．特にマイクロサービスアーキテクチャに代表される分散型のアーキテクチャでは，システムを構成するソフトウェアコンポーネントの数が増えるほど，システム全体の故障箇所も増える．コンポーネントの数が増えるほど，障害は複数のコンポーネントが関わり複雑になる [5]．Alibaba 社では 20,000 を超えるマイクロサービスが動作しており，7 日間で 100 億のトレースが収集されている [6]．システムが大規模になるほど，システム管理者による手動での調査では，大量のログやトレース，メトリクスを調査する必要がある．

障害の原因箇所の調査は手動で行われている．特にどのメトリクスやログ，トレースを見るべきかの判断は，運用の知識や経験が必要であり属人化しやすい．熟練したシステム管理者は短い時間で行えるが，経験の浅いシステム管理者は時間がかかる．調査にかかる時間は復旧時間に含まれている．復旧時間を短縮するために，障害の原因調査の自動化が必要である．

各章の概要

2 章では関連する障害の原因調査の手法を紹介する．3 章では提案手法を説明する．4 章では提案手法の評価を行う．5 章では提案手法に関する議論を行う．6 章では本稿のまとめを行う．

2. 関連研究

障害の原因調査はシステム管理者にとって時間のかかるタスクであるため，自動化の手法が提案されてきた．自動化の手法はシングルモーダル型とマルチモーダル型に大別される．シングルモーダル型の手法では，単一のデータソースを使い障害の原因調査を行う．メトリクスを使った手法では，メトリクスから障害の原因調査を行っている [7,8]．トレースを使った手法では，トレースから障害の原因調査を行っている [9,10]．ログを使った手法では，ログから障害の原因調査を行っている [11,12]．ミドルウェアが原因の問題ではミドルウェアとアプリケーションの両方をログやメトリクス，トレースを組み合わせて調査する必要がある．一方で，シングルモーダル型の手法では調査に単一のデータソースを使用するため正確さに課題がある．

マルチモーダル型の障害の原因調査が提案されている [13–16]．マルチモーダル型の手法では，複数のデータソースを使い障害の原因調査を行う．DeepTraLog はログとトレースを使い深層学習での異常検知を提案している [13]．この手法ではアプリケーションの依存関係を作成するためにログとトレースを取得している．この手法では異常検知に着目しており，障害の原因調査に改善の余地がある．Eadro では教師ありの機械学習での障害の原因調査

を提案している [14]．この手法では教師なしの機械学習に比べて高い精度を実現している．機械学習を使う場合には訓練データへのラベル付けが必要であり，システム管理者にラベル付けを行う作業は負担であり，負担を減らす方法の構築に改善の余地が残されている．Nezha はトレース，ログ，メトリクスを使いリクエスト単位の可観測性を向上した [15]．この手法ではトレースからイベントを構築しグラフを構築している．ミドルウェアで発生した障害がアプリケーションの障害に連鎖する場合には，障害の原因調査の対象がアプリケーションに限られる．障害の原因調査のために，過去の障害から単純なイベントの因果グラフを算出し，重みを計算する手法がある [16]．この手法ではデータセットに含まれる過去の障害を使用している．過去の障害は再発防止策が講じられており，再現する可能性は低いため適用範囲が限られる．

Pivot Tracing では動的なテレメトリの収集を提案している [17]．この手法では Java の instrument ライブラリを使い Hadoop クラスタのテレメトリを収集している．ミドルウェアの実装に使われるプログラミング言語のランタイムの実装によっては，ミドルウェアのソースコードの修正なしにテレメトリが収集できない．例えば，Go 言語ではランタイムに対する自動計測ができないため OpenTelemetry では eBPF を使った手法が提案されている[*4]．Pivot Tracing はミドルウェアの実装で使われているプログラミング言語に依存しており，適用可能なプログラミング言語が限られている．

3. 提案手法

本稿では障害の原因分析の手法である EventRCA を提案する．EventRCA はトレース，リソース定義，イベントからミドルウェアの連鎖障害の原因を求める．EventRCA は Kubernetes リソースに対応するトレースとリソース定義から依存グラフを作成する．依存グラフをもとにイベント数を集計し，スコアの高い順に連鎖障害の原因として出力する．本稿では連鎖障害の原因の候補の算出を目指す．主な貢献は障害の原因調査を手動での調査なく実現することである．

