Webアクセスログのステータスコードの分割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：システム障害が発生した際にシステム管理者はログを検索することで原因の特定を行う．迅速にシステム障害を復旧させるために，ログ検索の応答時間を短くする方法がある．ログ検索の応答時間を短くするために，並列で検索する手法がある．並列で検索するためにファイルの振り分けた数によって検索の分担を決めることができる．その際はファイルの振り分ける数によって検索の応答時間が異なる．検索の応答時間を短縮させるためにはファイルの振り分ける数を適切に設定しなければならない．本稿の目的はシステム障害発生時に，検索の応答時間を短縮することでシステム管理者の原因調査の時間を短縮することである．本提案手法はログをログサーバーに保存する前にファイルを振り分ける数を変えて検索の応答時間を測定し，検索の応答時間が短くなるファイルの振り分ける数を決める．検索の応答時間は並列処理の時間，ファイルの読み込み性能とトレードオフの関係であると想定し，ログの振り分ける数を決めることで検索の応答時間を短縮する．実験では Harvard Dataverse に載っている EClog の 2020 年 3 月 16 日の HTTP アクセスログを用いてステータスコード 400 以上の検索の応答時間を比較した．

1. はじめに

背景

システム障害が発生した際にシステム管理者はログを検索することで原因の特定を行う．ログはシステムの動作を記録したテキストデータである [1]．ログは用途によって種類が異なる．例えばアクセスログは Web サーバーにアクセスしてきた際に記録され，アクセス時刻，アクセス先，通信のメソッドが記録されている [2,3]．

システム管理者はログを検索しやすくするためにログサーバーを設置し，ログを収集する [4,5]．Web サーバーが 1 台の場合，ログを検索する場所が 1 箇所となる．対して，例えばスケーラビリティを高めることが目的として，Web サーバーを複数台設置した際にはログが分散する．ログが分散すると，システム管理者は複数のサーバーに SSH 接続して検索することが必要になる．そのため，システム管理者がログを検索する前に 1 つのサーバーにログを集約することで検索する場所を 1 箇所にする．

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 [6–8]．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．Elasticsearch には設定により，アルゴリズムに基づいて望ましいバランスを計算し，シャードを最適なノードに移動する機能がある[*1]．また，Elasticsearch の検索に用いられるソフトウェアとして Kibana があげられる [9–11]．

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 [12]．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．

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 [13,14]．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 [15]．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．

障害復旧に至るまでにシステム障害の調査をするためにログを検索する．ログ検索によりシステム障害の原因を調査し，暫定的な対応をした後もシステム障害が再度起きて

[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

[*1] https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-cluster.html

いないか確認する．ログ検索の時間を短くすることで，調査や暫定対応の内容確認の時間を短くすることができる．そのための方法の 1 つとして検索の応答時間を向上させる方法がある．検索の応答時間を向上させる方法のひとつは，ログを分散配置して並列に検索する手法である [16]．

課題

迅速にシステム障害を復旧させるために，ログ検索の応答時間を短くする方法がある．ログ検索の応答時間を短くするために，並列で検索する手法がある．並列で検索するためにファイルの振り分けた数によって検索の分担を決めることができる．その際はファイルの振り分ける数によって検索の応答時間が異なる．検索の応答時間を短縮させるためにはファイルの振り分ける数を適切に設定しなければならない．

本稿の目的はシステム障害発生時に，検索の応答時間を短縮することでシステム管理者の原因調査の時間を短縮することである．実際の原因調査をする時間は例をあげると，94 分だった[*2]．その中でログ検索が関わる時間は調査と暫定対応の確認による 14 分であった．その中でログ検索のレスポンスが 20 ms で，10 回検索したとすると，200ms かかる．この 200ms を短縮することで，システム管理者の原因調査の時間を短縮する．ただし，この検索の応答時間の合計である 200ms は原因調査の時間である 94 分やログに関わる時間の 14 分に対して誤差であり，検索の応答時間を短縮することで原因調査の時間である 94 分や，ログに関わる時間も 14 分が変わらないため，効果が薄い．

