Sock Shopにおける CPU 使用量の変化にもとづくスパイクアクセス時の原因特定

石川裕人[1]　疋田辰起[1]　串田 高幸[1]

概要：ソフトウェア開発のアーキテクチャの一種で，大規模なアプリケーションを小さな独立したサービスに分割する手法としてマイクロサービスというものがある．マイクロサービスで構成されている EC サイトでスパイクアクセスが発生した場合，冗長なエラーメッセージを受信することで管理者はエラーメッセージを無視や誤解をしてしまい，障害復旧に時間がかかり遅れが発生することが今回の課題である．本稿の提出手法では，マイクロサービスで構築するアプリケーションにおいて管理者の対応が遅れないようにするために各サービスの Pod の CPU 使用量の変化でエラーの原因を特定する．特定の基準として CPU 使用量が最新の CPU 使用量の平均値が直前の CPU 使用量の平均値より減少したかで判断する．基礎実験ではマイクロサービスで構成されているデモアプリケーションの Sock Shop を対象とし，アクセスするユーザ数によって各サービスの Pod の平均 CPU 使用量に変化があるか実験を行った．その結果，ユーザ数が増加すると CPU 使用量が減少した．基礎実験によって，提案手法が有効であることが確認できた．評価実験では，1 秒に 1 人ユーザを増加させて毎秒の最大ユーザ数を 600 で 10 分間負荷試験を行い，carts の Pod がエラーの原因として特定され，原因として特定された carts の Pod の CPU リソースをスケールアップして再度，負荷試験を行った．結果は，エラーが出なくなったため提案により原因の特定ができた．

1. はじめに

背景

マイクロサービスとは，サービス全体が独立した小さなサービスの集合で構成されるアプリケーション開発手法である [1]．マイクロサービスのデモアプリケーションとして Sock Shop[*1]がある．Sock Shop は，靴下販売サイトで EC サイトに必要な機能が実装されている．このソフトウェアはオープンソースとして公開されているため，実験や研究の対象として用いられる [2].

マイクロサービスは高可用性，優れた柔軟性，高速な応答性，拡張性の向上と複数の利点をもたらす．しかし，複数のサービスで構成されているため，依存関係が複雑という欠点がある [3]．障害が発生した場合には，複雑な依存関係があることから多数の異常が検出される．よって冗長なアラートが通知させる [4]．アラートは，オンラインサービスシステムの監視システムにおける重要なデータソースの一種であり，サービスコンポーネントの異常を記録し，管理者に報告するために使用される [5]．よって，各サービスに問題が発生した場合に，状況を素早く把握する必要がある．そのためにマイクロサービスでは監視を行う．

マイクロサービス化されたシステムの監視では，各マイクロサービスからログを収集する．システムにエラーが発生した場合，障害の原因を特定するためには，原因となるメトリックを重点的に考慮する必要がある．例としてリソースが不足している状態でエラーが発生した場合，マイクロサービスの CPU 使用率がエラーの原因を特定するのに役立つメトリックとなる [6].

障害の原因特定が遅れることで経済面において膨大な損失を引き起こしたり，ユーザの満足度を低下させてしまう．例として，米国の 63 ヶ所のデータセンターを対象に実施された調査によると，サービスのダウンタイムの平均コストは，2010 年の 50.5 万ドルから 2016 年には 74.0 万ドルへと着実に増加している [7]．また，2018 年に Amazon が行う年に一度の大規模のプライムデーというセールイベントがあり，1 時間のダウンタイムで最大 1 億ドルの損失となった [8].

課題

本稿における課題は，マイクロサービスにおいて冗長なエラーメッセージを受信することで管理者は異常がある場所の特定が困難ということである．課題のケースとしては，図 1 のように管理者はエラーメッセージを無視や誤解をし

[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

[*1] https://microservices-demo.github.io/

てしまい障害復旧に時間がかかり遅れが発生する．障害の発見が遅れたことでダウンしたネットワークやシステムに依存している企業にとって大きな損失となってしまう [9]．また，エラーメッセージはリクエストエラーとしか表示されないため余計に時間がかかってしまう．