図 2 に提案手法の概要を示す．Kubernetes クラスタはアプリケーションが動作する分散コンピューティング基盤である．マイクロサービス群ではマイクロサービスがコンテナとして動作している．コントロールプレーンは Kubernetes クラスタを管理する役割をもつソフトウェアコンポーネントの総称である．トレースはマイクロサービスの内部で動作するアプリケーションから出力されるマイクロサービス同士の通信を記録したデータである．リソース定義は Kubernetes クラスタで動作するリソースの仕様

[*4] https://github.com/open-telemetry/opentelemetry-go-instrumentation

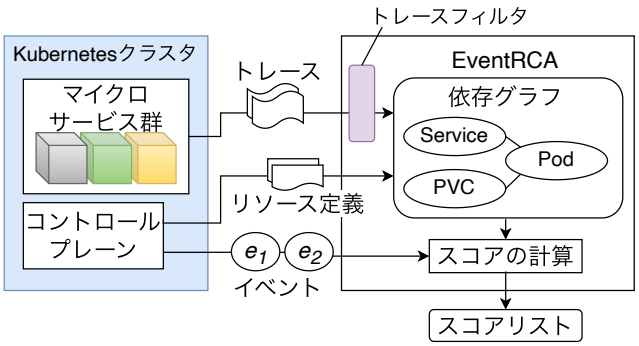

図 2 提案手法の概要

を定義したデータである．イベントは Kubernetes クラスタで動作するリソースへの変更を記録したデータである．EventRCA は提案手法であり，入力としてトレース，リソース定義，イベントを受け取る．トレースフィルタではトレースからエラーを含むものだけを EventRCA の入力として扱い，それ以外は破棄する．EventRCA ではトレースとリソース定義から依存グラフを構築する．依存グラフはトレースごとに作成され，トレースごとに異なる依存グラフが作成される．依存グラフとイベントからスコアを計算し，スコアリストを出力する．

既存の研究では障害の原因調査の手法を提案した [18]．この論文では提案にとどまっており，実際のシステム障害において手法が効果的であるかの評価が残されている．また，出力されるスコアリストのスコアが 0 に近い値に偏るため，スコアの分布に改善の余地が残されていた．さらに，トレースの収集でエラーを含まないトレースが取得される場合があり，スコアリストの結果の低下を引き起こしていた．本稿では，実際に発生した障害をもとに評価シナリオを作成した．これに加えてスコアの計算で正規化を追加しスコアの偏りの改善を提案した．また，トレースを選択するためにトレースフィルタを追加した．

依存グラフの作成

図 3 に依存グラフの作成方法を示す．マイクロサービス群は Kubernetes クラスタで動作する複数のマイクロサービスを表す．トレースフィルタは障害に関連したトレースを絞り込む役割をもつ．トレースフィルタではトレースにエラーが含まれる場合にトレースを取得する．エラーが含まれない場合にはトレースを破棄する．トレースは Kubernetes クラスタで動作するアプリケーションから収集される．トレースには個別のマイクロサービスでの処理にかかった時間やマイクロサービス間の呼び出し関係が記録されている．トレースは個別のマイクロサービスでのリクエスト処理を記録したスパンで構成される．スパンにはリクエストの処理にかかった時間や HTTP リクエストのパス，HTTP リクエストのメソッド，マイクロサービスが配置された Pod 名が含まれる．リソース定義は Kubernetes

クラスタから収集された Kubernetes クラスタで動作するリソースの仕様を定義したデータである．リソース定義は YAML 形式で記述されており，リソースの種類を表す kind パラメータやリソースの仕様を表す spec パラメータ，メタ情報を表す metadata パラメータを含む．依存グラフは Kubernetes クラスタで動作するリソースの依存関係を表す．例えば，Pod リソースを公開するために Service リソースが必要な場合，Pod リソースと Service リソースの間に依存関係がある．プログラム 1 に依存グラフの例を示す．各行はキーバリューペアで構成されており，キーは “from” または “to” であり，値は依存関係のあるリソースのネームスペース，リソースの種類，リソース名をスラッシュで結合した値が含まれる．2 行目の “from” パラメータは依存グラフのノードの一方を表す．2 行目はネームスペースが front であり，リソースの種類が Pod であり，リソース名が front-app-deploy-6b4dbf8d84-rbwvx であることを表す．3 行目の “to” パラメータは依存グラフのノードの他方を表す．3 行目はネームスペースが front であり，リソースの種類が ConfigMap であり，リソース名が istio-ca-root-cert であることを表す．依存グラフの作成の流れを以下に示す．

