分散検索手法におけるアクセスログの検索の応答時間の削減

大野 有樹[1]　小山 智之[1]　串田 高幸[1]

概要：Web アプリケーションにおいて，アクセスログはシステム障害が発生した時に根本の原因を分析するのに役に立つ．システム障害の解決のためのログ検索は短い応答時間が必要になる．なぜなら，システム障害を分析している時間はシステムの利用者がシステムを使えない時間になるためである．検索の応答時間は，ログの件数が多いほど長くなる．ログの単位時間あたりの件数が多くなった場合に検索対象のログ件数が増える．そのため，検索にかかる時間が長くなる．提案では検索をする時にアクセスする検索対象のログ件数を減らすことで検索の応答時間を短くする．具体的にはアクセスログをアクセス先，ステータスコードでグループ化されたブロックとして保存する．検索クエリはアクセス時刻，ステータスコード，アクセス先の中から 1 つを指定するか 2 つ，もしくは全てを指定したものとする．評価方法は提案手法とアクセス時刻のみでグループ化した手法と検索の応答時間を比較する．

1. はじめに

背景

ログはシステムの動作を記録したものである [1]．システム管理者はシステムの障害発生時にログからシステムの動作を確認して原因を調べるために使用する．

Web サービスでダウンタイムが発生すると収益損失が発生する．具体例として，Amazon は 2015 年に 13 分のダウンタイムが発生した．Amazon の 2015 年の収益は 1,070 億ドルのため，13 分で約 265 万ドルの収益損失が発生している．そのため Web サービスは 24 時間 365 日稼働することが要求され，年々規模は大きくなり，構造も複雑化している [2]．

Web アプリケーションにおいて，アクセスログはシステム障害が発生した時の原因を分析するのに役に立つ．アクセスログにはステータスコードがある．ステータスコードは 400 番台はクライアントエラー，500 番台であればサーバーエラーが出ていることを示している [3]．

検索の応答時間は検索のクエリを発行してから検索結果を出力するまでの時間とする．このとき，システム障害の解決のためのログ検索は短い応答時間が必要になる．なぜなら，システム障害を分析している時間はシステムの利用者がシステムを使えない時間になるためである．ログ検索の応答時間の削減はシステムの利用者がシステムを利用できない時間の削減に繋がる．検索の応答時間は，ログの件数が多いほど長くなる．検索の応答時間を短くするために複数ノードを用いて分散処理する手法がある．

課題

課題はログの検索対象の件数が多くなったときに，検索の応答時間が長くなることである．例えば，検索時にデータベースのソフトウェアである Elasticsearch は全てのログのレコードそれぞれと検索条件が一致しているかどうか確かめ，一致している場合は検索結果として出力する．ログはログファイルから構造化されてデータベースにレコードとして保存される．ログのレコードはストレージに保存されているため，検索時にデータベースのソフトウェアはストレージからレコードを読み込んでいる．ストレージに保存されたデータを読み書きする方が，メモリに展開された同じ量のデータを読み書きするよりも時間がかかる [4]．そのため，ログの検索対象の件数が多くなった場合に，検索の応答時間が長くなる．

基礎実験

基礎実験では，ログの件数が増えるほど検索の応答時間が増えることを調べる．検索の応答時間はターミナルで検索クエリを発行してからターミナルで検索結果が出力されるまでとする．EC サイトのアクセスログは HARVARD Dataverse より EClog を使用した [5]．EClog を出力した Web サイトは PHP7.0 で実装され，MySQL データベースを使用する Linux 上で Apache の Web サーバーをホストしている．EClog は 2019 年 12 月 1 日から 2020 年 6 月 1 日までの 183 日間の Web サーバーのアクセスログが 35,157,691[件] が記録されている．検索の範囲を 1 ヶ月分の 7,175,241[件]，2 ヶ月分の 12,735,845[件]，3 ヶ月分