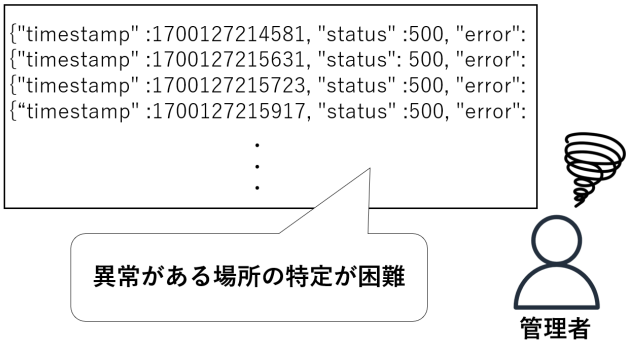

図 1　課題の概要図

各章の概要

本稿は以下のように構成される．第 1 章では，本稿の背景と課題について述べる．第 2 章では，本稿の関連研究について述べる．第 3 章では，本稿での課題を解決するための提案手法について述べる．第 4 章では，提案した手法の実装について述べる．第 5 章では，評価実験の内容とその分析について述べる．第 6 章では，提案手法の議論を述べる．第 7 章は，本稿のまとめである．

2. 関連研究

システムトレースログから学習することで，マイクロサービスアプリケーションの潜在的なエラー予測と障害位置特定を行う MEPFL を提案している研究がある [10]．MEPFL は，システムトレースログをもとにアプリケーションの実行とフォールトインジェクションから学習した予測モデルを使用して，実稼働環境でエラーと障害を予測する．この提案にトレースログが使用されているが，トレースログは高負荷の CPU などのリソース系の障害に対する特定精度が低下するという課題が挙げられる．

ログ解析にもとづくモデルには明らかに欠点があるということから FastTransLog が提案された研究がある [11]．この研究は前処理でログパーサーを使用する代わりに，生のログメッセージを直接使用する．ログメッセージから情報が失われることがないため，FastTransLog は，ログ解析に依存する他のアプローチと比較して，ログメッセージの内容を正確に学習できる．アルゴリズムの速度が大幅に向上するだけでなく，データセットの精度も最大 99 %向上している．しかし，この研究はログを使用して特定するため，エラーメッセージとして出力されたエラーにしか対応できないという課題が挙げられる．

システムログと監視メトリックを融合した障害位置特定を提案している研究がある [12]．この研究ではサービスの依存関係を検出し，システムログと監視メトリクスを利用して各マイクロサービスの異常を検出する．障害のあるサービスを特定し，サービスの依存関係と検出された異常にもとづいて原因となる場所のシステムメトリクスを推測する．実験結果として検出精度が約 75 %であることを示している．

サービス依存関係グラフ (SDG) の概念を導入してノード間の複雑な呼び出し関係を表現し，TraceVAE と ModelCoder で構成される TraceModel を提案している研究がある [13]．TraceModel は異常検出結果にもとづいて，障害の特徴を事前定義された障害モデルと比較することにより，平均 110 秒以内に異常を検出し，未知の障害の根本原因を特定することができる．この研究では事前に定義された障害モデルを用意する必要がある．

LightGBM 法にもとづくマイクロサービスの障害特定方法を提案している研究がある [14]．LightGBM 方式は，マイクロサービスの過去の操作情報を分析し，障害原因を学習特定し，障害特定に利用することができ，迅速に障害を特定し，マイクロサービスの高い可用性を確保することができる．この研究では，マイクロサービスの過去の操作情報で特定するため，新たな環境で利用することが難しい．

3. 提案

提案方式

本稿では，マイクロサービスで構築するアプリケーションにおいて，エラーの原因となる場所を各サービスの Pod の CPU 使用量の変化によって特定する．CPU は処理を超える負荷が発生した際，エラーを返すことしか処理が行われないため，CPU 使用量の増加が穏やかになる．

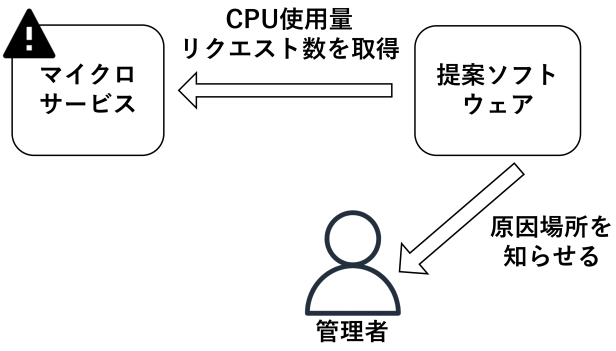

図 2　提案の概要

図 2 に提案の概要を示す．提案ソフトウェアによりマイクロサービスで構築するアプリケーション内にある各サービスの Pod の CPU 使用量とリクエスト数を取得する．アクセスするユーザ数は増加しているが CPU 使用量は減少

