ログのPriorityを用いたシステム障害発生時における非圧縮による検索時間の短縮

金子 拓磨[1]　大野 有樹[2]　串田 高幸[1]

概要：
ログは web サイトやシステムが正常に稼働しなくなったときに，原因を特定する目的で使用される．そのため出来る限り残しておく必要があり，圧縮して保存することにより容量の削減が出来る．しかし圧縮して保存すると，圧縮せずに保存しているログに比べて検索時間が長くなる．課題はログを圧縮または非圧縮で保存する際の基準をどのように決定するかである．システム障害の発生原因に関するログのみを非圧縮で保存することにより，検索時間の短縮と容量の確保が可能である．本稿では，ログの重要度の指標である Priority の Warning 以上のログを非圧縮にして保存する．検索対象は非圧縮ファイルのみであるため，検索の応答時間をすべて圧縮するよりも短縮できる．基礎実験では Priority が記載されているログを出力を確認した．Locust による負荷試験によって Sock Shop からログを出力させた．結果，stdout，stderr を含めたログが 29,469 件出力され，そのうち stdout で発生した Priority が INFO のログが 90 件出力された．この結果より Priority のみの範囲設定ではなく，別のログレベルやステータスコードも組み込んだ非圧縮の基準に設定し直す必要がある．

1. はじめに

背景

web サイトでは一般的にログを一定の形式で記録し保存する．ログはコンピュータシステムやソフトウェア，ネットワークで発生する情報を記録するためのファイルである．

ログの重要性は広く知れ渡っている．ログはシステムの正常性情報を確認することのできる現状唯一のデータである [1]．ログはバク修正，異常検出，テスト結果の分析，システム監視と多くのソフトウェア開発活動中に使用される．ログの幅広い用途と有用性により，開発者はソースコードに大量のログ記述を埋め込むようになった [2, 3]．ログは上記のことを確認できるようにするため出来る限り削除せず保存する必要がある．

ログに記述されている内容の中に Priority という指標の項目がある．Priority は RFC5424 というメッセージフォーマットに記述されており，そのログの重要度の示す項目である[*1]．RFC5424 とは，今までデファクトスタンダードだった syslog をセキュリティ強化のために標準化作業を行うIETF の Syslog Working Group によって 2001 年に標準化され公開された RFC3164 が廃止されたのちに公開されたものである[*2]．重要度は高い物から，Emergency, Alert, Critical, Error, Warning, Notice, Informational, Debug となっている．例として Emergency はシステムが使えないという情報のログで，Error はエラー状況の情報のログ，Informational は単純な情報メッセージのログが該当する．

IT 分野の仮想化技術においてコンテナという技術がある．コンテナとはアプリケーションの実験環境を，その他のコンピュータシステムと分割して影響を与えないようにする仕組みである [4]．コンテナの特徴はいくつかある．1 つ目は他のコンピュータシステムと独立しているためサービス同士が互いに干渉しないこと．2 つ目は仮想化マシンよりも軽量な点である．システムのリソースが効率的であり，起動や終了が速く，スケーリングが容易である．3 つ目は高い移植性である．コンテナは実験環境に依存しないため，別環境でも一貫した動作が可能になる．4 つ目は管理の行いやすさである．コンテナはパッケージ化したイメージをそのままにアプリケーションを稼働させる．そのため複雑なセットアップ作業を必要とせず，アプリケーションの管理が容易になる[*3]．この管理をするときに用いられる

[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

[*1] https://datatracker.ietf.org/doc/html/rfc5424

[*2] https://www.rfc-editor.org/info/rfc5424

[*3] https://business.ntt-east.co.jp/content/cloudsolution/column-420.html

ものに Kubernetes が存在する．Kubernetes は k8s とも呼ばれており，コンテナ化されたアプリケーションをデプロイ，スケーリング，管理するためのオープンソースのプラットフォームである [5].