(1) マイクロサービスからトレースフィルタを介してトレースを取得する．
(2) トレースに含まれる各スパンから対応するリソース名を取得する．図 3 ではスパン 1 から Pod1 が得られる．スパン 2 から Pod2 が得られ，スパン 3 から Pod3 が得られる．
(3) Kubernetes クラスタのコントロールプレーンからリソース名に対応するリソース定義を取得する．図 3 では Pod1, Pod2, Pod3 のそれぞれに対応するリソース定義を取得する．
(4) 取得したリソース定義のパラメータを解析し，Kubernetes クラスタから依存関係のあるリソースを取得する．図 3 では Service リソースの “app: author” と Pod リソースの “app: author” が一致しており，この 2 つのリソース間に依存関係がある．
(5) 取得した依存関係から依存グラフを作成する．依存グラフは無向グラフであり，各ノードにはリソースの存在するネームスペース，リソースの種類，リソース名が含まれる．

スコアの計算

スコアの計算では入力としてイベントを受け取り，スコアリストを出力する．プログラム 2 にイベントの例を示す．このイベントは Kubernetes クラスタから収集されたものである．イベントは Type, Reason, Age, From, Message の属性をもつ．Type 属性はイベントの状態を表す．Reason 属性はイベントの種類を表す．Age 属性はイベントが発生した時刻からの経過時間を表す．From 属性はイベントを発生

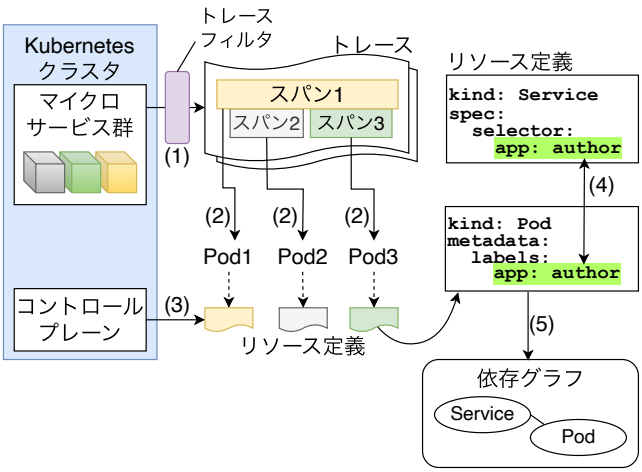

図 3 依存グラフの作成方法

プログラム 1 依存グラフの例

```
1 {
2   "from":"front/Pod/front-app-deploy-6
    b4dbf8d84-rbwvx",
3   "to":"front/ConfigMap/istio-ca-root-cert"
4 }
```

プログラム 2 イベントの例

```
1 Type Reason Age From Message
2 ---- ------ ---- ---- -------
3 Normal Scheduled 69s default-scheduler
  Successfully assigned default/ubuntu to
  clematis-worker2
```

プログラム 3 スコアリストの例

```
1 0.0495 /StorageProvisioner/rook-ceph.rbd.csi.
  ceph.com
```

させたリソースを表す．Message 属性はイベントの詳細を表す．プログラム 2 のイベントは，Pod (default/ubuntu) が 69 秒前に Kubernetes ノード clematis-worker2 にスケジュールされたことを表す．

スコアリストの例をプログラム 3 に示す．スコアリストは各行にスコアとリソース名が含まれている．スコアはその値が高いほど，障害の原因箇所の可能性が高いことを表す．リソース名は Kubernetes クラスタ上に存在するリソースを表す．リソース名はネームスペース，リソースの種類，リソースの名称がスラッシュで結合されている．プログラム 3 ではスコアが 0.0495 である．ネームスペースはなしであり，リソースの種類は StorageProvisioner であり，リソースの名称は rook-ceph.rbd.csi.ceph.com である．

アルゴリズム 1 にスコアの計算を示す．入力は依存関係を表すリスト形式の dependencyList であり，グラフの一方と他方のペアがリスト形式で含まれている．出力はスコアを表すリスト形式の result であり，パスとスコアのペアがリスト