している Pod をリクエスト数と各 Pod の CPU 使用量で判断する．CPU 使用量は最新の平均値が直前の平均値より減少した Pod をエラーの原因とし管理者に通知する．原因特定の通知は slack[*2]から行われる．

基礎実験の環境

提案を証明するために基礎実験を行った．図 3 に基礎実験の環境を示す．基礎実験の環境として K3s 上にマイクロサービスのデモアプリケーションである Sock Shop を構築した．そして疑似的に大量のリクエストを送ることができる負荷試験ツールの Locust[*3]を用いて実験を行った．そして負荷試験が終わった後に各サービスの Pod の CPU 使用量を取得し，CSV ファイルに結果を出力した．

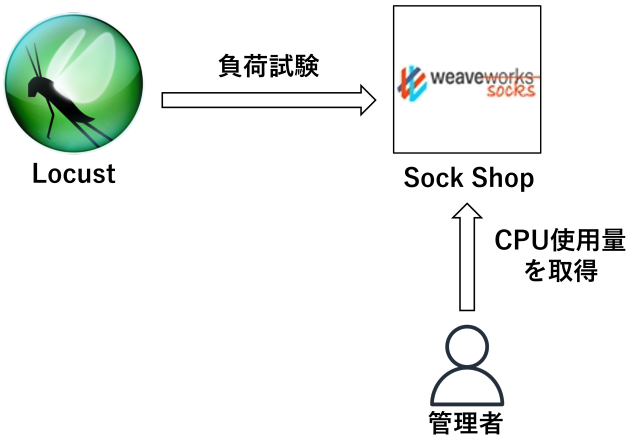

図 3 基礎実験の環境

基礎実験

基礎実験は Sock Shop に毎秒のアクセスユーザ数を 200，300，400，500，600，700 で指定し，負荷試験のシナリオとして Microservices Demo が提供しているシナリオ[*4]を使用した．アクセスするユーザ数によって各サービスの Pod の平均 CPU 使用量に変化があるか実験を行った．実験はそれぞれ 10 分間を 3 回ずつ行った．

基礎実験の結果

基礎実験を行った結果，図 4 に各ユーザ数の平均 CPU 使用量を示す．縦軸が CPU 使用量を表す．横軸はアクセスしたユーザ数を表す．carts はユーザ数 500 が最も高い CPU 使用量であり，600 から下がった．

catalogue，front-end，user のこの 3 つはユーザ数 400 が最も高い CPU 使用量であり，500 から下がった．

orders はユーザ数 600 が最も高い CPU 使用量であり，700 で下がった．結果として，始めはユーザ数と CPU 使用量は増加していたが，ある一定のユーザ数を超えると CPU 使用量は減少した．

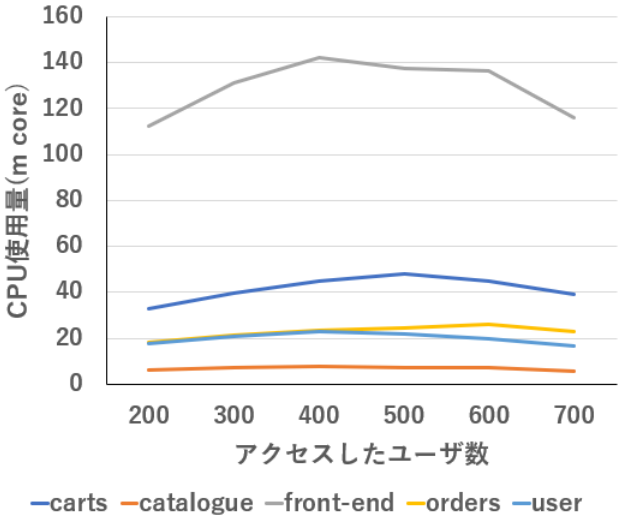

図 4 各ユーザ数の平均 CPU 使用量

ユースケース・シナリオ

本稿のユースケースとして EC サイトの Amazon を想定する．ユースケース・シナリオを図 5 に示す．タイムセールや有名商品の発売日でアクセススパイクが発生する．冗長なエラーメッセージを受信することで管理者は異常がある場所の特定が困難ということである．そのため，本稿で提案するソフトウェアにより CPU 使用量の変化とリクエスト数によってどこで問題が起きているかの原因を特定することができる．これにより，エラーの原因が見つかり管理者の対応がスムーズに行える．