Sock Shop は Docker，Kubernetes，Amazon ECS 環境で動かすことが出来る EC サイトのデモアプリケーションであり，実際にイギリスに拠点を構えている靴下専門販売ショップでもある．ソフトウェア開発はマイクロサービスとなっており，front-end をはじめとした計 15 種類のコンテナで起動している [6].

課題

本稿における課題はストレージの容量を削減するためにログを出来るだけ圧縮しながら検索範囲のログは非圧縮で保存するとき，その圧縮の判断をどのように選定するかである．もしログをすべて圧縮し保存した場合ストレージ容量は確保しやすくなるが，圧縮ファイルをメモリ上で解凍する時間が発生するため検索全体の時間が長くなる．反対にログをすべて非圧縮で保存した場合圧縮したファイルを解凍する時間が無くなるため検索全体の時間が短くなるが，必要なストレージ容量が増加してしまう．

圧縮ファイルと非圧縮ファイルの検索時間を比較する実験を行った結果，非圧縮ファイルの検索時間が最小で 22.80 秒，最大 23.86 秒で，圧縮ファイルの検索時間が最小で 42.40 秒，最大で 44.96 秒となり，非圧縮ファイルの方が圧縮ファイルより検索時間が短くなる [7].

各章の概要

第 2 章では関連研究について述べる．第 3 章では本稿の提案方式の説明とユースケースの説明する．第 4 章では今回の提案に対する基礎実験の方法について述べる．第 5 章では評価方法と分析手法について述べる．第 6 章では本稿の議論についてを述べる．第 7 章では本稿のまとめを述べる．

2. 関連研究

ログをマイニングするときの時間はログデータのサイズに依存する．そしてシステム障害が起きたとき，システムログは膨大なログを生成し繰り返し計算することが困難になる点を課題として挙げている．著者はこの解決方法として，システムログをメッセージごとに分割し，インクリメンタルにマイニングする方法を提示している [8]．本稿とは課題がログのサイズと時間の短縮という点は一致しているが，そのログを使う対象がマイニングという違いがある．

ログの解析と抽象化のアプローチは多く研究されており，現場での適用性が低いことが問題である．著者は情報セキュリティの観点から最先端のログ解析および抽象化アルゴリズムを分析し，以前の分析を再現する．そしてフォレンジック分析を目的としたログファイルの解析と抽象化の能力を評価している．[9] 本稿とは課題がログの解析を最適化するという点は関連しているが，方法が再現を用いて評価することを重要としているという違いがある．

トラブルシューティングや問題の診断のため，分散システムによって生成されたシステムログが使用される．しかし，分散システムの規模と複雑さが増大しているため，手動でシステムログを検査して異常を検出することは現実的ではない．著者は異常検知の為の非構造ログ分析手法を提案している [10]．本稿とは課題がログの検索という点は酷似しているが，検索の速度ではなく異常の検出に焦点を当てている点が異なっている．

イベントログやログファイルはシステムとネットワーク管理において重要な役割を果たしているが，未知の障害ログが発生した場合，ログモニターではパターンデータベース内に一致するメッセージが存在しないため，ファイル内の対応するメッセージを無視してしまう．そのため著者はログファイルから頻繁に発生するパターンを検出し，ログファイルプロファイルを構築し，異常なログファイル行を特定するのに役立つログファイルデータセット用の新しいクラスタリングアルゴリズムを提案している [11]．本稿とは異常なログを発見するという目的は一致しているが，提案方式が新たなクラスタリングアルゴリズムの提案という点が異なっている．

3. 提案

提案方式

本稿の提案方式では，ログの一文の構成要素にある重要度を示す指標である Priority を基準とし，図 1 に示すように圧縮するログの対象を決定する．Priority は Sock Shop のソフトウェアの開発者の中のログの Priority を管理している人が設定している．

