Webアクセスログのステータスコードと URLの分類による検索の応答時間の短縮

大野 有樹[1]　串田 高幸[1]

概要：システム管理者は原因を調査することで，システム障害の対応策を決める．ログはシステムの動作を記録したデータであり，原因の調査に用いられる．システム管理者はログを検索することでシステム障害の原因の絞り込みや発生時刻を特定している．検索の応答時間はシステム障害が解消するまでの時間に含まれる．したがって，検索の応答時間の増加はシステム障害が解消するまでの時間の増加につながる．課題はシステム障害の原因調査のためにログ検索をする際に，ログの検索対象の件数が多くなるにつれ，検索の応答時間が長くなることである．提案では HTTP アクセスログから共通したステータスコードと URL ごとに分けて保存することで，ログの検索時に条件に当てはまらない検索対象を検索する時間を無くし，検索の応答時間を短縮した．基礎実験として障害対応の時間を調査するために，Kubernetes クラスタ内で発生した Evicted エラーの原因を調査した時間を記録した．システム障害対応で行った項目のうち，Web サーバーを構築している Flask の Pod のログ検索の時間が合計で 4 分であった．システム障害の対応時間の全体が 26 分であったため，約 15%がログ検索の時間であることが分かった．

1. はじめに

背景

システム障害が発生した際に障害を迅速に復旧させなければ企業は金銭的リスクを被る [1, 2]．例えば世界最大のインターネットオークションサイトの eBay で発生したダウンタイムでは 1 時間当たり 225,000 ドルの損失が試算されている [3]．そのためシステム管理者は迅速に障害を復旧させなければならない．

システムの障害発生時にシステム管理者はログを検索することで原因を調査する [4]．ログはシステムの動作を記録したテキストである [5]．ログの種類はアプリケーションやサーバーによって異なる．例えば, Web サーバーにアクセスがあった際に記録されるのがアクセスログである．アクセスログ内には，Web サーバーに対してアクセスを行った端末の IP アドレスや，ステータスコード，アクセスしてきた端末の情報が記録される [6]．システム管理者は原因を調査することで，システム障害の対応策を決め，対処にあたる．例えば，オンライン決済サービスに Stripe がある．Stripe は仕様変更により，障害が発生した．Stripe の原因となった仕様変更は 3 ヶ月前であり，システム管理者は障害の発生した原因の調査のためにログを検索した [7]．

システム管理者はログを検索することでシステム障害の原因の絞り込みや発生時刻を特定している．検索の応答時間はシステム障害が解消するまでの時間に含まれる．したがって，検索の応答時間の増加はシステム障害が解消するまでの時間の増加である．

アクセスログのステータスコード 500 番台はサーバーエラーを示している．システム管理者は 500 番台でどのログが一番多いかを調べる．502 Bad Gateway はアプリケーションが起動していない場合に出力される．503 Service Unavailable は Web サーバーへのリクエスト数が多く返答ができない際に出力される．504 Gateway Timeout はアプリケーションの応答がタイムアウトする際に出力される．このことからも，原因調査の絞り込みの際に Web サイトの状態を判断することができる．

ログ検索に用いられるソフトウェアとして分散型検索・分析エンジンである Elasticsearch がある [8–10]．Elasticsearch は、効率的な全文検索と分析のために設計されたオープンソースの分散ビッグデータシステムである．

課題

課題はシステム障害の原因調査のためにログ検索をする際に，ログの検索対象の件数が多くなるにつれ，検索の応答時間が長くなることである．以前のテクニカルレポートの実験からログの件数が多くなるほど検索の応答時間が長

[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〒 192–0982 東京都八王子市片倉町 1404-1

くなることが分かった [11]．検索の応答時間はシステム障害が解消するまでの時間に含まれる．したがって，検索の応答時間の増加はシステム障害が解消するまでの時間の増加である．

以下に以前のテクニカルレポートで記録した基礎実験の概要を載せる [11]．実験は，Elasticsearch の API を用いて 100 回検索し，応答時間の最小値を記録する．検索対象は Web サーバーのアクセスログの EClog である [12]．VM は vCPU 3[Core] と RAM 4[GB] で実験する．