形式で含まれている．1 行目は GET_PATHS 関数を呼び出し，依存関係リストからパスを取得する．パスはグラフの一方と他方のペアからなる．例えば，プログラム 1 の依存グラフのパスは “(front/Pod/front-app-deploy-6b4dbf8d84-rbwvx, front/ConfigMap/istio-ca-root-cert)” である．2 行目は *counts* を空リストで初期化する．*counts* は各パスに対するイベント数を表す．3 行目から 9 行目は各パスに対するイベント数を計算する．4 行目は *count* を 0 で初期化する．*count* は現在のパスに対するイベント数を表す．5 行目から 7 行目は現在のパスに対する各リソースのイベント数を計算する．6 行目は GET_EVENTS 関数を呼び出し，リソースのイベント数を Kubernetes API から取得し，*count* に更新する．8 行目は *counts* にエンドツーエンドのパスとそのイベント数のペアを追加する．例えば，*counts* が空でエンドツーエンドのパスが “(front/Pod/front-app-deploy-6b4dbf8d84-rbwvx, front/ConfigMap/istio-ca-root-cert)” で，イベント数が 10 の場合には，*counts* の変数は “((front/Pod/front-app-deploy-6b4dbf8d84-rbwvx, front/ConfigMap/istio-ca-root-cert), 10)” に更新される．10 行目から 12 行目はイベントの件数の最大と最小を計算し，その差分を求めている．これは出力結果を 0 から 1 までの範囲に正規化するために使用する．13 行目は *result* を空リストで初期化する．*result* はエンドツーエンドのパスとスコアのペアを表す．14 行目から 21 行目は各パスに対するスコアを計算し，パスとスコアのペアを *result* に追加する．15 行目から 16 行目では 12 行目の差分が 0 より大きい場合に *score* を 0 から 1 までの範囲に正規化している．17 行目から 19 行目では 12 行目の差分が 0 以下の場合に *score* を 1 に設定する．20 行目は *result* にエンドツーエンドのパスの最後のリソースとスコアのペアを追加する．22 行目は *result* をスコアの降順でソートして返す．

アルゴリズム 1 スコアの計算

Input: *dependencyList*: list of (src, dest) pairs
Output: *result*: list of (path, score) pairs
1: $paths \leftarrow$ GET_PATHS(*dependencyList*)
2: $counts \leftarrow []$
3: **for** each $path \in paths$ **do**
4: $\quad count \leftarrow 0$
5: $\quad$ **for** each $resource \in path$ **do**
6: $\quad\quad count \leftarrow count +$ GET_EVENTS($resource$)
7: $\quad$ **end for**
8: $\quad counts \leftarrow counts \cup (path, count)$
9: **end for**
10: $max\_val \leftarrow \max(\{count \mid (path, count) \in counts\})$
11: $min\_val \leftarrow \min(\{count \mid (path, count) \in counts\})$
12: $diff \leftarrow max\_val - min\_val$
13: $result \leftarrow []$
14: **for** each $(path, count) \in counts$ **do**
15: $\quad$ **if** $diff > 0$ **then**
16: $\quad\quad score \leftarrow (count - min\_val)/diff$
17: $\quad$ **else**
18: $\quad\quad score \leftarrow 1.0$
19: $\quad$ **end if**
20: $\quad result \leftarrow result \cup$ (LAST($path$), $score$)
21: **end for**
22: **return** *result* sorted by *score* in descending order

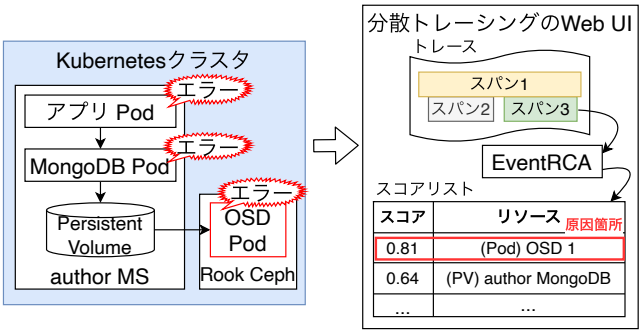

図 4 ユースケース・シナリオ

ユースケース・シナリオ

提案手法を分散トレーシングの Web UI へ統合することで，システム管理者が手動で障害の原因調査をする作業を削減する．図 4 にユースケース・シナリオを示す．Kubernetes クラスタは分散コンピューティング基盤である．アプリ Pod はアプリケーションの動作する Pod である．MongoDB Pod はデータベースの動作する Pod である．Persistent Volume は MongoDB Pod のデータを保存するための永続ストレージである．OSD Pod は分散ストレージを提供する Rook Ceph を構成するソフトウェアの 1 つである．分散トレーシングの Web UI では 1 件のトレースを確認する機能がある．この機能の中に提案手法の EventRCA を組み込むことで，システム管理者はスコアリストの中から障害の原因候補を探せる．スコアリストの提示によりシステム管理者が手動で障害の原因調査をする作業を削減する．