[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

の 17,997,264[件]，4 ヶ月分の 23,183,227[件]，5 ヶ月分の 28,746,880[件]，6 ヶ月分の 35,157,691[件] とログの件数を増やしながら検索の応答時間を計測した．

実験は Elasticseach を起動している VM から Elasticsearch の API へ curl コマンドを用いて Web サーバーのアクセスログを検索する．検索クエリは URL ごとにステータスコードを集計するものを使用する．検索の応答時間は 100 回計測し，それぞれ最小値，最大値，平均値，中央値を取得した．時間の計測は curl の-w オプションを使用した．計測の度に Elasticsearch のキャッシュを API で削除した．検索に使用した VM のスペックは vCPU 3[Core] と RAM 4[GB] である．

以下に検索クエリの例を示す．例 1 は 1 ヶ月分である，2020 年 1 月 1 日までの範囲で検索したときを示している．

```
curl -s -o /dev/null -w '%{json}' -XGET
↪ 'http://localhost:9200/koyama/_search'  -H
↪ 'Content-Type: application/json' -d '
{
    "query": {
        "range": {
            "TimeStamp": {
                "lte": "2020/01/01"
            }
        }
    },
    "aggs": {
        "interests": {
            "terms": {
                "field": "Uri.keyword"
            },
            "aggs": {
                "group_by_state": {
                    "terms": {
                        "field":
↪ "ResponseCode.keyword"
                    }
                }
            }
        }
    }
}'
```

Listing 1: 検索クエリの例

検索の応答時間の測定結果を図 1 に示す．縦軸は検索の応答時間を示している．縦軸の単位は秒である．横軸は件数を示している．横軸の単位は 1,000,000 件である．凡例は 4 つあり，最小値，最大値，平均値，中央値がある．最小値をみると，1 ヶ月分から 6 ヶ月分にかけて，0.94[秒]，1.41[秒]，1.87[秒]，2.32[秒]，2.78[秒]，3.38[秒] と線形増加している．検索の応答時間は最大値を除いて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った．

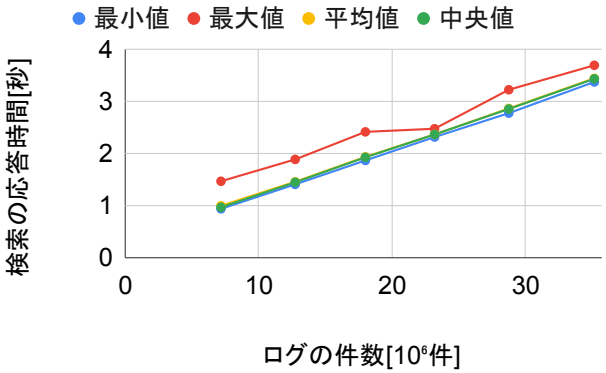

図 1　ログの件数を変えたときの検索の応答時間の変化

各章の概要

2 章では関連する既存研究を述べる．3 章では提案手法を説明する．4 章では実装方法と実験方法を述べる．5 章は，評価と分析の手法を説明する．6 章では提案手法の議論をする．最後，7 章は全体を簡潔にまとめている．

2. 関連研究

分散手法である scatter-gather パターンの検索時にデータのクラスタを作成し二分木で検索の応答時間を削減しようとした研究がある [6]．既存手法である LAIR2 の課題は，分散するクラスタを選択するたび二分木の階層が 1 番初めから開始するため，データセットが大規模になるほど検索の応答時間が遅くなることである．この論文では，二分木の検索の該当結果が含まれていない方の下層へ検索するアルゴリズムと二分木で検索できるようにデータのクラスタを作成するアルゴリズムを提案している．しかし結果は，検索精度が上がったものの，検索の応答時間は長くなっていた．

分散クエリ処理のためにどこに検索対象があるかのパスを検索するアルゴリズムの研究がある [7]．この研究における提案手法のパス検索のオーバーヘッドをどれだけ削減できたかを述べた論文である．この研究で提案した RippleLog は，パス検索の遅延のみ考慮しているため，ログ検索の応答時間全体まで考慮されていない．