検索条件の期間を 1 カ月ずつ伸ばして，ログの件数を増やしながら検索の応答時間を測定した．1 ヶ月ずつ増やした理由はログの件数が増えるにつれ検索の応答時間が増えることを確かめるため，ログの対象の増やし方としてもっとも試しやすいものであるためである．ログの件数はそれぞれ 1 カ月ごとに 1 ヶ月分が 7,175,241[件]，2 ヶ月分が 12,735,845[件]，3 ヶ月分が 17,997,264[件]，4 ヶ月分が 23,183,227[件]，5 ヶ月分が 28,746,880[件]，6 ヶ月分が 35,157,691[件] となっている．

アクセスログを検索するクエリは URL ごとのステータスコードの集計と，ステータスコードごとの URL の集計の 2 つとする．

ステータスコードごとに URL がそれぞれ何件あるかを集計することによって，ステータスコードの番号ごとにどの URL がアクセスされているか分かる．使い方としては，例えばエラーのステータスコードである 502 が一番多い件数の URL を調べ，システム障害の原因となっている URL が分かるようになる．システム障害の原因となる URL が分かれば，どのアプリケーションのどのログを見れば解決できるかの足がかりとなる．

ステータスコードごとに URL がそれぞれ何件あるかを集計した検索クエリの応答時間はそれぞれ，1 ヶ月分で 1.52[秒]，2 ヶ月分で 2.5[秒]，3 ヶ月分で 3.48[秒]，4 ヶ月分で 4.42[秒]，5 ヶ月分で 5.55[秒]，6 ヶ月分で 5[秒] であった．

URL ごとにステータスコードがそれぞれ何件あるかを集計した検索クエリの応答時間はそれぞれ 1 ヶ月分で 1.05[秒]，2 ヶ月分で 1.64[秒]，3 ヶ月分で 2.2[秒]，4 ヶ月分で 2.75[秒]，5 ヶ月分で 3.32[秒]，6 ヶ月分で 4.05[秒] であった．

検索の範囲は最大で 1 週間を想定している．監視では分からないことがログで調べたいことである．例えば，管理者は障害発生時刻を知るためにログを検索する．障害発生時刻が分かると，他の関連するシステムを調べる際に影響がないかを調べることができる．また，障害報告を出すときにいつからいつにシステム障害が起きたか，どれぐらいのユーザーに影響があったかを正確にする必要がある．

5 分おきに監視をしている場合を想定する．8 月 2 日の 8:00 にアラートが管理者に向けて来たとする．管理者は監視で監視が失敗し始めた期間のログを検索する．そのため，8 月 2 日の 7:55 から 8 月 2 日の 8:00 を指定したログ検索を管理者はする．次に，エラーに関連していると思われるログメッセージを検索し，エラーメッセージが出力されているか確認する．エラーメッセージを見つけた際に，管理者はエラーメッセージを検索し，そのエラーメッセージが最初に出たログを見つける．最初にエラーメッセージが出力された時刻から障害の発生時刻が分かる．また，特定の時間とかアクセスが多くなると起きる障害の場合，再現性が低いため，障害の調査が長期化する．そのため，1 週間といった期間に渡って検索することもある．

以上の結果からログの検索対象の件数が多くなるにつれ，検索の応答時間が長くなることが分かる．

各章の概要

第 2 章の関連研究では関連する既存研究を述べる．第 3 章の提案では課題の解決するための提案手法とユースケースシナリオについて説明する．第 4 章の実装では実装方法について述べる．第 5 章の実験と分析では，実験環境と実験結果，分析について説明する．第 6 章の議論では提案手法について議論をする．最後に，第 7 章の結論では全体を簡潔にまとめている．

2. 関連研究

syslog を読み込み，ログの時刻に基づいてディレクトリパスを作成する手法がある [13]．ログの保存ではシステムログを読み込み，構造化し，ログ時刻に基づいたディレクトリパスにデータフレーズを格納している．検索では GNU Parallel を使って複数の SQL コマンドを並列に実行し，UNIX の Pipe メカニズムを使って実行結果を集約している．1 分毎にログを保存するため，検索操作は 1 分以上前に記録されたログに対してのみしかできない欠点がある．

必要に応じて読み込む検索エンジンを作成した手法がある [14]．第 1 に読み出しの増幅を低減し大規模リクエストを可能にするために，異なるステージのデータをグループ化しインデックス作成時にコンパクトに格納する．第 2 にブルームフィルタを採用し，早期にパスを刈り込み，転置インデックス内の位置の I／O を削減する．これにより，不要なデータの読み込みを回避している．この手法はレイテンシが 16 倍削減できるが，ストレージ領域の使用量が最大 50%増加するところに改善の余地がある．