4. 評価

評価指標・評価方法

提案手法から出力されたスコアリストの内容を評価するために，MRR と Hits@k を計測する．MRR は Mean Reciprocal Rank の略であり，検索結果の評価に使用される指標である．MRR は関連研究でも使用されている主要な指標である [19–21]．MRR はスコアリストで障害の原因リソースが出現した順位とスコアリストの個数を入力として式 1 で計算される．N はスコアリストの個数であり，rank_i はスコアリスト i で障害の原因リソースが出現した順位である．

$$\mathrm{MRR} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{\mathrm{rank}_i} \tag{1}$$

Hits@k は検索結果の上位 k 件の中に正解が含まれているかを表す指標である．386 人のエンジニアへのアンケー

トによると，障害の原因調査において重視する結果は上位5件であった [22]．そのため，k の範囲は 1 以上かつ 5 以下の整数とした．提案手法を実行した際のオーバーヘッドを計測するために，実行時間を計測した．実行時間はプログラムの実行を開始してからプログラムの実行が終了するまでの時間である．

図 5 に評価方法を示す．障害シナリオはインターネットから収集した障害や実際に発生した障害を想定したシナリオである．根本原因は障害シナリオから取得したシステム障害の根本原因のリソース名である．根本原因を正解として以降の評価で使用する．Kubernetes クラスタは分散コンピューティング基盤であり Doktor のアプリケーションが動作する．SigNoz は監視サーバでありログやトレース，メトリクスの保存や検索を実現する．EventRCA は提案手法であり障害原因のリソース名を含むスコアリストを出力する．Evaluator は評価を行うソフトウェアであり，EventRCA の出力と根本原因のリソース名を比較して，MRR と Hits@k を計算する．一連の動作の流れを以下に示す．

(1) 障害シナリオから根本原因を取得する．
(2) Kubernetes クラスタに障害シナリオをもとに障害を発生させる．
(3) Kubernetes クラスタからログやトレース，メトリクスを取得し，SigNoz に保存する．
(4) EventRCA は SigNoz からトレースを取得する．
(5) EventRCA は Kubernetes クラスタからリソース定義とイベントを取得する．
(6) Evaluator は EventRCA のスコアリストを読み込む．
(7) Evaluator は根本原因からリソース名を取得する．
(8) Evaluator は入力をもとに MRR と Hits@k を計算し出力する．

障害シナリオ

実際に発生したシステム障害をもとに障害シナリオを作成した．シナリオ 1 は Rook Ceph の OSD Pod のエラーである．Rook Ceph は分散ストレージを提供するソフトウェアであり，Kubernetes クラスタで動作する．Kubernetes クラスタの 1 台のノードが応答しなくなり，再起動が必要になった．OSD Pod で再起動の際に正常に終了しないため，データの破損が発生した．このシナリオでは Kubernetes クラスタを構成する 1 ノードの電源を切ることで，障害を再現する．

シナリオ 2 は Doktor の開発環境の Rook Ceph で 2025 年 11 月 21 日に実際に発生した障害である．Kubernetes クラスタを構成する 1 台のワーカーノードが高いメモリ使用率になり OOM kill が発生した．OOM kill の発生にともない paper マイクロサービスでオブジェクトストレージの MinIO の障害が発生した．MinIO の Pod は OOM kill が発生すると，別の Kubernetes ノードで再起動した．複数のボリューム割り当てが発生したため，再起動した MinIO の Pod はオブジェクトの読み取りや書き込みに失敗した．その結果，paper マイクロサービスの論文ダウンロードの機能で障害が発生した．