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 章の実験結果と分析では，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

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 [17]．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．

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 [18]．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．

3. 提案

提案方式

本提案手法はログをログサーバーに保存する前にファイルを振り分ける数を変えて検索の応答時間を測定し，検索の応答時間が短くなるファイルの振り分ける数を決める．検索の応答時間は並列処理の時間，ファイルの読み込み性能とトレードオフの関係であると想定し，ログの振り分ける数を決めることで検索の応答時間を短縮する．また，並列の処理の時間とファイルの読み込み性能以外も検索の応答時間に影響があるが，この 2 つに着目して提案を構築している．

検索クエリとして最初にどんな異常なリクエストがあったかを調べるために，ステータスコード 400 番台，もしくは 500 番台の Web サーバーのアクセスログを調べるため，あらかじめ分けて検索する範囲を小さくする．そのため振り分ける数が決まった後はステータスコード 200 番台から 300 番台と 400 番台から 500 番台でアクセスログを振り分けする．ステータスコード 200 番台から 300 番台は正常系であり，400 番台から 500 番台で異常系である．

並列処理の検索を開始する時刻が一致していると仮定すると，検索の処理時間を均一にすることで，検索の応答時間に待ち時間が無くなり，並列処理における処理時間の待ち時間をなくす．検索の処理時間を均一にするためには各検索の検索する件数を同じにする．保存するログは振り分けする数に合わせて 1 件ずつファイルに追記して保存する．ログの件数を 1 件ずつに分けることで，各並列処理で検索する範囲の件数を 1 件以内の差にした．検索範囲の件数を揃えることで，各処理が完了するまでのタスク量を揃えて検索の応答時間に差ができなくなる．

検索は Index 検索を採用する．Index は索引であり，ステータスコードごとにどのファイルの位置に保存しているかを示している．Index 作成にあたり，例えばログは項目ごとに時刻とステータスコードごとに区切られて保存される．項目ごとに区切ることで検索条件に該当する項目のみを抜き出した検索できる．

図 1 はファイルの個数の決め方を示している．ログサーバーを構築した際の使用するディスクによって，Read 性能は異なる．そのため，ログサーバーごとにディスクの Read 性能に合わせたファイルの個数の決定が必要である．まず，ログファイルをステータスコードで分類する．200 番台と 300 番台を正常系のログとしてまとめる．同様に 400 番台と 500 番台を異常系のログとしてまとめる．400 番台と 500 番台を振り分けする数を増やしながらファイルを作

[*2] https://atmarkit.itmedia.co.jp/ait/articles/1705/09/news009_2.html

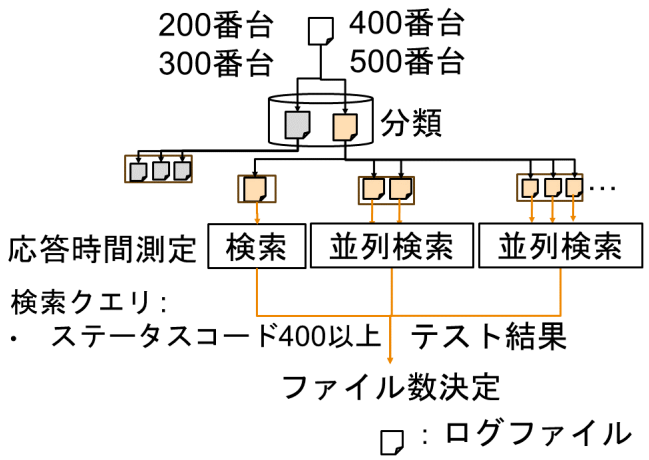

図 1 ファイルの個数の決め方

成する．作成したファイルで同じ検索クエリを実行し，並列検索することで検索の応答時間を測定するテストを実施する．テストでは EC サイトのアクセスログをユースケースとするため，実際の EC サイトのアクセスログを使用して実施する．テスト回数は検索した時間帯にノイズがあり検索の応答時間にばらつきがあると仮定して複数回計測する．検索のクエリはステータスコード 400 以上のクエリを実行する．実行後は応答時間を集計し，一番短い検索の応答時間を計測した振り分けする数に決定する．