3. 提案方式

提案方式

検索をする時にアクセスする検索対象の数を減らすことで検索の応答時間を短くする．検索対象の減らし方はアクセスログをステータスコードと URL でグループ化する．グループ化はステータスコードと URL が一致するアクセスログをまとめてブロックとして扱う．本提案方式ではブロック単位でログを保存し，ブロック単位でログを検索する．

図 2 は保存の全体図を示している．サーバーは Web サー

バーとログサーバーがある．ユーザーが Web サーバーにアクセスしたあとに，ログサーバーへ保存する流れを説明する．ユーザーは Web サーバーへアクセスするときに

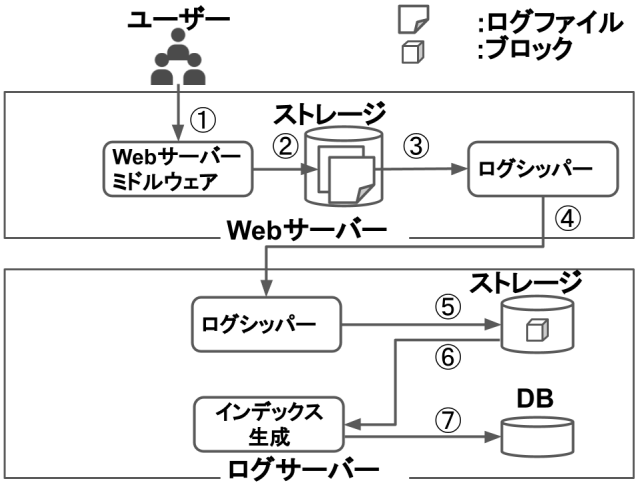

図 2　保存の全体図

Web サーバーへアクセスする．Web サーバーミドルウェアはログをストレージに出力する．Web サーバーにあるログシッパーはストレージにあるログの出力を検知し，ログサーバーへログを転送する．ログサーバーにあるログシッパーは Web サーバーから転送されたログを受け取る．ログサーバーにあるログシッパーはログサーバーのストレージにブロックとして保存する．提案ソフトウェアの 1 つであるインデックス生成はストレージからログブロックの配置を読み取り，配置場所を記録する．インデックス生成は配置場所の記録先として DB に保存する．

図 3 はログの保存する際におけるブロックの作成方法を示している．図 3 は図 2 における Web サーバーからのログをログサーバーで受け取る場面を起点にした図を示している．まず，ログシッパーは Web サーバーからのログを受け

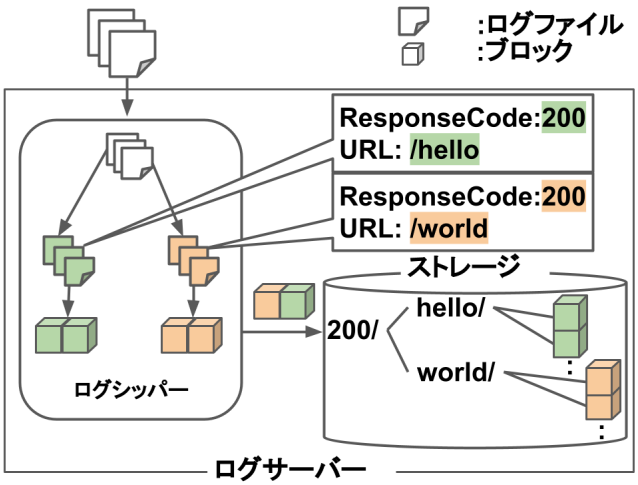

図 3　ブロックの作成方法

取る．ログシッパーは内部でブロックを作成する．ブロックはステータスコードとアクセス先の URL が一致するログをまとめて作成される．ログシッパーは作成したブロックをストレージに保存する．ストレージは 2 階層のディレクトリ構造を持っている．第 1 階層はステータスコードごとに分かれている．第 2 階層はアクセス先の URL ごとに分かれている．ログシッパーは対応するステータスコードとアクセス先の URL が一致しているブロックをディレクトリ構造に従って配置する．