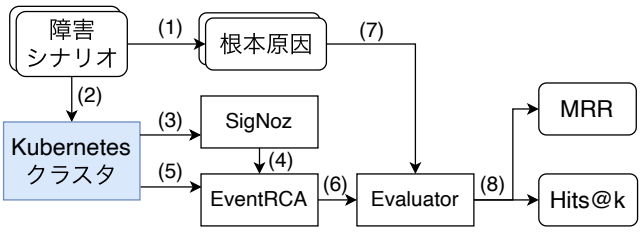

図 5 評価方法

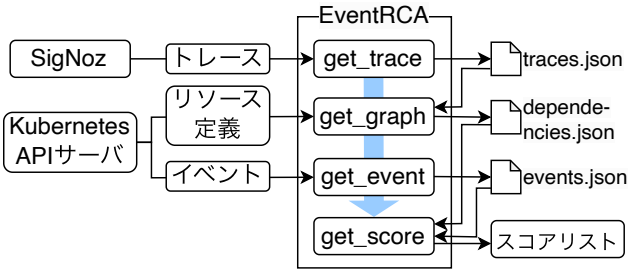

図 6 提案ソフトウェアの実装

実験環境・実装

実験にはマイクロサービスアーキテクチャで設計された Web アプリケーションである Doktor を使用した．Doktor は stats，front，paper，fulltext，author，thumbnail の 6 つのマイクロサービスから構成される．これらのマイクロサービスはコンテナとして Kubernetes クラスタ上で動作する．それぞれのマイクロサービスのコンテナには OpenTelemetry SDK が導入されておりログやトレース，メトリクスが収集されている．Kubernetes クラスタは分散コンピューティングの環境であり，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 クラスタのノードは VMware ESXi 8.0 に作成された仮想マシンである．それぞれの仮想マシンは vCPU 8 コア，メモリ 8GB，ディスク 40GB である．Kubernetes のエンジンには K3s v1.31.5 を使用した．ストレージとして Rook Ceph v1.13.6 を使用した．監視サーバには SigNoz v0.88.1 を使用した[*5]．SigNoz はログやトレース，メトリクスの保存や検索を提供する監視サーバである．

図 6 に提案ソフトウェアの実装を示す．提案ソフトウェアの EventRCA は 4 つの関数で構成される．関数 get_trace は SigNoz からトレースを読み込み，traces.json に保存する．関数 get_graph は Kubernetes API サーバからリソース定義を読み込み，traces.json からトレースを読み込む．

[*5] https://signoz.io/

その後，依存グラフを作成し，dependencies.json に保存する．関数 get_event は Kubernetes API サーバからイベントを読み込み，events.json に保存する．関数 get_score は dependencies.json と events.json から依存グラフとイベントを読み込む．その後，依存グラフからスコアを計算し，スコアリストを出力する．

評価結果

図 7 に実行時間の比較を示す．X 軸は手法を表し，Y 軸は実行時間 (秒) を表す．MicroRCA はベースライン手法であり，代表的な原因調査の手法である [7]．それぞれの値は 10 回計測した平均を使用した．EventRCA の実行時間は約 4.56 秒であり，MicroRCA の実行時間は約 6.74 秒であった．EventRCA の実行時間は MicroRCA に比べて約 2.18 秒短く，実行時のオーバーヘッドがないことを示している．

図 8 に MRR の比較を示す．X 軸は障害シナリオを表し，Y 軸は MRR を表す．凡例は手法を表す．MicroRCA はベースライン手法である．EventRCA は提案手法である．シナリオ 1(s1) では MicroRCA の MRR は 0.00 であり，EventRCA の MRR は 0.26 であった．この結果は EventRCA が 4 回の実験で原因箇所を 3 位，3 位，5 位，6 位で出力したため，$(0.333 + 0.333 + 0.2 + 0.166)/4 \approx 0.26$ より 0.26 となった．EventRCA の MRR は MicroRCA に比べて 0.26 高い結果を示している．シナリオ 2(s2) では MicroRCA の MRR は 0.00 であり，EventRCA の MRR は 0.30 であった．EventRCA の MRR は MicroRCA に比べて 0.30 高い結果を示している．EventRCA の MRR は，MicroRCA の MRR に比べて平均で 28.0%高かった．

図 9 に Hits@k の比較を示す．X 軸はトップ k 件の結果を表し，Y 軸は Hits@k を表す．凡例は手法と障害シナリオの組み合わせを表す．MicroRCA はベースライン手法である．EventRCA は提案手法である．シナリオ 1(s1) とシナリオ 2(s2) のいずれでも EventRCA が上位 5 件の候補リストに障害の原因を含むことを示した．一方で MicroRCA は，上位 5 件の候補リストに障害の原因を含まなかった．MicroRCA はアプリケーションに関して CPU 使用率をはじめとしたメトリクスを解析するため，アプリケーションとミドルウェアの依存関係の解析がされておらず，ミドルウェアが原因の連鎖障害が検出されなかった．EventRCA の平均の Hits@5 は $(1.00 + 0.75)/2 \approx 0.88$ であり，MicroRCA の平均の Hits@5 に比べて 0.88 高い結果を示した．