検索クエリに合わせて検索を分類した手法を提案している [15]．主に有効なレコードの集合の一部と交差するレコードを検索対象とすることを指定する交差点クエリと，特定のレコードがデータ構造の中にあるかどうかを問うベストマッチクエリを分割している．この手法は検索の応答時間について改善の余地がある．

3. 提案

提案方式

HTTP アクセスログから共通したステータスコードと URL ごとに分けて保存することで，検索の応答時間を短縮する．ステータスコードと URL はそれぞれ完全一致するものを共通したログと定義する．例えば，EClog を例にあげると URL の場合，/szukaj.html と/szukaj.html/szukaj=query51341/opis=tak/nrkat=tak/kodprod=tak は異なる URL として保存する．/szukaj.html/szukaj=query51341/opis=tak/nrkat=tak/kodprod=tak はクエリ部分を変換と加工したものであるため，szukaj.html の後ろの URL はクエリとして扱う [12]．

図 1 は提案の概要について示している．提案の流れを以下に説明する．まず，ユーザーが Web サーバーに対してリクエストを送る．Web サーバーにあるサイトはユーザーからのリクエストを受け取りログを出力する．出力されたログはログ転送によってログサーバーに送られる．ログサーバーではログ配置がログ転送からのログを受け取り，ストレージにログを分類して，ログブロックとしてまとめて保存する．そのため，ブロックサイズは一定ではなく不均一になる．分類の基準はステータスコードごととアクセス先の URL ごとである．分類して保存したログは配置保存によってインデックスに保存場所が記録される．

次に検索の流れについて説明する．管理者はログサーバーにある検索システムに対して検索クエリを発行する．検索システムは発行した検索クエリからインデックスに検索をし，ログの保存場所を取得する．検索システムは取得した保存場所からストレージにアクセスし，検索結果として検索クエリに該当するログを取得する．このとき，ストレージ全体からではなく，検索クエリに該当するログブロックのみを取得するため，検索の応答時間を短縮することができる．そうして，得た検索結果を管理者へ表示して検索が完了する．

図 2 はログの分類について示している．ログの分類は Web サーバーで出力されたログをログサーバーに転送し，ResponseCode と URL で一致するものに分けて保存する．ログの分類について流れを説明する．まず，Web サーバーにあるサイトからログが出力される．出力されたログはログ転送によってログサーバーに転送される．転送されたログはログサーバー内のログ配置によって分類する．分類方法は ResponseCode と URL が一致するものである．例えばアクセスログが 200 の ResponseCode で/home の URL の場合，200 と名付けたディレクトリの下部に home とディレクトリ名をつけ，ログを保存する．また，アクセスログが 200 の ResponseCode で/user の URL の場合，200 と名付けたディレクトリの下部に user とディレクトリ名をつけ，