ユースケース・シナリオ

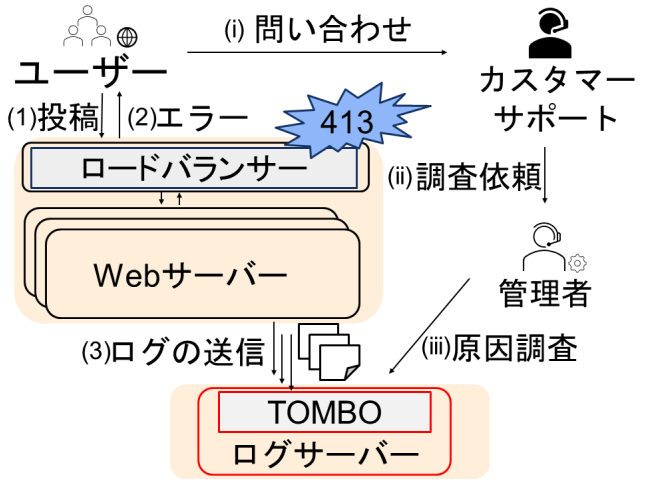

図 2 ユースケースシナリオ

図 2 は EC サイトにおけるシステム障害の対応を示している．Web サーバーでは EC サイトを構築している．スケーラビリティを高めることが目的として，EC サイトは複数台複製した EC サイトを起動している．Web サーバーとユーザーの間にはロードバランサーがあり，ユーザーからのリクエストを受け取っている．ロードバランサーから出力されたアクセスログを集約して保管する．ログサーバーには TOMBO と名付けた本提案手法のログ検索アプリケーションを起動する．登場人物として，EC サイトを閲覧するユーザーと，ユーザーからの問い合わせを受けるカスタマーサポートと，カスタマーサポートからの調査依頼を受けてログを検索して原因を調査する管理者がいる．

本ユースケースシナリオは EC サイトを運営しているシステム管理者を対象とする．システム管理者は EC サイトを安定して運用するために，EC サイトへのアクセスログを収集，監視を行う．

一例として，ユーザーのレビューの投稿に画像をアップロードする際にアップロードできないユーザーがいたとする．原因を調査するためにシステム管理者はログを検索する．画像をアップロードできなかった時間帯の Web サーバーのアクセスログを調べる．時間帯の指定はアップロードできなかった連絡を受け取り検索をし始めるまでの時間となるので，1 時間から起きた事象であれば，1 時間前から現在時刻までの Web サーバーのアクセスログを調べる．システム管理者はどのリクエストでクライアントエラーである 400 番台とサーバーエラーである 500 番台エラーが発生し，どのリクエストで発生しないのかを確認する．まず，1 時間以内の Web サーバーのアクセスログから 400 番台と 500 番台に絞って検索をする．ここからどのリクエストでエラーが発生しているのか条件を絞って特定していく．見る箇所は URI や Method となる．レビューの投稿に画像をアップロードする際にアップロードできないユーザーがいるため，検索条件の設定例として，投稿から POST リクエスト，レビューの投稿の URI から/review_post を設定することができる．そのため，検索クエリは期間が現在時刻から 1 時間前で，ステータスコードは 400 番台と 500 番台で，POST リクエストの URI が/review_post と条件を設定する．その中で，同じ URI と同じ Method でもエラーが出ているログとエラーが出ていないログがあった．例えばステータスコード 413 番エラーが出力されていたとする．413 は Request Entity Too Large であり，リクエストエンティティがサーバーによって定義された制限よりも大きいことを示している[*3]．エラーが出ているログと出ていないログを比較すると，送信しているファイルサイズが大きいものがエラーを出力していた．そこから，システム管理者はファイルサイズが大きいものをアップロードする際にエラーが起きていると推測する．