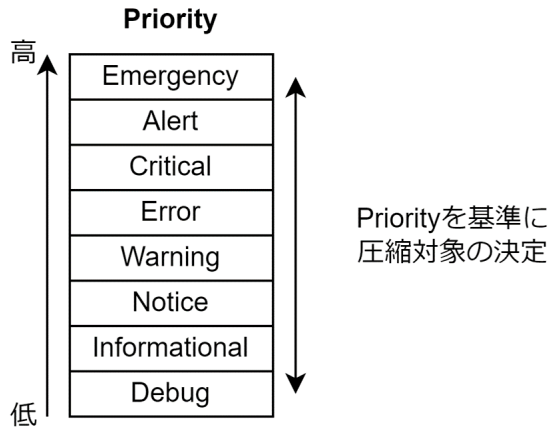

図 1 提案方式における Priority の基準

ログに書かれている Priority の Warning 以上のログを

基準に圧縮の判断をする．基準を Warning 以上にする理由としては，システムの障害に関わるログが Warning からであるためである．. Warning 未満の Priority として，Debug，Informational，Notice が存在するが，Debug はデバックレベルの報告メッセージ，Informational はシステムの情報メッセージ，Notice は重要だが正常なメッセージである．以上の理由から Priority のを用いた圧縮基準として Warning 以上のログを設定する．

図 2 はシーケンス図であり，提案方式のログの動きを示している．ライフラインの構成要素としてログ確認，ログ圧縮，ログ保存，Elasticsearch が挙げられる．以下にそれぞれのライフラインで何を実行するのかを説明する．

最初にログ確認から Elasticsearch に向けて，保存されている一日分のログの中に Priority の Warning 以上のログが含まれているかの確認を行うためのリクエストを送信する．Elasticsearch が検索を行い，その検索結果をログ確認に送信する．ログ確認は Elasticsearch から送られてきた検索結果に Warning 以上のログが含まれていないと確認したとき，ログ圧縮に対して該当ログの一日分のログの取得を依頼する．ログ圧縮はログ確認から依頼されたとき，Elasticsearch に向けて該当ログを要求し，Elasticsearch から非圧縮ファイル状態の該当ログを取得する．ログ圧縮はログを取得したとき非圧縮ファイルを圧縮し，ログ保存へ圧縮ファイルを受け渡す．ログ保存はログ圧縮から受け取ったログをログサーバ内に存在するストレージに保存する．

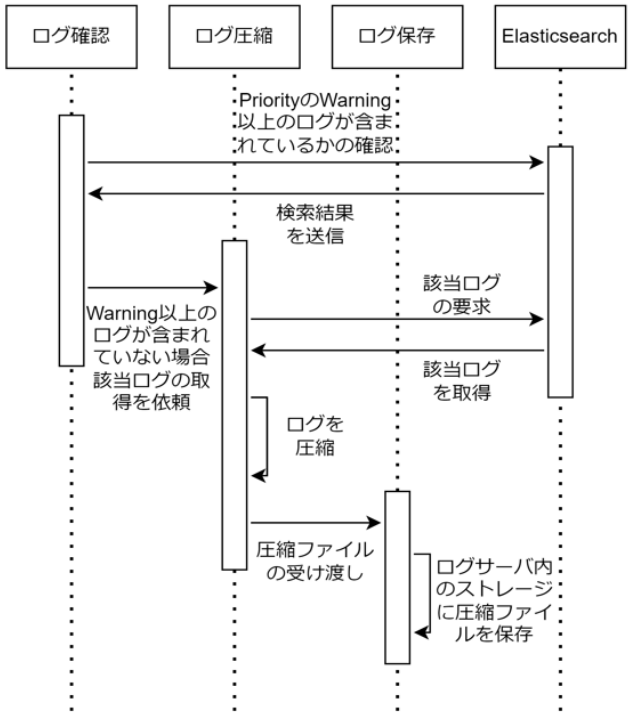

図 2　提案方式のシーケンス図

図 3 は図 2 の中のログ確認を示している．ログの中には 5 つの要素があり，Timestamp，Hostname，Tag，Priority，Message となっている．まずログ確認はログの Priority を確認する．ログの Priority を確認するためログを Logstash でパースして，先ほど挙げた 5 つの要素ごとにデータを構造化して Elasticsearch に保存する．その中から Priority を取り出し確認する．Priority は 8 種類あり Emergency，Alert，Critical，Error，Warning，Notice，Informational，Debug である．Priority が Warning 以上である Emergency，Alert，Critical，Error，Warning が含まれる場合は Warning 以上が含まれているログとして扱う．対して Priority が Warning 未満である Notice，Informational，Debug のみしか含まれない場合は Warning 未満のログとして扱う．