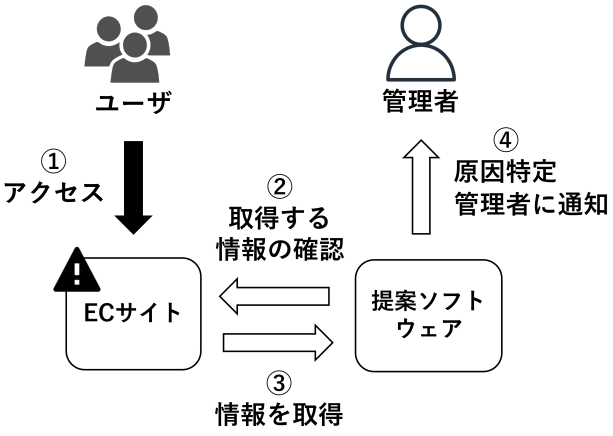

図 5 ユースケース・シナリオの概要

[*2] https://slack.com/intl/ja-jp/
[*3] https://locust.io/
[*4] https://github.com/microservices-demo/load-test

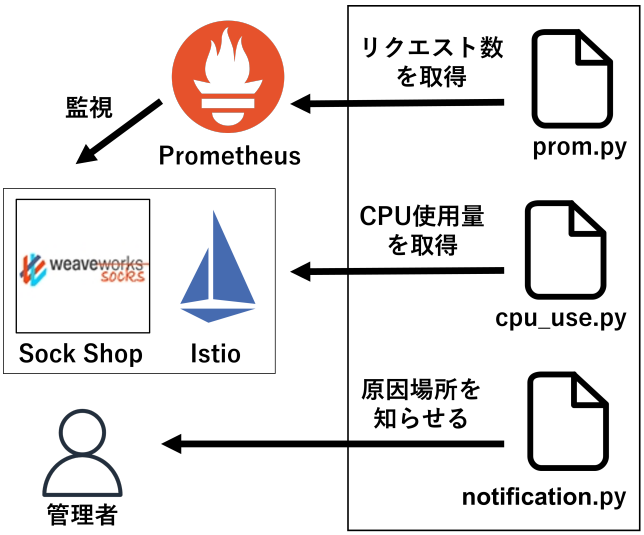

図 6 提案ソフトウェアの概要

4. 実装

提案ソフトウェアの概要を図 6 に示す．提案ソフトウェアは prom.py と cpu_use.py と notification.py の構成となっている．prom.py では，Prometheus のクエリからリクエスト数を取得する．cpu_use.py では，kubectl top コマンドにより各 Pod の Sock Shop の CPU 使用量を取得する．取得した Pod の CPU 使用量を 1 分間ごとの平均値を出す．最新の平均値が直前の平均値より減少したかを判定する．notification.py では，cpu_use.py によって原因場所と判定された Pod を管理者の slack に通知が送られる．

実装環境の構成は仮想環境に K3s を構築し，K3s 上にマイクロサービスのデモアプリケーションである Sock Shop を構築した．リクエスト数を取得するために Sock Shop に対してサービスメッシュの Isito [*5]を構築し，Istio の addon である Prometheus[*6] から提案ソフトウェアによってリクエスト数を取得している．

5. 評価実験

実験では，Sock Shop に Locust で負荷試験を行いスパイクアクセスを発生させる．1 秒に 1 人ユーザ数を増加させ，毎秒の最大ユーザ数を 600 で 10 分間行う．アクセスシナリオは基礎実験で行ったシナリオ[*4]から負荷試験を行い，本稿の課題としているエラーの原因が提案ソフトウェアによって特定できるかを評価する．シナリオの流れを説明する．ユーザは始めに Sock Shop のホームページにアクセスし，ログインページでユーザ登録を行う．次に商品カタログページからランダムに 1 つの商品をカートに追加し，配送先・支払い方法を設定して注文を確定する．