原因は Nginx で送信リクエストの上限を設定していて，上限を超えるリクエストは受け取らないようにしていた．Nginx はデフォルトで client_max_body_size と呼ばれるクライアントからのリクエストの受け取り最大サイズをデフォルトで 1MB と定義している．この設定を変更しない限りユーザーは 1MB を超えるリクエストをサーバーに送

[*3] https://developer.mozilla.org/ja/docs/Web/HTTP/Status/413

信することができない．Nginx の設定を変更することで画像がアップロードされることになる．また，ファイルサイズの制限を JavaScript でクライアント側で警告が出力されるように設定し，サーバー側でリクエストを受け取らない状況を未然に防ぐことで対応できる．

この様に，システム管理者は原因を調査するために，Web サーバーのアクセスログの分析を行う．Web サーバーのアクセスログの発生数は一日ごとにおいてもばらつきがあり，日にちごとにファイルに分かれて保存される．そのため，検索対象の範囲が広いとファイルごとに検索応答時間が異なり，落ちこぼれ問題が発生し検索応答時間が増加する．この問題に対して，提案システムである TOMBO を使用する．これにより，検索対象の Web サーバーのアクセスログがログサーバの CPU のコア数に応じて，Web サーバーのアクセスログを分割する．例えば CPU のコア数が 3 のとき，ログファイルを正常系，異常系にそれぞれ 3 つずつファイルを作り，ログを 1 件ずつ追記していく．分割された Web サーバーのアクセスログの中から，サーバ側のエラーが原因で発生する 500 番台の Web サーバーのアクセスログを検索する．分割先の Web サーバーのアクセスログ数が均等なため，落ちこぼれ問題が発生せずに検索を実施することが可能である．

4. 実装

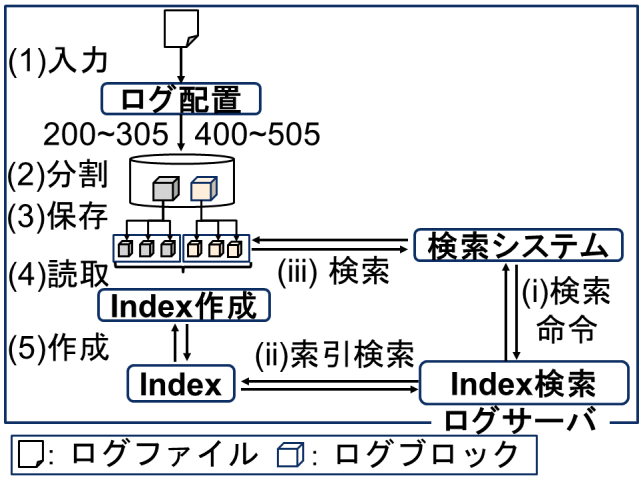

図 3 実装の全体図

図 3 は実装の全体図を示している．実装ではログの保存とログの検索の 2 つに分けられる．まず，ログの保存から説明する．流れとしては，ログを読み取り分割して保存した後に保存場所を index に記録する．以下に流れを箇条書きにして載せる．

(1) ログファイルの入力
(2) ステータスコードでログファイルを分割
(3) 1 件ずつ追記しながら保存してログブロックを作成
(4) index 作成のためにログブロックの読取
(5) ログの保存場所から index を作成