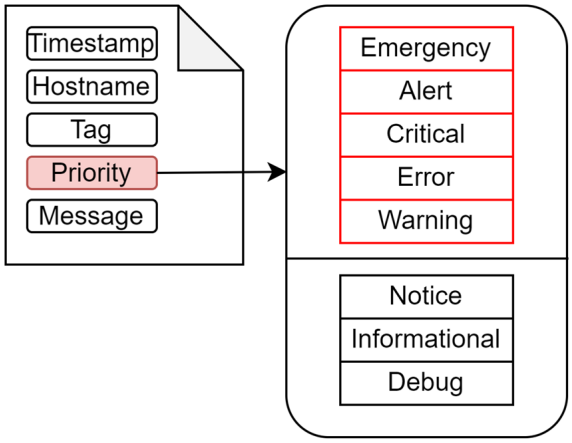

図 3　ログ確認

図 4 はログ圧縮の条件を示している．

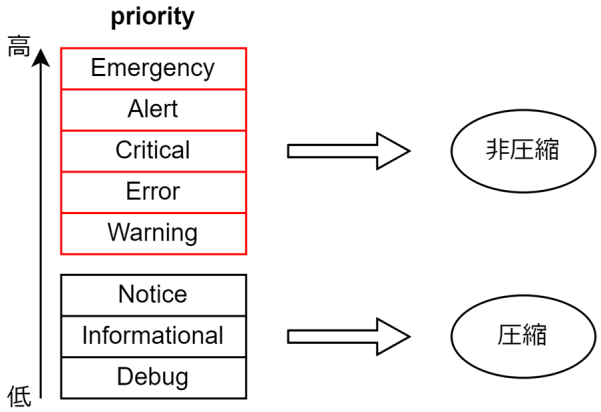

図 4　ログ圧縮の条件

ログ検索から Elasticsearch に検索を求めた時，Priority の 8 種類の要素の内 Warning 以上のログが含まれている場合は Elasticsearch 内にそのまま残しておく．Warning 以上のログが含まれていない場合は，ログ圧縮が一日分のロ

グを要求し取得したログを圧縮する．

図 5 は図 2 の中のログ圧縮を行うタイミングを示している．毎日 0 時になったときに，ログ確認が Elasticsearch に向けて保存している前日分の非圧縮ログの Priority に Warning 以上のログが含まれているかの確認をする．ログをファイルとして管理しやすくするため，日付によって分かれるように 0 時に実行し，前日分を保存する．Elasticsearch の検索で Warning 以上のログが含まれていないという検索結果が出た場合，ログ圧縮が該当ログを取得する．その後ログ圧縮は取得した非圧縮ログを圧縮しログ保存へ送る．ログ保存は受け取った圧縮ログをログサーバ内のストレージに保存する．

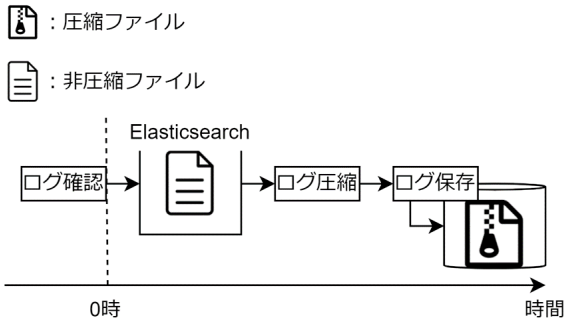

図 5　ログの圧縮と非圧縮のタイミング

ユースケース・シナリオ

本稿の提案方式は靴下のオンライン販売をしている Sock Shop を想定している．システムはアプリケーションサーバでサイトを運用しており，アプリケーションサーバ内で発生したログをログサーバに転送して保存を行う．ログを保存する際にログサーバの容量の削減のため，圧縮して保存することを前提とする．システム障害が起きているときログの検索を行う際に，検索時間の短縮のために圧縮せずに保存する．そのため圧縮時の検索と比較して検索時間の短縮を測る．