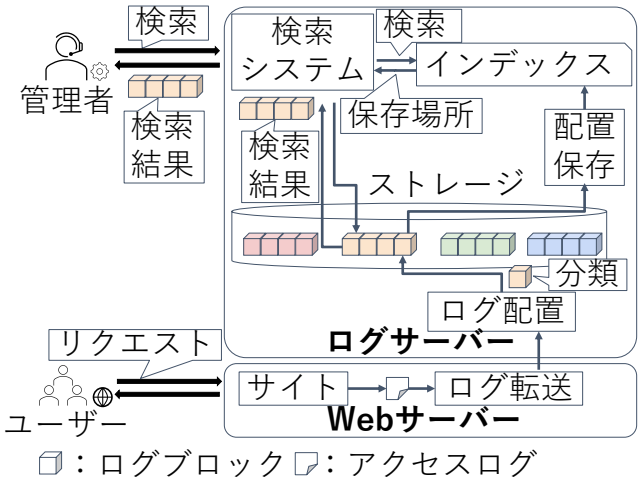

図 1 提案の概要

先ほどの home と異なるディレクトリにログを保存する．

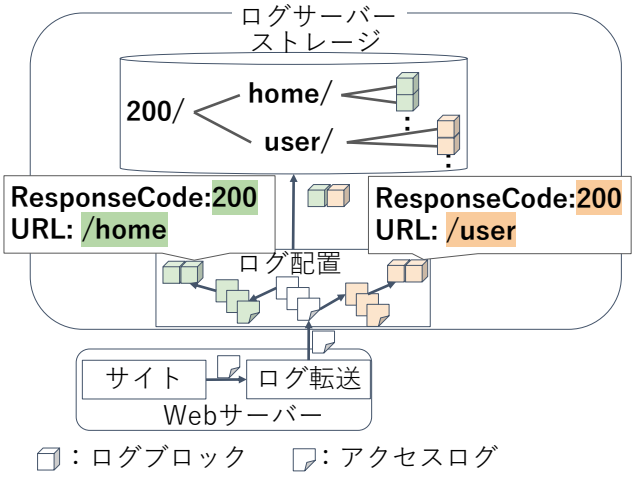

図 2 ログの分類

図 3 はログのインデックス作成について示している．ログのインデックス作成はストレージに保存されたログから ResponseCode インデックスと URL インデックスの 2 つを作成する．ログのインデックス作成について流れを説明する．まず，ストレージに保存されたログを配置保存が検知する．配置保存は新しく配置されたログがどのディレクトリに保存されたかを確認し，インデックスを作成する．URL はフルパスで完全に一致するものごとに分けて保存する．例えば，ResponseCode が 200 で URL が/home のアクセスログが追加された場合，ResponseCode インデックスの 200 と URL インデックスの/home にログがあることを記録する．また，ResponseCode が 200 で URL が/user のアクセスログが追加された場合，ResponseCode インデックスの 200 にログがあると記録し，と URL インデックスの/user にログがあることも記録する．

ユースケース・シナリオ

本ユースケースシナリオでは Web 小説のレビューサイ

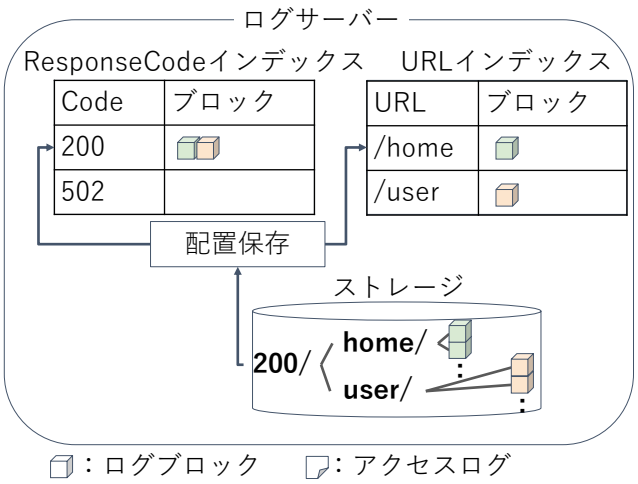

図 3 ログのインデックス作成

トを運営しているシステム管理者をターゲットにしている．図 4 はユースケースシナリオの図を示している．レ

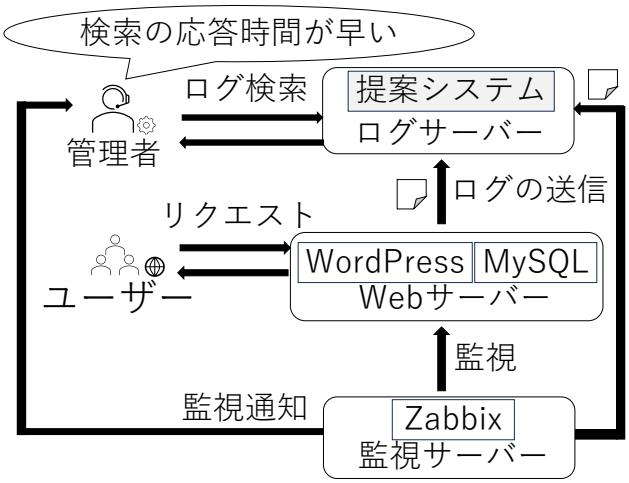

図 4 ユースケースシナリオ

ビューサイトは WordPress で製作している．外形監視には Zabbix[*1]を使用し，Web サイトを外部から監視している．ログ検索のシステムは提案のシステムを起動する．

監視では分からないことがログで調べたいことである．例えば，管理者は障害発生時刻を知るためにログを検索する．障害発生時刻が分かると，他の関連するシステムを調べる際に影響がないかを調べることができる．また，障害報告を出すときにいつからいつにシステム障害が起きたか，どれぐらいのユーザーに影響があったかを正確にする必要がある．