(1) ログファイルの入力はログサーバー内にあるログファイルを入力としてログ配置に読み込ませる．読み込ませるログファイルは Web サーバーのアクセスログとする．(2) ステータスコードでログファイルを分割はステータスコードの正常系と異常系に分ける．ステータスコード 200 番台から 300 番台は正常系であり，400 番台から 500 番台で異常系である．正常と異常を判別する前にログを TSV ファイルにパースする．パースは 10 項目に分ける．項目を列挙すると，IpId: アクセス元の IP アドレス，UserName: ユーザー名，TimeStamp: アクセスした時刻，HttpMethod: アクセスしたときのメソッド，Uri: アクセス先，HttpVersion: http のバージョン，ResponseCode: ステータスコード，Bytes: 送信した byte 数，Referer: Uri にアクセスする前のアクセス先，UserAgent: クライアントの OS やブラウザである．パースをすることで，ステータスコードがある場所を Tab 区切りの 7 番目と指定することができる．(3)1 件ずつ追記しながら保存してログブロックを作成はあらかじめ分けられたブロックにログを等しく分けるように 1 件ずつログを追記する．ログブロックはファイルとして保存する．ファイル名は正常の場合，normally_list000.tsv と名前を付けて保存する．異常の場合，abnormally_list000.tsv と名前を付けて保存する．次のファイルを作成するときは 000 の部分を加算し 001 のファイルを作成して，保存する．(4)index 作成のためにログブロックの読取は作成したブロックを読取り，index を作成する．index の構造はステータスコード，ブロック名（ファイル名），ファイルの何行目にそのステータスコードのログがあるかを示した配列という 3 階層の構造になっている．index は json 形式で保存する．インデックスは正常の場合，normally_index.json，異常の場合，abnormally_index.json で保存する．

次に，ログの検索を説明する．流れとしては，検索の命令を index 検索にし，index を検索し，結果からログを検索する．以下に流れを箇条書きにして載せる．

(i) 検索命令
(ii) 索引検索
(iii) 検索

(i) 検索命令は検索システムが Index 検索に検索命令を発行する．検索命令には Index の中から何を検索するかの命令を検索システムから Index 検索に発行する．(ii) 索引検索は検索システムからの検索命令に従って index を検索する．index は json ファイルでできているため，正常を調べる場合は normally_index.json，異常の場合，abnormally_index.json を開き，検索をする．検索クエリとしてステータスコードを受け取り，ブロック名と行数の配列を取得する．(iii) 検索は Index 検索で受け取った結果からログブロックを開き index からログブロックの行数から検索の該当結果を取り出して，検索結果とする．index を検索

した結果からブロック名と行数を取得しているため，ブロックを開いて該当する行数のみを取り出す．検索結果は該当するログをまとめて，最後に時刻順に並べて表示する．

Index は JSON 形式で作成する．abnormally_list_000.log と abnormally_list_001.log にログを保存している場合を例にあげる．ステータスコード 404 が abnormally_list_000.log の 0byte 目と 224byte 目，abnormally_list_001.log の 0byte 目と 409byte 目に保存していたとする．その場合の Index をソースコード 1 に示す．

ソースコード 1　Index

```
1 "404":{
2   "abnormally_list000.tsv":[0, 224],
3   "abnormally_list001.tsv":[0,409]
4 }
```

Index の階層は 3 つに分かれている．1 つはステータスコードである．まず，検索のときはこの階層を確認することで，ステータスコードごとにログがどこに格納されているかを確認することができる．2 つ目の階層はファイル名である．ステータスコードごとに分かれている 1 つ目の階層に対して 2 つ目の階層は該当するステータスコードを持つログのファイル名を格納している．3 つ目の階層は byte 数である．これはファイルの何 byte 目に該当するログの始まりの文字があるかを示している．検索時はこの 3 つ目の階層の byte 数を用いて最初の文字を指定して改行までをログの 1 件として扱い，検索結果の配列に格納する．この 3 つの階層によってステータスコードの Index を構成している．

5.　評価実験

実験環境

EClog の中から，2020 年 3 月 16 日のログからステータスコードの 400 番以上のログを検索する実験をする．400 番以上のログの検索は 160,446 件から 604 件を探し出す検索である．ファイルを 1 つに分けた際，2 つに分けた際と数を増やしていき，検索の応答時間を比較する．また同様の仮想マシンを作成し，仮想マシンは Ubuntu22.04 をインストールした．仮想マシンの性能は vCPU: 3[core], RAM:4[GB]，SSD:30[GB] で作成した．