図 6 は本稿のユースケースシナリオを示している．システム管理者は，アプリケーションサーバとログサーバを管理している．ログファイルはアプリケーションサーバから Filebeat によってログサーバの Logstash へ転送される．転送されたログは Logstash でパースされる．パースされたログは Elasticsearch に送られ保存される．毎日 0 時になったとき，圧縮の判断が Elasticsearch に検索を呼びかけ，条件にあったログを取得して圧縮し保存用のストレージに保存する．システム管理者がファイルを検索するときは，Kibana に対して検索を呼びかけ Elasticsearch に保存されている非圧縮のログを検索する．

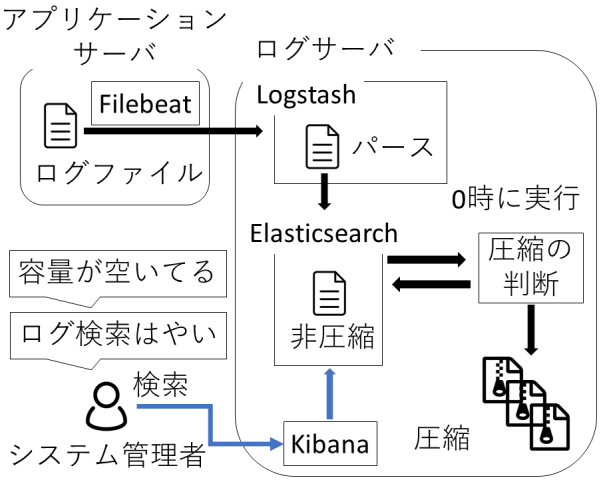

図 6　ユースケースシナリオ

4.　実装

本稿の実装では提案方式を用いてログを分割し VM 内で検索を行い検索速度の短縮を図る．

図 7 は実装における構成要素の図である．構成要素としてログ確認，ログ圧縮，ログ保存の 3 つがある．

ログは Logstash によってパースされ Elasticsearch に保存されている．

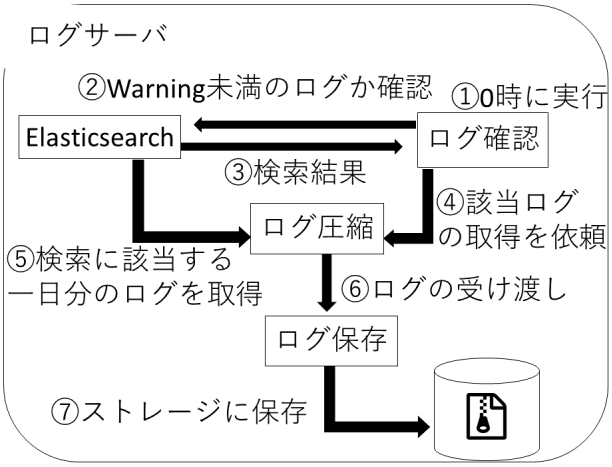

図 7　実装するソフトウェアの構成図

ログ確認は毎日 0 時になった時 Elasticsearch に保存されている非圧縮ファイルに対して，Priority の Warning 以上のログが存在するかの確認のリクエストする．Elasticsearch から検索結果が返ってきたとき，Warning 以上のログが含まれていないことが分かった場合ログ圧縮へ該当ログの取得を依頼する．Warning 以上のログが含まれている場合はログ圧縮に対しての出力は何も行わない．