原因特定の基準として CPU 使用量が最新の CPU 使用量の平均値が直前の CPU 使用量の平均値より減少したかで判断する．

*5 https://istio.io/
*6 https://prometheus.io/

実験環境

図 7 は実験環境の流れを示す．実験環境は実装環境で構成したサービスメッシュである Istio を導入した Sock Shop に対して実験を行う．提案ソフトウェアによって CPU 使用量と Prometheus からリクエスト数を取得する．負荷試験は Locust を用いて実験を行う．Locust はマスターノード 1 つ，ワーカーノード 2 つで構築されている．実験の手順を以下に示す．

① Sock Shop に対して負荷試験を行いスパイクアクセスを発生させる．

② 提案ソフトウェアによりリクエスト数を取得し，各サービスの Pod の CPU 使用量の変化からエラーの原因を特定する．

③ エラーの原因として特定した Pod を管理者に通知される．

④ 特定したサービスの CPU リソースをスケールアップする．

再度，負荷試験を行いエラーが発生するかどうかを確認する．

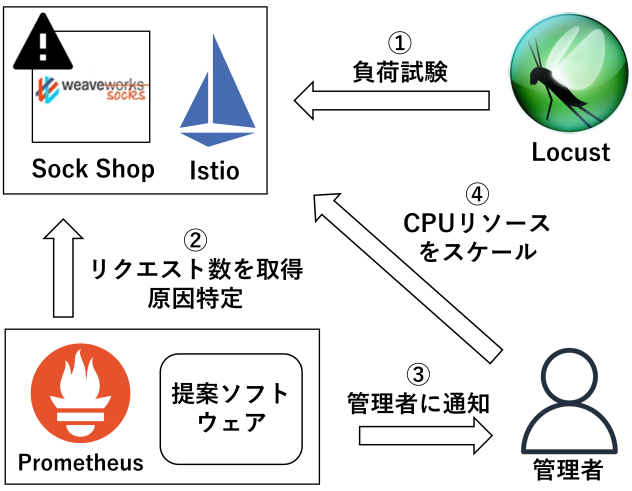

図 7 実験環境の流れ

実験結果と分析

図 8 に提案ソフトウェアによって slack に特定した Pod の通知を示す．10 分間で実験を行った結果，carts の CPU 使用量が減少し原因と通知された．図 9 は Prometheus で

取得したリクエスト数を示す．Timestamp は 1 分間ごとに取得しているため，60 ずつ上がっている．Value はリクエスト数を示しており，増加していることがわかる．この 2 つの図からリクエスト数は増加しているが，carts の CPU 使用量は減少していることがわかる．carts の CPU リソースをスケールアップして再度，実験を 2 回行った．結果は，図 10 にスケールアップ後の通知を示す．エラーと carts での原因の通知は発生しなかったが，oders と front-end の CPU 使用量が減少したと通知された．提案ソフトウェアによって carts の CUP リソースをスケールアップして，エラーが出なくなったため原因を特定することができた．

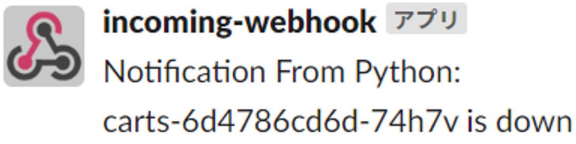

図 8 特定した Pod の通知

図 9 Prometheus で取得したリクエスト数

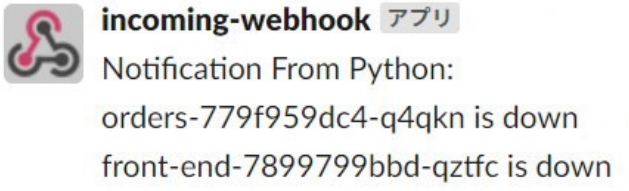

図 10 carts の CPU リソースをスケールアップ後の通知

6. 議論

本稿の提案は，各サービスの Pod の CPU 使用量の変化を分析することで，エラーの原因を特定することである．しかし，この提案の信頼性向上のために，CPU 使用量だけでなくアクセスログも同時に分析することが必要である．Sock Shop におけるアクセススパイクの実験では，CPU 使用量が減少した場合，サービスのアクセスログも確認する．具体的には，500 番以上のエラーコードを持つアクセスログも分析することでより詳細な異常の特定が可能である．