図 4 はログの保存先を示すインデックスの作成方法を示している．図 4 は図 2 における Web サーバーからのログをログサーバーで受け取る場面を起点にした図を示している．まず，図 3 と同様に Web サーバーからのログをログシッ

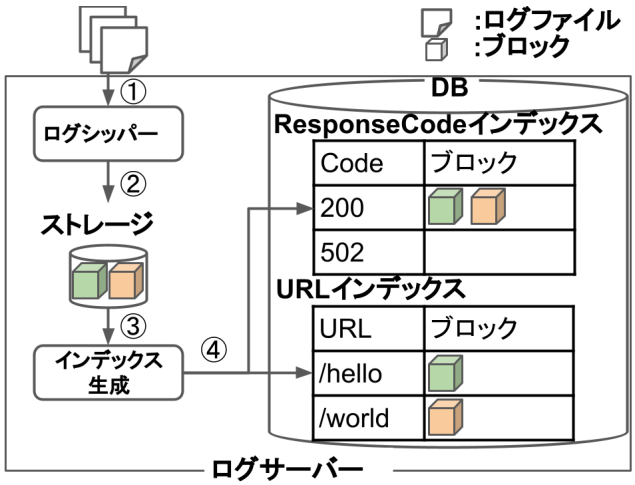

図 4　インデックスの作成方法

パーで受け取り，ストレージに保存する．保存したログのブロックからインデックスを生成する．インデックスは検索のときにログの配置場所を明らかにし検索の範囲を削減するために作成する．インデックスは ResponseCode インデックスと URL インデックスに分かれる．ResponseCode インデックスは DB に該当するステータスコードを含むブロックの配置場所を記録している．URL インデックスも同様に DB に該当するアクセス先の URL を含むブロックの配置場所を記録している．

図 5 は検索方法を示している．まず，システム管理者はログサーバーの検索ソフトウェアに検索クエリを発行する．検索ソフトウェアはインデックスに検索をしてブロックの配置場所を調べる．検索ソフトウェアは調べたブロックの配置場所をもとにストレージからブロックを取得する．検索ソフトウェアは取得したブロックを検索結果としてシステム管理者へ送信する．

ユースケース・シナリオ

本ユースケースシナリオでは，Web サイトへアクセスできない時にシステム管理者がログ検索をして原因を特定する．システム管理者はアクセスログの 400 番以上の URL

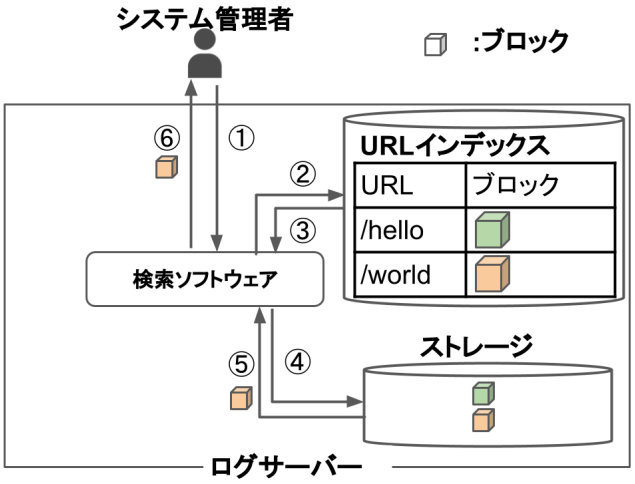

図 5 検索方法

を調べ，Web サイトのどの部分にアクセスできないかを調査する．その後，URL ごとのステータスコードをカウントして調べることでその URL にアクセスするユーザー全てにエラーが出ているか，一部だけかを判断し，障害解消に向けての対処方法を絞り込む．