ログ圧縮はログ確認から該当ログの取得の依頼がされた

とき，Elasticsearch から検索のあった Priority の Warning 以上が含まれていない一日分のログを取得する．その後取得した非圧縮ファイルを圧縮し，ログをログ保存へ受け渡す．

ログ保存はログ圧縮から受け取った圧縮ファイルをサーバ内にあるストレージに保存する．

5. 実験

本稿では提案のために Priority の判断基準を決定するための実験を行う．本稿の実験の目的はどんな障害を発生させれば，どのようなエラーが出るかを確認するためである．各障害の条件に合ったエラーの Priority を確認し，提案方式での圧縮・非圧縮の判断のラインを決定するために使用する．今回の実験では Locust を用いて負荷試験を行う．合わせて Chaos Mesh を用いて障害注入も行う．

実験環境

実験環境は ESXi の VM を用いる．VM は Sock Shop 用の KubernetesVM と負荷試験用の LocustVM を構築している．

KubernetesVM は OS が Ubuntu 22.04 LST であり，vCPU が 2 コア，RAM メモリは 5GB で，HDD が 50GB に設定する．KubernetesVM では Sock Shop と Chaos Mesh をコンテナで起動している．

LocustVM は OS が Ubuntu 22.04 LST であり，vCPU が 4 コア，RAM メモリは 8GB で，HDD が 16GB に設定する．LocustVM では直接 Locust をインストールしている．Number of users は最大のユーザ数を示しており 180 に設定, Spawn rate は 1 秒ごとに増えるユーザ数を示しており 1.0 で設定した．Run time は障害の発生時間で 3 分に設定した．Number of users は 1988 年のサッカーワールドカップの web サイトの 1 秒間の平均アクセス数である [12]．Run time が 3 分なのはすぐにログを確認できるよう短期間に設定するためである．Spawn rate は 3 分間で最大 180 件のアクセスに到達させるため，1 秒につき 1 リクエストずつ増加させている．

Chaos Mesh はコンテナとしてインストールし実行している．Inject into は障害を注入する対象を示しており Kubernetes を対象とした．Kind of obstacle はどのような障害を注入するかを示しており Network Attack の Loss の障害を発生させた．これは障害を発生させるための設定を簡単に選択できるためである．The percentage of packet loss はどのくらいの Loss を発生させるのかを示しており，数値は 50%に設定する．ログとして出力されたとき目視で確認しやすくするためである．Direction は障害を発生させる対象への Loss となっている．Duration は障害を発生させる時間であり，60 秒の障害時間に設定した．Chaos Mesh が 60 秒なのは，Locust の障害発生中に別の障害を追加したときの変化を確認するためである．

以下に実験に用いる VM の構成要素と Locust，Chaos Mesh の実行条件を示す．

KubernetesVM の構成要素

- OS：Ubuntu 22.04 LST
- vCPU：4 コア
- RAM：5GB
- HDD：50GB

LocustVM の構成要素

- OS：Ubuntu 22.04 LST
- vCPU：4 コア
- RAM：8GB
- HDD：16GB

Locust の実行条件

- Number of users：400
- Spawn rate：20.0
- Run time：3m

Chaos Mesh の実行条件

- Inject into：Kubernetes
- Kind of obstacle：Network Attack and Loss
- The percentage of packet loss：70(%)
- Direction：to
- Duration：60s

実験結果と分析

実験結果のグラフを図 8 に示す．

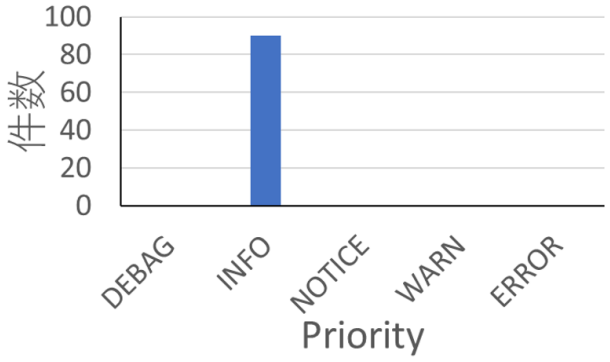

図 8 基礎実験の結果

縦軸は件数, 横軸は Priority を示している．Priority の出力として，DEBUG，INFO，NOTICE，WARN，ERROR がそれぞれ何件出力されたかを示している．