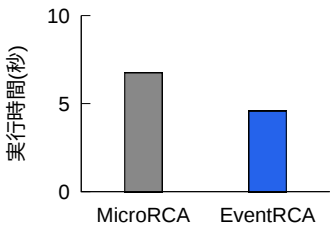

図 7 実行時間の比較

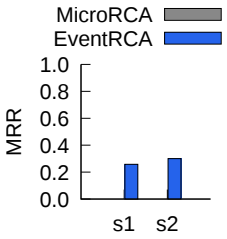

図 8 MRR の比較

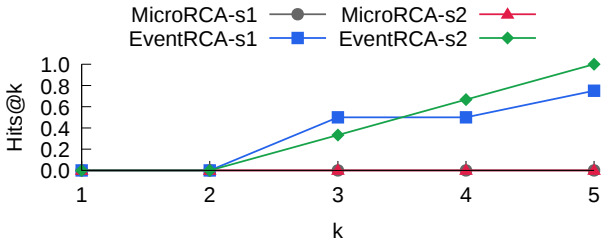

図 9 Hits@k の比較

5. 議論

提案手法では Kubernetes のリソースイベントを使って障害の原因調査を行った．リソースイベントには，リソースの作成や削除を含むリソースのライフサイクルの情報が含まれる．提案手法ではイベントの件数に着目したが，リソースの内部状態の解析は行っていない．ログにはリソースの内部状態が含まれているため，リソースイベントに比べてリソースの内部状態の解析を詳細に行える．商用システムではトラフィックが増えるにつれログの件数が増える．Tencent 社の商用システムでは 1 秒あたり数千万件のログが出力される [23]．数千万件のログの保存や検索には，ハードウェア資源が必要であるため，ログのサンプリングが行われる [24]．固定した割合でのサンプリングは調査に必要なログを破棄する場合がある．システムの異常度にもとづく動的なサンプリングは，調査に必要なログを保持するため，ログのサンプリングの問題を解決できる [25]．

6. おわりに

本稿では連鎖障害の原因調査を行う手法として EventRCA を提案した．EventRCA はリソース定義とトレースを使い依存グラフを作成し，リソースイベントと依存グラフから障害原因の候補を含むスコアリストを求めた．提案手法を評価するためにマイクロサービスで構成される論文検索の Web サービスを対象に実際に発生した障害を再現した．評価指標には，提案手法の出力の精度を計測するために MRR と Hits@k を使用した．また，提案手法のオーバーヘッドを評価するためにソフトウェアの実行時間を計測した．提案手法の MRR はベースライン手法に比べて平均で 28.0%高い結果を示した．評価実験の結果より提案手法の Hits@k は，k=5 においてベースライン手法に比べて平均で 0.88 高い結果を示した．提案手法の実行時間はベースライン手法に比べて平均で 2.18 秒短い結果であった．

謝辞 本研究は, JSPS 科研費 JP23K11073, JP23K11087 の助成を受けたものである．

参考文献

[1] Wang, Y., Li, G., Wang, Z., Kang, Y., Zhou, Y., Zhang, H., Gao, F., Sun, J., Yang, L., Lee, P. et al.: Fast outage analysis of large-scale production clouds with service correlation mining, *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, IEEE, pp. 885–896 (2021).

[2] Zhao, N., Chen, J., Yu, Z., Wang, H., Li, J., Qiu, B., Xu, H., Zhang, W., Sui, K. and Pei, D.: Identifying bad software changes via multimodal anomaly detection for online service systems, *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 527–539 (2021).

[3] Ghosh, S., Shetty, M., Bansal, C. and Nath, S.: How to fight production incidents? an empirical study on a large-scale cloud service, *Proceedings of the 13th Symposium on Cloud Computing*, pp. 126–141 (2022).

[4] Li, X., Yu, G., Chen, P., Chen, H. and Chen, Z.: Going through the life cycle of faults in clouds: Guidelines on fault handling, *2022 IEEE 33rd International Symposium on Software Reliability Engineering (ISSRE)*, IEEE, pp. 121–132 (2022).

[5] Zhou, X., Peng, X., Xie, T., Sun, J., Ji, C., Liu, D., Xiang, Q. and He, C.: Latent error prediction and fault localization for microservice applications by learning from system trace logs, *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pp. 683–694 (2019).

[6] Luo, S., Xu, H., Lu, C., Ye, K., Xu, G., Zhang, L., Ding, Y., He, J. and Xu, C.: Characterizing microservice dependency and performance: Alibaba trace analysis, *Proceedings of the ACM symposium on cloud computing*, pp. 412–426 (2021).

[7] Wu, L., Tordsson, J., Elmroth, E. and Kao, O.: Microrca: Root cause localization of performance issues in microservices, *IEEE/IFIP Network Operations and Management Symposium (NOMS)* (2020).