まず，小説のレビューサイトを対象とした外形監視が通知をする．外形監視で通知されるときはユーザーがサイトを閲覧できなくなることを示している．外形監視はユーザーと同じく外のネットワークからリクエストを送信し応答があるかを確かめる．HTTP のリクエストの返答やリクエストのレイテンシがどれだけかかったかを外形監視から取得できる．外形監視でトップページに 5 分おきにリクエストを送信すると想定する．

8 月 2 日の 8:00 にアラートが管理者に向けて来たとする．管理者は監視で監視が失敗し始めた期間のログを検索する．そのため，8 月 2 日の 7:55 から 8 月 2 日の 8:00 を指定したログ検索を管理者はする．ステータスコードごとにそれぞれ何件のログが出力されているかを調べる．その中で一番多い異常系のログすなわちステータスコード 400 以上で一番件数が多いステータスコードを調べる．調べた一番多い異常系のステータスコードのアクセスログを検索し，一番最初に発生した時刻を調べることで障害の発生時刻が分かる．障害の発生時刻が分かれば，その時刻から障害が収束した時刻までのステータスコードを調べることで何件のリクエストが失敗したか分かり，どのぐらいのユーザーに影響があったかが分かる．

このときに，提案のシステムを使うことで検索の応答時間を短縮できるため，原因調査の時間を短縮できる．

4. 実装

提案手法を基に，ログを分割して保存するソフトウェアを新たに作成する．

図 5 は実装の全体図を示している．

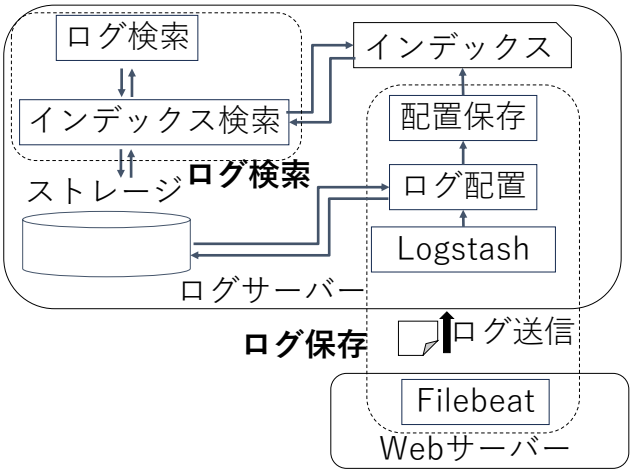

図 5 実装の全体図

作成するソフトウェアはログを保存するソフトウェアとログを検索するソフトウェアの 2 つである．ログの保存には 4 つの構成要素があり，Web サーバーに Filebeat，ログサーバーに Logstash とログ配置，配置保存がある．Filebeat と Logstash は既存ソフトウェアである．ログ検索にも 2 つの構成要素があり，ログサーバーにログ検索とインデックス検索がある．まず，ログの保存について説明する．

ログ保存は Web サーバーで出力されたログをログサーバーに保存するソフトウェアである．以下に構成要素を説

[*1] https://www.zabbix.com/jp

明する．Filebeat は Web サーバーで生成されたログをログサーバーに送信する．Logstash は Filebeat から受け取ったログをパースして保存する．Filebeat と Logstash は既存のソフトウェアを使用する．ログ配置は Logstash から受け取ったパースされたログをストレージに保存する．保存する際はログに記載されている ResponseCode とアクセス先の URL を調べ，記載されている内容からディレクトリを作成する．作成したディレクトリの中には ResponseCode とアクセス先の URL が一致するログを集めて保存する．ディレクトリ名は ResponseCode と URL に連番でつけた ID を”-”で区切り，命名する．配置保存はログ配置で保存したログをインデックスに保存する．インデックスは key-value ストアで保存する．key は ResponseCode とアクセス先の URL，value はストレージへのログの配置場所を記述する．

ログ検索はログ保存でストレージに保存したログを検索するソフトウェアである．以下に構成要素を説明する．ログ検索はシステム管理者からの検索クエリを読み取りインデックス検索へ検索内容を送信する．検索内容を受け取ったインデックス検索はインデックスに検索内容に該当するログがあるかを検索する．検索結果をもとにストレージに保存されているログを取り出し，インデックス検索はログ検索へ検索結果を送信する．ログ検索はインデックス検索から受け取った検索結果を検索クエリを実行したターミナルに出力する．