実験の結果，負荷試験の 3 分間で 29,469 件のログが発

生し，stdout が 23,823 件，stderr が 5,646 件であった．Priority が含まれるログは stdout のログにのみ発生し，件数は，INFO が 90 件，他の DEBUG，NOTICE，WARN，ERROR がそれぞれ 0 件だった．出力された 90 件の INFO はすべて carts サービスから出力されたログである．90 件のログはいずれもタイムスタンプと割り振られた ID や数値が異なるのみで，同じ内容のログだった．出力されたログの例をソースコード 1 に示す．stdout として出力されたソースコード 1 は mongodb とのコネクションを示したログであると分かる．

ソースコード 1 INFO の例

```
/var/log/pods/sock-shop_carts-85dddb79b4-
    dnjtd_cda29425-dd04-419c-9e89-bb90af54b67b
    /carts/0.log:2023-08-07T05:22:10.695723863Z
    stdout F 2023-08-07 05:22:10.695 INFO [carts
    ,f51974aca1d1eb11,a05cc388d9bbc31b,false]
    6 --- [-nio-80-exec-17] org.mongodb.driver.
    connection : Opened connection [
    connectionId{localValue:12, serverValue
    :12}] to carts-db:27017
```

本来 Priority が Warning 以上であるログの出力を発生させることにより提案方式を用いることとしたが，今回の基礎実験では Priority が Informatinal のログしか出力されなかったため，提案手法である Priority のみでの圧縮範囲の設定が適切ではないことが分かった．

6. 議論

本稿の議論は 2 つある．1 つ目は Elasticsearch にある非圧縮で保存されているログが検索されなかった場合，そのログをいつまで非圧縮で保存しておくかという問題である．この解決方法として，一営業日以内に検索がされなかった場合，対象のログを圧縮してログサーバのストレージに保存し直すことが挙げられる．検索は始業から終業までの時間内で行われるものとする．対象のログが検索されたかの判断は，そのログのファイルが一度でも開かれたかで判定する．

2 つ目は今回の基礎実験によって出力された Priority が記述されている stdout のログがすべて Informational であることである．Priority の出力は Informational のみである．その解決策として，提案の対象のログを stdout のみに発生した Priority だけではなく，出力されたログレベルやステータスコードに 500 番台が含まれる条件を付け加える．これにより提案の改善が見込まれる．ステータスコードを 500 番台に設定した理由は，このステータスコードがサーバエラーを示しているためである．今回の実験対象は既存のアプリケーションを使用した．これにより Priority の調整を行うことができない．そのために Informational 以外の Priority も出力できる環境を作成し，Priority は Warning 以上のログを出力させる．その際の Priority の出力条件を過去のシステム障害の事例を参考にし，その障害を再現することが必要である．そのために大量のリクエストが送られたことが原因でシステム障害が発生した新型コロナウイルスのワクチン予約サイトを再現し，Priority の条件を設定する[*4]．そして今回は障害注入で Loss を使用したが，現実の障害を再現するために他の障害注入を行う．