図 6 は，ユースケースシナリオを示している．サーバーは Web サーバー，監視サーバー，ログサーバーがある．Web サーバーは Web サイトを構築しているサーバーである．監視サーバーは Web サーバーのアクセスの可否を確認する．ログサーバーは Web サーバーから出力されたログをログサーバー内に保存して検索をできるようにする．ユーザーは Web サーバーにアクセスして Web サイトを閲覧する．システム管理者は Web サーバー，監視サーバー，ログサーバーを管理し，Web サイトを運営する．(1) と (2) はアクセスログが保存する流れを示している．(1) はユーザーからの EC サイトへのアクセスを示している．リクエスト数は 10,000[requests/分] とする．(2) はユーザーがリクエストするたびにアクセスログとして HTTP アクセスログをログサーバーへ送信している．検索のシナリオの順番を以下に説明する．

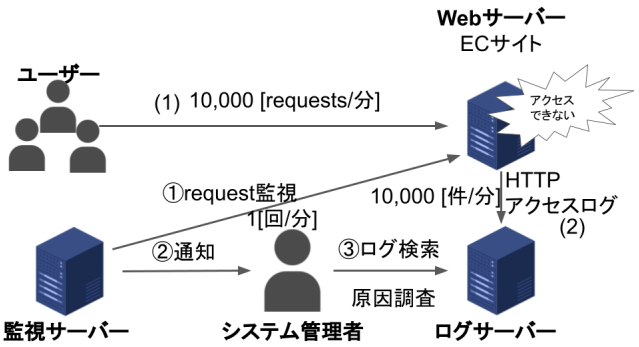

図 6 ユースケースシナリオ

①から③はシステム管理者がログを検索する流れを示している．まず，監視サーバーは Web サイトのアクセス可否を常に監視しているため，Web サイトにアクセスできなくなったときシステム管理者へ通知をする．通知を受けたシステム管理者は Web サイトへアクセスできるようにするために原因調査をする．システム管理者はログサーバーへステータスコード 400 番台と 500 番台を条件に Web サーバーのアクセスログを検索する．ステータスコード 400 番台はクライアントエラーを示している．また，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 [3]．システム管理者は 400 番台と 500 番台エラーのアクセスログが発生している Web サイトの URL を特定し，Web サイトの障害が起きている影響範囲を特定する．

4. 実装と実験方法

実装

図 7 は実装の全体図を示している．実装のソフトウェアは 3 つあり，ログシッパーとインデックス生成と検索ソフトウェアである．このうちログシッパーは既存のソフトウェアである Fluentd を使用する．インデックス生成と検索ソフトウェアは Pyhton3 で実装する．以下に実装するソフトウェアの流れを説明する．(1) から (4) はログファ

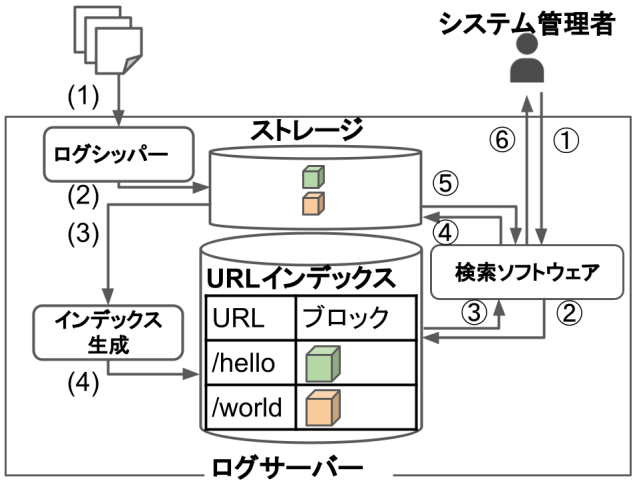

図 7 実装の全体図

イルの保存を示している．(1) はログシッパーが Web サーバーからログファイルを受け取っている．(2) はログシッパーがログファイルをステータスコードと URL ごとに分類した後，ストレージに保存している．(3) はインデックス生成が (2) でストレージに書き出されたブロックを読み出し，保存場所を確認する．(4) はインデックス生成が保存場所をインデックスとして保存する．