Elasticseach は ECK を用いて Kubernetes 上で起動する．

実験結果と分析

以下に提案システムを実行したときの検索の応答時間を図 5 に載せる．EClog の中から，2020 年 3 月 16 日のログからステータスコードの 400 番以上のログを検索する実験をした．2020 年 3 月 16 日のログは 160,446 件であり，ステータスコード 400 番以上のログは 604 件である．この数値は Elasticsearch で集計して取得した．ファイルを 1 つに分けた際から 10 つに分けた際まで比較する．それぞれ 10 回計測し，平均値を実験結果とする．ファイルの振り分ける数に合わせてプロセスの数も増やしている．

以下に 2024 年 2 月に実施した提案システムである TOMBO を実行したときの検索の応答時間を図 4 に載せる．

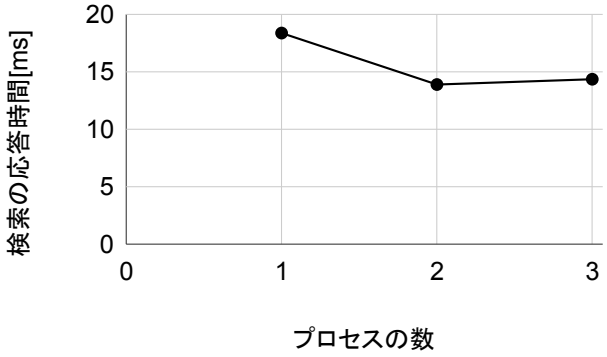

図 4　TOMBO で実行したときの検索の応答時間の比較

ファイルを 1 つに分けた際，2 つに分けた際，3 つに分けた際を比較する．それぞれ 10 回計測し，平均値を実験結果とする．

縦軸は検索の応答時間であり，単位は ms である．横軸は分けたファイルの数を示している．ファイル 1 つのときは約 18 ms，2 つのときは約 13 ms，3 つのときは約 14 ms となった．検索の応答時間はファイル 1 つのときとファイル 2 つのときを比較すると減少している．ファイルを 2 つに分けたときからファイルを 3 つに分けたときまで，検索の応答時間は微増していることが分かる．

以下に 2024 年 3 月に実施した提案システムである TOMBO を実行したときの検索の応答時間を図 5 に載せる．

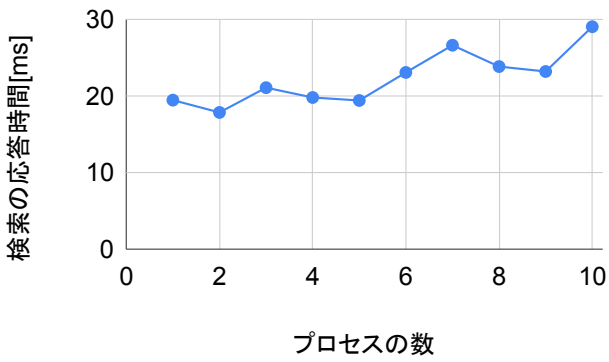

図 5　さらにプロセスの数を変えて実行したときの検索の応答時間の比較

縦軸は検索の応答時間であり，単位は ms である．横軸は分けたファイルの数を示している．プロセスの数が 1 つから 5 つのときまでは検索の応答時間が 20ms 前後で推移していることが分かる．プロセスの数が 6 つ以降になると 25ms 前後での推移となった．

検索の中でも検索の応答時間にばらつきがあり，最小で約 15 ms，最大で 38ms の検索の応答時間であった．プロセスの数に着目すると検索の応答時間は 3 つに分けた際が最小で 15 ms，最大で 26 ms となり，11ms の差がある．

実験の時期が異なることで検索の応答時間と傾向が異なることがわかった．ESXi で VM を起動しているため，プロセスの実行の優先度が ESXi の使用状況によって異なるため，誤差として出ている．