7. おわりに

ログは web サイトやシステムで障害が発生したときに，原因を特定する目的で使用される．そのためログを削除することはせず長期的に保存する必要がある．ストレージにログが溜まっていくと容量が圧迫されるため圧縮して保存することでストレージの容量を削減することができる．しかし圧縮されたログの検索を行う場合，非圧縮のログを検索するよりも時間がかかってしまう．本稿ではシステム障害発生時のログを確認するため，ログの重要度を示す指標である Priority の Warning 以上のログを非圧縮にして保存することで，ログの検索の時間を短縮しつつストレージ容量の確保を図る方法を提案した．Warning 以上と設定したのはシステム障害に関わるログが Warning からであるためである．本稿では Warning 以上の Priority という設定が提案の条件が適切かを確認するための基礎実験を行った．基礎実験では Locust と Chaos Mesh を用いて障害を注入し実際にどのようなエラーが発生するのかを確認した．その結果出力されたエラーが全体で 29,469 件で，Priority の情報が含まれているログが 90 件，さらに 90 件すべてが INFO のログであり，Priority の Warning 以上のログと言う条件が適切ではないことが分かった．この結果より Priority のみの範囲設定ではなく，別のログレベルやサーバエラーのメッセージである 500 番台のステータスコードも組み込んだ非圧縮の基準に設定し直す．もしくは過去のシステム障害事例を確認し，その障害を再現することで Priority の条件を設定し直す必要がある．

謝辞 本テクニカルレポートを執筆にあたりご指導いただきました東京工科大学院バイオ・情報メディア研究科コンピュータサイエンス専攻の河竹純一さん，東京工科大学コンピュータサイエンス学部先進情報専攻の三上智徳さん，西嶌知良さん，増田和範さん，平尾真斗さんに御礼申し上げます．

参考文献

[1] Guo, S., Liu, Z., Chen, W. and Li, T.: Event extraction from streaming system logs, *Information Science and Applications 2018: ICISA 2018*, Springer, pp. 465–474 (2019).

[*4] https://piyolog.hatenadiary.jp/entry/2021/05/19/055949

[2] Hassani, M., Shang, W., Shihab, E. and Tsantalis, N.: Studying and detecting log-related issues, *Empirical Software Engineering*, Vol. 23, pp. 3248–3280 (2018).
[3] Zhang, L., Xie, X., Xie, K., Wang, Z., Lu, Y. and Zhang, Y.: An efficient log parsing algorithm based on heuristic rules, *Advanced Parallel Processing Technologies: 13th International Symposium, APPT 2019, Tianjin, China, August 15–16, 2019, Proceedings 13*, Springer, pp. 123–134 (2019).
[4] Bentaleb, O., Belloum, A. S., Sebaa, A. and El-Maouhab, A.: Containerization technologies: Taxonomies, applications and challenges, *The Journal of Supercomputing*, Vol. 78, No. 1, pp. 1144–1181 (2022).
[5] Baur, A.: Packaging of kubernetes applications, *Proceedings of the 2020 OMI Seminars (PROMIS 2020)*, Vol. 1, Universität Ulm, pp. 1–1 (2021).
[6] 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).
[7] 金子拓磨，高橋風太，大野有樹，串田高幸：サーバダウン時におけるログファイルの非圧縮による検索時間の短縮，技術報告 CDSL-TR-135，Tokyo University of Technology CDSL Technical Report, (online: https://ja.takcslab.org/tech-report) (Jan.18, 2023).
[8] Mizutani, M.: Incremental mining of system log format, *2013 IEEE International Conference on Services Computing*, IEEE, pp. 595–602 (2013).
[9] Copstein, R., Schwartzentruber, J., Zincir-Heywood, N. and Heywood, M.: Log abstraction for information security: Heuristics and reproducibility, *Proceedings of the 16th International Conference on Availability, Reliability and Security*, pp. 1–10 (2021).
[10] Fu, Q., Lou, J.-G., Wang, Y. and Li, J.: Execution anomaly detection in distributed systems through unstructured log analysis, *2009 ninth IEEE international conference on data mining*, IEEE, pp. 149–158 (2009).
[11] Vaarandi, R.: A data clustering algorithm for mining patterns from event logs, *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003)(IEEE Cat. No. 03EX764)*, Ieee, pp. 119–126 (2003).
[12] Arlitt, M. and Jin, T.: A workload characterization study of the 1998 world cup web site, *IEEE network*, Vol. 14, No. 3, pp. 30–37 (2000).