①から⑥はログ検索を示している．①はシステム管理者が検索ソフトウェアへ検索クエリを発行している．②は検索ソフトウェアがインデックスへ該当するブロックがどこに保存しているかを調べている．③はインデックスからブロックの保存場所を読み取っている．④は検索ソフト

ウェアが③で読み取ったストレージの保存場所へアクセスしている．⑤は④で読み取った保存場所からブロックを取得する．⑥は⑤で検索ソフトウェアはブロックごとの検索結果を結合してシステム管理者へ返している．

実際に扱う Web サーバーのアクセスログの具体例として研究室のブログサイト[1]から取得したアクセスログの例を例 2 に以下に示す．このアクセスログにおけるステータスコードは 502 である．

```
60.35.59.137 - - [07/Mar/2021:15:59:54 +0900] ”GET /2020-ug3/ HTTP/2.0” 502 0 ”https://accounts.google.com/” ”Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36”
```

Listing 2: アクセスログの例

実験環境

Kubernetes クラスタを ESXi の VM を使用して作成する．Kubernetes の Pod の数でノードの数による検索の応答時間の変化を再現する．ノードの再現をするため，Pod は CPU とメモリに limit を設定する．Pod に limit を設定することで，CPU とメモリを制限し，並列性による検索の応答時間の変化を計測することができる．ストレージは NAS を使用する．

5. 評価手法と分析手法

ノード数を Pod 数の変化によって再現し，アクセスログを保存するときにグループ化の仕方によって検索の応答時間がどれだけ変化するかを比較して評価する．使用するログは EClog と CDSL のアクセスログ[1]，1997 年のサッカーワールドカップの Web サイトのアクセスログである [5,8]．

6. 議論

本提案方式ではインデックスを URL とレスポンスステータスコードの 2 つのみ作成した．検索クエリで指定できる条件は URL とレスポンスステータスコードだけではない．アクセスログは他にもアクセス元の IP アドレス，プロトコル，リファラーがある．URL とレスポンスステータスコード以外の条件を指定した場合，インデックスが無いため検索の応答時間が低下する．そのため，インデックスを作成することで，応答時間の低下を防ぐ．

7. おわりに

ログの単位時間あたりの件数が多くなった場合に検索対象のログ件数が増える．そのため，検索にかかる時間が長くなる．解決方法として，インデックスによる検索範囲の削減をすることで検索の応答時間を削減した．

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 (2017).
[2] He, S., Zhu, J., He, P. and Lyu, M. R.: Loghub: a large collection of system log datasets towards automated log analytics, *arXiv preprint arXiv:2008.06448* (2020).
[3] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T.: Hypertext transfer protocol–HTTP/1.1, Technical report (1999).
[4] Cambazoglu, B. B., Plachouras, V. and Baeza-Yates, R.: Quantifying Performance and Quality Gains in Distributed Web Search Engines, *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR ’09, New York, NY, USA, Association for Computing Machinery, p. 411–418 (online), DOI: 10.1145/1571941.1572013 (2009).
[5] Chodak, G., Suchacka, G. and Chawla, Y.: EClog: HTTP-level e-commerce data based on server access logs for an online store (2020).
[6] Farsandaj, K., Ding, C. and Sadeghian, A.: A new approach to improve the accuracy of online clustering algorithm based on scatter/gather model, *2010 Annual Meeting of the North American Fuzzy Information Processing Society*, pp. 1–5 (online), DOI: 10.1109/NAFIPS.2010.5548181 (2010).
[7] Zhang, J., Wang, J. and Yang, D.: RippleLog: A Path Search Algorithm for a Distributed Query Processing, *2009 Fifth International Conference on Natural Computation*, Vol. 5, pp. 578–582 (online), DOI: 10.1109/ICNC.2009.350 (2009).
[8] 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 (online), DOI: 10.1109/65.844498 (2000).

[1] クラウド・分散システム研究室, `https://ja.tak-cslab.org/`, 2022 年 12 月 05 日