5. 実験

実験環境

基礎実験としてシステム障害発生時にログ検索にかかる時間を確かめた．実験環境を図 6 に示す．実験環境はログサーバーと自作したコメントを投稿できる Web サイトを用いた．ログサーバーは Elastic Stack である，Elasticsearch，Kibana，Filebeat，Logstash を起動している．Web サーバーは Flask と MySQL を組み合わせて Web サイトを構築している．

Ubuntu 22.04 をインストールした 6 台の仮想マシンを用いた．仮想マシンは ESXi を使用して作成した．そのうち 5 台の仮想マシン上には RKE2 による Kubernetes クラスタを作成している．1 台の仮想マシンは kubectl コマンドを実行するための仮想マシンサーバーとしている．Kubernetes クラスタを形成している仮想マシンの性能はいずれも同一の機能 (vCPU: 4[core],RAM:4[GB],SSD:50GB) で作成した．

この Kubernetes クラスタ内で発生した Web サーバーの Pod，Flask と MySQL が Evicted エラーを出力した原因を調査した時間を記録してシステム障害発生時にログ検索にかかる時間を確かめた．

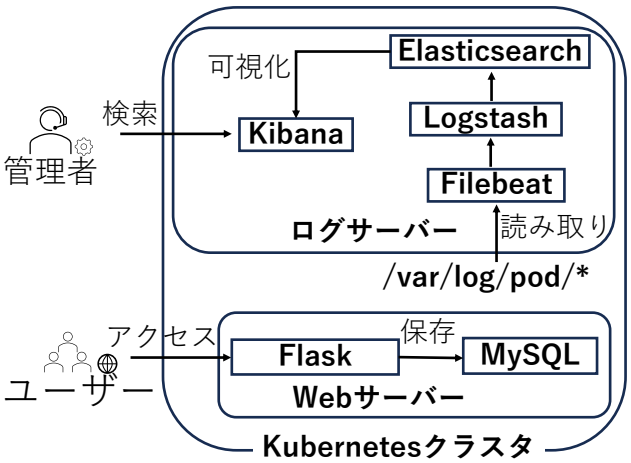

図 6 実験環境

実験結果と分析

実験で行った障害対応の流れを箇条書きにして載せる．

(1) 検知
(2) Elasticsearch によるログ検索
(3) /var/log/pods/の確認
(4) Kubernetes のステータスのログ探し
(5) kubelet.log の確認
(6) エラー文を Google 検索
(7) 障害が Google 検索の結果と一致しているか
(8) ストレージの増設
(9) Pod の再起動

(1) 検知: Web サイトにアクセスできないと知らせを受けて，Kubernetes にある Web サイトを構築している Flask と MySQL の Pod のステータスが Evicted になっていることを確認した．実際に Web サーバーにアクセスしたところ，ステータスコード 500 番の Internal Server Error が表示されていてアクセスできなくなっていた．

(2)Elasticsearch によるログ検索:/var/log/pods/のディレクトリにあるログは Elasticsearch に保存している．

(3)/var/log/pods/の確認: Pod が落ちる前のログを確認するために/var/log/pods/にあるログを確認した．ログから再起動した時刻が分かった．しかし，なぜ Pod が再起動したのか，Evicted になっているのか，原因となるログは記述されていなかった．

(4)Kubernetes のステータスのログ探し: 手がかりをつかむために Kubernetes のステータスが変化したログを記述したログを探した．Kubernetes のステータスは kubelet にて判断していることから kubelet のログを探した．研究室に居た学部 4 年の森井 佑誠に kubelet のログの場所を教えてもらい，/var/lib/rancher/rke2/agent/logs/kubelet.log にあると分かった．Kubernetes のマスターノードにあたる VM へ SSH 接続でログインし，/var/lib/rancher/rke2/agent/logs/kubelet.log が

あることを確認した．本来であれば，このログは Elasticsearch に転送して検索できるようにする必要のあるログである．

(5)kubelet.log の確認:AGE と再起動した時刻から 08-01T23 と 8 月 1 日の 23 時と指定して/var/lib/rancher/rke2/agent/logs/kubelet.log を検索した．検索をした結果，エラー文のようなものとして nodeCondition=[DiskPressure] を発見した．

(6) エラー文を Google 検索: エラー文である nodeCondition=[DiskPressure] を Google 検索をしてエラー文を調べた．検索した結果で一番上に出たサイトの「【Kubernetes】Pod がファイルシステムを逼迫させた際におきた事象[*2]」からファイルシステムの使用量が 90%を超えたときに起きることが分かった．