[8] Pham, L., Ha, H. and Zhang, H.: Baro: Robust root cause analysis for microservices via multivariate bayesian online change point detection, *Proceedings of the ACM on Software Engineering*, Vol. 1, No. FSE, pp. 2214–2237 (2024).

[9] Yu, G., Chen, P., Chen, H., Guan, Z., Huang, Z., Jing, L., Weng, T., S un, X. and Li, X.: Microrank: End-to-end latency issue localization with extended spectrum analysis in microservice environments, *Proceedings of the Web Conference 2021*, pp. 3087–3098 (2021).

[10] Li, Z., Chen, J., Jiao, R., Zhao, N., Wang, Z., Zhang, S., Wu, Y., Jiang, L. n., Yan, L., Wang, Z. et al.: Practical root cause localization for microservice systems via trace analysis, *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, IEEE, pp. 1–10 (2021).

[11] Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y. and Chen, X.: Log clustering based problem identification for online service systems, *Proceedings of the 38th international conference on software engineering companion*, pp. 102–111 (2016).

[12] He, S., Lin, Q., Lou, J.-G., Zhang, H., Lyu, M. R. and Zhang, D.: Identifying impactful service system problems via log analysis, *Proceedings of the 2018 26th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pp. 60–70 (2018).

[13] Zhang, C., Peng, X., Sha, C., Zhang, K., Fu, Z., Wu, X., Lin, Q. and Zhang, D.: Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning, *Proceedings of the 44th international conference on software engineering*, pp. 623–634 (2022).

[14] Lee, C., Yang, T., Chen, Z., Su, Y. and Lyu, M. R.: Eadro: An end-to-end troubleshooting framework for microservices on multi-source data, *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, IEEE, pp. 1750–1762 (2023).

[15] Yu, G., Chen, P., Li, Y., Chen, H., Li, X. and Zheng, Z.: Nezha: Interpretable fine-grained root causes analysis for microservices on multi-modal observability data, *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 553–565 (2023).

[16] Yao, Z., Pei, C., Chen, W., Wang, H., Su, L., Jiang, H., Xie, Z., Nie, X. and Pei, D.: Chain-of-event: Interpretable root cause analysis for microservices through automatically learning weighted event causal graph, *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pp. 50–61 (2024).

[17] Mace, J., Roelke, R. and Fonseca, R.: Pivot tracing: Dynamic causal monitoring for distributed systems, *ACM Transactions on Computer Systems (TOCS)*, Vol. 35, No. 4, pp. 1–28 (2018).

[18] 小山智之，串田高幸，生野壮一郎：Kubernetes のリソースイベントと依存グラフとトレースによる障害の原因調査にかかる時間の短縮，第 33 回マルチメディア通信と分散処理ワークショップ論文集，情報処理学会，pp. 268–275 (2025).

[19] Wang, D., Chen, Z., Ni, J., Tong, L., Wang, Z., Fu, Y. and Chen, H.: Interdependent causal networks for root cause localization, *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5051–5060 (2023).

[20] Zheng, L., Chen, Z., He, J. and Chen, H.: MULAN: multi-modal causal structure learning and root cause analysis for microservice systems, *Proceedings of the ACM Web Conference 2024*, pp. 4107–4116 (2024).

[21] Xie, Z., Zhang, S., Geng, Y., Zhang, Y., Ma, M., Nie, X., Yao, Z., Xu, L., Sun, Y., Li, W. et al.: Microservice root cause analysis with limited observability through intervention recognition in the latent space, *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 6049–6060 (2024).

[22] Kochhar, P. S., Xia, X., Lo, D. and Li, S.: Practitioners' expectations on automated fault localization, *Proceedings of the 25th international symposium on software testing and analysis*, pp. 165–176 (2016).

[23] Yu, M., Lin, Z., Sun, J., Zhou, R., Jiang, G., Huang, H. and Zhang, S.: TencentCLS: the cloud log service with high query performances, *Proceedings of the VLDB Endowment*, Vol. 15, No. 12, pp. 3472–3482 (2022).

[24] Li, B., Peng, X., Xiang, Q., Wang, H., Xie, T., Sun, J. and Liu, X.: Enjoy your observability: an industrial survey of microservice tracing and analysis, *Empirical Software Engineering*, Vol. 27, No. 1, p. 25 (2022).

[25] Meng, S. and Liu, L.: Enhanced monitoring-as-a-service for effective cloud management, *IEEE Transactions on Computers*, Vol. 62, No. 9, pp. 1705–1720 (2012).