実験結果により，原因と通知された carts の CPU リソースをスケールアップし負荷試験を行い，carts でのエラーは発生しなかった．しかし carts のみにしか実験を行っていないため，他の Pod がエラーが発生した時にこの提案手法は有効なのか実験する必要がある．また，冗長エラーメッセージが出た時に原因となる Pod を特定するものだが，エラーが発生していない場合にも Pod の CPU 使用量が減少したと通知された．改善としてエラーが発生しなかったが，CPU 使用量が減少することは原因があるので管理者に注意を通知することでエラーが発生した時に速やかに対処が可能である．

CPU 使用量が減少したことに対して，自動的に CPU リソースをスケーリングする機能を提案ソフトウェアに統合することにより，管理者が手動でスケーリングを行う手間が省かれ，負担が軽減される．

7. おわりに

マイクロサービスのアプリケーションでアクセススパイクが起きた際に管理者は異常がある場所の特定が困難であることが本稿の課題である．基礎実験では，始めはユーザ数と CPU 使用量は増加していたが，ある一定のユーザ数を超えると CPU 使用量は減少した．そのため，エラーの原因となる各サービスの Pod を CPU 使用量の変化によって特定することを提案とした．評価として，リクエスト数は増加しているが carts の CPU 使用量は減少し原因と通知された．そのため，carts の CPU リソースをスケールアップして再度，実験を行った結果，carts でのエラーと原因の通知は発生しなかった．提案の改善として，信頼性を高めるため，CPU 使用量だけでなくアクセスログも同時に分析することが必要である．具体的には，500 番以上のエラーコードを持つアクセスログも分析することでより詳細な異常の特定が可能である．

参考文献

[1] Thönes, J.: Microservices, *IEEE software*, Vol. 32, No. 1, pp. 116–116 (2015).
[2] 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)*, IEEE, pp. 200–210 (2019).
[3] Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J. and Tilkov, S.: Microservices: The journey so far and challenges ahead, *IEEE Software*, Vol. 35, No. 3, pp. 24–35 (2018).
[4] Behera, A., Panigrahi, C. R., Behera, S., Patel, R. and Bera, S.: trACE-Anomaly Correlation Engine for Tracing the Root Cause on Cloud Based Microservice Architecture, *Computación y Sistemas*, Vol. 27, No. 3, pp.

791–800 (2023).

[5] Zhao, N., Chen, J., Peng, X., Wang, H., Wu, X., Zhang, Y., Chen, Z., Zheng, X., Nie, X., Wang, G. et al.: Understanding and handling alert storm for online service systems, *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*, pp. 162–171 (2020).

[6] Wang, L., Zhao, N., Chen, J., Li, P., Zhang, W. and Sui, K.: Root-cause metric location for microservice systems via log anomaly detection, *2020 IEEE international conference on web services (ICWS)*, IEEE, pp. 142–150 (2020).

[7] Chen, J., He, X., Lin, Q., Zhang, H., Hao, D., Gao, F., Xu, Z., Dang, Y. and Zhang, D.: Continuous incident triage for large-scale online service systems, *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, pp. 364–375 (2019).

[8] Chen, J., Zhang, S., He, X., Lin, Q., Zhang, H., Hao, D., Kang, Y., Gao, F., Xu, Z., Dang, Y. et al.: How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems, *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pp. 373–384 (2020).

[9] Liu, G., Mok, A. K. and Yang, E. J.: Composite events for network event correlation, *Integrated Network Management VI. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management.(Cat. No. 99EX302)*, IEEE, pp. 247–260 (1999).

[10] 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).

[11] Wang, Y. and Li, X.: FastTransLog: A Log-based Anomaly Detection Method based on Fastformer, *2022 9th International Conference on Dependable Systems and Their Applications (DSA)*, IEEE, pp. 446–453 (2022).

[12] Zhang, Q., Jia, T., Wu, Z., Wu, Q., Jia, L., Li, D., Tao, Y. and Xiao, Y.: Fault localization for microservice applications with system logs and monitoring metrics, *2022 7th International Conference on Cloud Computing and Big Data Analytics (ICCCBDA)*, IEEE, pp. 149–154 (2022).

[13] Cai, Y., Han, B., Su, J. and Wang, X.: TraceModel: An Automatic Anomaly Detection and Root Cause Localization Framework for Microservice Systems, *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, IEEE, pp. 512–519 (2021).

[14] Jing, N., Li, H. and Zhao, Z.: A microservice fault identification method based on LightGBM, *2022 IEEE 8th International Conference on Cloud Computing and Intelligent Systems (CCIS)*, IEEE, pp. 709–713 (2022).