(7) 障害が Google 検索の結果と一致しているか: ファイルシステムの使用率の調べ方として，df コマンドを用いた．df コマンドの結果，ファイルシステムの/dev/mapper/ubuntu–vg-ubuntu–lv の使用率が 90%となっていることが分かった．

(8) ストレージの増設: ワーカーノードのストレージがそれぞれ 50GB になるように指定していたが，ファイルシステムに 25GB しか適応されていないことがわかった．対処方法としては Ansible を用いてパーミッションを変更してファイルシステムに適応させてストレージを増設した．

(9)Pod の再起動: ストレージを増設し，Evicted になる原因を取り除いたため，Web サイトを構築している Flask と MySQL の Pod を再起動させる．再起動させた後，Pod のステータスが Running になっていることを確認して，障害対応を終了した．ストレージの増設は根本原因の解決に至っているかどうかは判断できない．Kubernetes クラスタ内の PV，PVC がどのようにどれだけ使っているか，Kubernetes クラスタを構築する以外のストレージの使用量がどれだけあるかを調べ，ストレージ増設の判断が適切であったかを調査する必要がある．

図 7 に実験結果を示す．かかった時間は時刻から算出したため誤差は 1 分前後ある．

縦軸は先ほどあげた障害対応それぞれにかかった時間であり，単位は分である．横軸は先ほどあげた 9 つの障害対応の流れに対応した項目である．

システム障害対応で行った項目のうち，(2)Elasticsearch によるログ検索，(3)/var/log/pods/の確認が Flask の Pod のログ検索の時間であり，合計で 4 分である．この時間はすべてのログがあらかじめ Elasticsearch に転送されていれば，より短い時間になると分かる．システム障害の対応時間の全体が 26 分であったため，約 15%がログ検索の時間であることが分かった．

[*2] https://www.mtioutput.com/entry/k8s-node-pressure

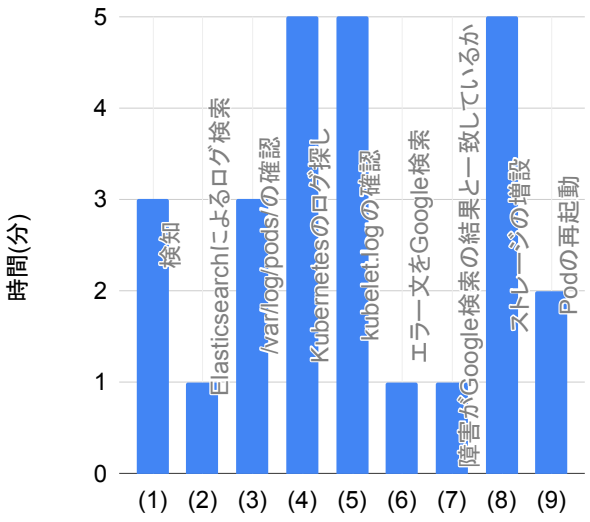

図 7 システム障害の対応にかかる時間の実験結果

6. 議論

現状ではログを保存する際に，ログブロックのサイズは出力傾向に依存している．ログブロックはファイルとして管理している．ログを検索する際にブロックの中身を参照するため，ファイルの Open，Close の時間がブロックを参照する分だけかかることになる．検索の応答時間を早めるためには，ファイルの Open，Close の時間とファイルを読む時間から 1 ファイルあたり何行のログがあると検索の時間を短くすることができるか調べる必要がある．そのため，基礎実験でファイルの Open，Close の時間とファイルを 1 行読むのにかかる時間を計測する．1 ファイルあたり何行のログが一番短く検索できるかを基準値として設定し，その値に合わせてブロックを作成する．ログブロックが基準値となる行数を超えた際に新しいブロックとして保存する．

提案手法では URL の分割方法をフルパスで一致するものとしている．しかし，実際にフルパスで分割すると，1 ファイルに 1 件のログが保存されることがある．インデックスの利点である集合を検索する時間の削減を活かすことができず，1 個 1 個検索することと違いが無くなる．そのため，フルパスで分割するのではなく，URL の/区切りの 1 つ目のみが一致するもので分けることでインデックスを使えば検索が早くなる状況を作り出す．