Elasticsearch の検索の応答時間を測定した結果を図に載せる．

6. 議論

本提案手法はログを保存する前にファイルの振り分ける数を変えて検索の応答時間を測定し，検索の応答時間が短くなるファイルの振り分ける数を決める．このログを保存する前の検索の応答時間の測定をする際に，ファイルの振り分ける数をどこで止めるかが問題になる．まず，ファイルの振り分ける数を N とおく．次に検索の応答時間を S とおく．N を 1 から試していき，検索時間が $S_N = S_{N+1}$ となった際にファイルの振り分ける数を増やして検索の応答時間の測定を止める．振り分ける数を N から 1 つ増やした際に検索の応答時間が長くなる場合，N+1 以降の検索の応答時間は上昇傾向にあると予想できるためである．

7. おわりに

課題は検索の応答時間を短縮させるためにはファイルの振り分ける数を適切に設定しなければならないことである．本稿の目的はシステム障害発生時に，検索の応答時間を短縮することでシステム管理者の原因調査の時間を短縮することである．本提案手法はログをログサーバーに保存する前にファイルを振り分ける数を変えて検索の応答時間を測定し，検索の応答時間が短くなるファイルの振り分ける数を決める．検索の応答時間は並列処理の時間，ファイルの読み込み性能とトレードオフの関係であると想定し，ログの振り分ける数を決めることで検索の応答時間を短縮する．実験では Harvard Dataverse に載っている EClog の 2020 年 3 月 16 日の HTTP アクセスログを用いてステータスコード 400 以上の検索の応答時間を比較した．

謝辞 本テクニカルレポート執筆にあたり，この提案ソフトウェアの名前をつけてくださり，執筆の手伝いをして頂いた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 4 年の増田 和範さん，アドバイスを頂きました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 4 年の大沢 恭平さん，平尾 真斗さん，三上 智徳さん，圖齋 雄治さん，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 1 年の高橋 風太さんに心から感謝します．本当にありがとうございました．

参考文献

[1] 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 (2018).

[2] Suneetha, K. and Krishnamoorthi, R.: Identifying user behavior by analyzing web server access log file, *IJCSNS International Journal of Computer Science and Network Security*, Vol. 9, No. 4, pp. 327–332 (2009).

[3] Singh, S. P. and Meenu: Analysis of web site using web log expert tool based on web data mining, *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pp. 1–5 (online), DOI: 10.1109/ICIIECS.2017.8275961 (2017).

[4] Özgit, A., Dayioglu, B., Anuk, E., Kanbur, I., Alptekin, O. and Ermis, U.: Design of a Log Server for Distributed and Large-Scale Server Environments., Vol. 2869, pp. 885–891 (2003).

[5] Noorhuzaimi, M., Junaida, S. and Mohamad Mazrul, R.: An analysis of network services using association rules, *2009 2nd IEEE International Conference on Computer Science and Information Technology*, pp. 469–473 (online), DOI: 10.1109/ICCSIT.2009.5234905 (2009).

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

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

[8] 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).

[9] Kononenko, O., Baysal, O., Holmes, R. and Godfrey, M. W.: Mining modern repositories with elasticsearch, *Proceedings of the 11th working conference on mining software repositories*, pp. 328–331 (2014).

[10] AERIS-Consulting: Elasticsearch, *software, version*, Vol. 6, No. 1 (2018).

[11] Gupta, Y.: *Kibana essentials*, Packt Publishing Ltd (2015).

[12] Berners-Lee, T., Fielding, R. and Frystyk, H.: Hypertext transfer protocol–HTTP/1.0, Technical report (1996).

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

[14] 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).

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

[16] Hamadi, Y.: Distributed interleaved parallel and cooperative search in constraint satisfaction networks, *Proc. IAT*, Vol. 1 (2001).

[17] He, J., Wu, K., Kannan, S., Arpaci-Dusseau, A. and Arpaci-Dusseau, R.: Read as Needed: Build-

ing 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).

[18] 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).