7. おわりに

課題はシステム障害の原因調査のためにログ検索をする際に，ログの検索対象の件数が多くなるにつれ，検索の応答時間が長くなることである．提案手法は HTTP アクセスログから共通したステータスコードと URL ごとに分けて保存することで，ログの検索時に条件に当てはまらない

検索対象を検索する時間を無くし，検索の応答時間を短縮する．基礎実験として障害対応の時間を調査するために，Kubernetes クラスタ内で発生した Evicted エラーの原因を調査した時間を記録した．システム障害対応で行った項目のうち，ログ検索の時間が合計で 4 分であった．システム障害の対応時間の全体が 26 分であったため，約 15%がログ検索の時間であることが分かった．

謝辞 本テクニカルレポート執筆にあたり，ご助言を頂きました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の高橋 風太さん，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平尾 真斗さん，圖齋 雄治さん，三上 智徳さん，増田 和範さん，森井 佑誠さんに御礼申し上げます．

参考文献

[1] Oostenbrink, J.: Financial impact of downtime decrease and performance increase of IT services (2015).

[2] Wang, S. S. and Franke, U.: Enterprise IT service downtime cost and risk transfer in a supply chain, *Operations Management Research*, Vol. 13, No. 1, pp. 94–108 (online), DOI: 10.1007/s12063-020-00148- (2020).

[3] Patterson, D. A.: A Simple Way to Estimate the Cost of Downtime, *LiSA* (2002).

[4] Koyama, T. and Kushida, T.: Log message with JSON item count for root cause analysis in microservices, *2023 6th Conference on Cloud and Internet of Things (CIoT)*, pp. 55–61 (online), DOI: 10.1109/CIoT57267.2023.10084901 (2023).

[5] He, P., Zhu, J., He, S., Li, J. and Lyu, M. R.: Towards Automated Log Parsing for Large-Scale Log Data Analysis, *IEEE Transactions on Dependable and Secure Computing*, Vol. 15, No. 6, pp. 931–944 (online), DOI: 10.1109/TDSC.2017.2762673 (2018).

[6] Grace, L. K. J., Maheswari, V. and Nagamalai, D.: Analysis of Web Logs and Web User in Web Mining (2011).

[7] Sillito, J. and Kutomi, E.: Failures and Fixes: A Study of Software System Incident Response, *CoRR*, Vol. abs/2008.11192 (online), available from ⟨https://arxiv.org/abs/2008.11192⟩ (2020).

[8] Kathare, N., Reddy, O. V. and Prabhu, V.: A comprehensive study of elasticsearch, *International Journal of Science and Research (IJSR)* (2020).

[9] L'Hôte, A. and Jeangirard, E.: Using Elasticsearch for entity recognition in affiliation disambiguation, *arXiv preprint arXiv:2110.01958* (2021).

[10] Zamfir, V.-A., Carabas, M., Carabas, C. and Tapus, N.: Systems Monitoring and Big Data Analysis Using the Elasticsearch System, *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*, pp. 188–193 (online), DOI: 10.1109/CSCS.2019.00039 (2019).

[11] 大野有樹小山智之串田高幸 Web アクセスログのステータスコードと URL の分類による検索時間の削減 *Tokyo University of Technology CDSL Technical Report*, No. CDSL-TR-140 (Apr.4, 2023).

[12] Chodak, G., Suchacka, G. and Chawla, Y.: EClog: HTTP-level e-commerce data based on server access logs for an online store (2020).

[13] Abe, H., Shima, K., Sekiya, Y., Miyamoto, D., Ishihara, T. and Okada, K.: Hayabusa: Simple and Fast Full-Text Search Engine for Massive System Log Data, *Proceedings of the 12th International Conference on Future Internet Technologies*, CFI'17, New York, NY, USA, Association for Computing Machinery, (online), DOI: 10.1145/3095786.3095788 (2017).

[14] He, J., Wu, K., Kannan, S., Arpaci-Dusseau, A. and Arpaci-Dusseau, R.: Read as Needed: Building WiSER, a Flash-Optimized Search Engine, *18th USENIX Conference on File and Storage Technologies (FAST 20)*, Santa Clara, CA, USENIX Association, pp. 59–73 (online), available from ⟨https://www.usenix.org/conference/fast20/presentation/he⟩ (2020).

[15] Bentley, J. L.: Multidimensional Binary Search Trees Used for Associative Searching, *Commun. ACM*, Vol. 18, No. 9, p. 509–517 (online), DOI: 10.1145/361002.361007 (1975).