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

大野 有樹[1]　小山 智之[1]　串田 高幸[1]

概要：システム管理者はシステムの障害発生時にログを検索することで原因を調査する．検索のクエリによって検索の応答時間は異なる．応答時間が異なる原因はログの配置やインデックスにある．課題はシステムの原因調査のために月単位でログの検索をする際に，検索対象の件数が多くなるために，検索の応答時間が長くなることである．例えば，障害の影響範囲を調べるために URL ごとのステータスコードの集計をした際の検索の応答時間は，EClog を検索対象とした場合，1 ヶ月分の 7,175,241[件] では 1.52[秒] であったが，6 ヶ月分の 35,157,691[件] では 5.00[秒] となった．解決方法として，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 HTTP アクセスログを分けて保存する．共通したステータスコードと URL で HTTP アクセスログを分けることで，検索対象となるログの中から検索条件に合致するログを予め用意することができる．検索条件に合致するログの保存場所が分かることで，検索のためにログのレコードが検索条件に合致するかを調べる必要がなくなり，検索時間の短縮が可能になる．評価手法として，Elasticsearch と改善した提案手法とで検索の応答時間を比較した．検索の応答時間は検索対象が 1 ヶ月分の 7,175,241[件] で 2.0[秒]，6 ヶ月分の 35,157,691[件] で 16.2[秒] の検索の応答時間の増加があった．Elasticsearch に比べて改善した提案手法は 1 ヶ月から 6 ヶ月分の検索の応答時間を比較すると，平均で 438[%] 上昇した．

1. はじめに

背景

ログはシステムの動作を記録したテキストである [1]．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．検索の例として．システム障害の発生している箇所の特定をする場合を想定する．システム障害の影響範囲を調べるときに，HTTP アクセスログのステータスコード 500 番台が出力されているログを調べる．HTTP 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 [2]．システム管理者はステータスコード 500 番台のエラーが出力されている URL を調べることでシステム障害の影響範囲を調べることができる．

ログを検索するシステムの中に全文検索システムがある．全文検索システムにおいてインデックスは検索の応答時間を短くするために用いる．ここでは検索の応答時間をシステム管理者が検索クエリを発行してから検索結果が出力されるまでとする．インデックスは検索対象の保存場所を記録しているため，ログを検索するシステムが検索時に検索結果に必要なログへ直接アクセスすることができる．直接アクセスすることで保存しているレコードを 1 件 1 件探す必要がなくなり，検索の応答時間の削減に繋がる．

Web サーバーから出力されたログはログファイルに保存される．Web サーバーにあるログシッパーはログファイルへのログ追加を監視する．ログがログファイルに追加された場合に，ログシッパーはデータベースサーバーへログを転送する．Web サーバーから出力されたログは構造化されたレコードとしてデータベースに保存される．データベースのレコードはストレージに保存されている．データベースのソフトウェアはレコードをストレージから読み込み検索をしている．メモリに展開されたデータを読み書きするより，ストレージに保存されたデータを読み書きする方が時間がかかる [3]．

ログは、システムの動作を記録しているため，システム障害の根本原因を突き止めるのに役立つ．システム障害を解決するためにログをすばやく検索できることは重要である[*1]．

課題

課題はシステムの原因調査のために月単位の範囲でログ

[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.sentinelone.com/blog/how-search-log-files-extract-data/

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．例えば，オンライン決済サービスである Stripe を想定する．このとき，Stripe の設定変更により，Stripe からシステムへの呼び出しに障害が発生した際にログ検索をする．Stripe の変更は 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月単位でログを検索する [4].

基礎実験

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を示す．検索の応答時間は検索クエリをターミナルで発行してから検索結果をターミナルで出力するまでとする．実験で使用するログは EC サイトのアクセスログで，HARVARD Dataverse に提供されている EClog である [5]．EClog を出力した Web サイトは Linux 上で Apache の Web サーバーで，PHP7.0 で実装され，MySQL データベースを用いている．EClog は 35,157,691[件] の Web サーバーのアクセスログが記録されている．期間は 2019 年 12 月 1 日から 2020 年 6 月 1 日までの 183 日間である．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．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．

実際に扱う Web サーバーのアクセスログの具体例として EClog から取得したアクセスログの例をソースコード 1 に示す [5].

ソースコード 1 アクセスログの例

```
1PL - 2019/11/30 23:00:00 GET /p-9710.html HTTP
    /1.1 200 14226 https://shop.our-internet-
    company.pl/p-3880.html?gclid=
    Cj0KCQiAw4jvBRCJARIsAHYewPNbsjL
    CAycxbJdVeMl_Q4qaH-5
    qh48gij96mitVce4INSRXhgRznyQaAg40EALw_wcB
    Mozilla/5.0 (Windows NT 6.1) AppleWebKit
    /537.36 (KHTML, like Gecko) Chrome
    /78.0.3904.108 Safari/537.36
```

EClog は 10 の項目に分かれたフォーマットを持っている．

- IpId: アクセス元の IP アドレス
- UserName: ユーザー名前
- 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

IpId は匿名化されたアクセス元の IP アドレスである．このアクセスログにおける IpId は 1PL である．UserName はアクセスしているユーザーのユーザー名である．このアクセスログにおける UserName は値がない．TimeStamp はアクセスした時刻を記録している．このアクセスログにおける TimeStamp は 2019/11/30 23:00:00 である．HttpMethod はアクセスしたときのメソッドである．このアクセスログにおける HttpMethod は GET である．Uri は匿名化されたアクセス先を示している．このアクセスログにおける Uri は/p-9710.html でありURL を示している．HttpVersion はアクセスした http のバージョンを示している．このアクセスログにおける HttpVersion は HTTP/1.1 である．ResponseCode はステータスコードを示している．このアクセスログにおける ResponseCode は 200 である．Bytes はアクセスしたときに送信した byte 数を示している．このアクセスログにおける Bytes は 14226 である．Referer は Uri にアクセスする前のアクセス先を示している．このアクセスログにおける Referer は https://shop.our-internet-company.pl/p-3880.html?gclid=Cj0KCQiAw4jvBRCJARIsAHYewPNbsjLCAycxbJdVeMl_Q4qaH-5qh48gij96mitVce4INSRXhgRznyQaAg40EALw_wcB である．UserAgent はアクセスしてきたクライアントの OS やブラウザを示している．このアクセスログにおける UserAgent は Mozilla/5.0 (Windows NT 6.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.108 Safari/537.36 である．

実験は Elasticsearch の API を用いて行う．検索対象は Web サーバーのアクセスログの EClog である．Elasticsearch に Fluentd を用いて EClog を投入した．計測は 100 回行い，検索の応答時間の最大値と最小値を記録した．検索に用いるコマンドは curl の-w オプションとする．検索のたびに Elasticsearch の API を用いてキャッシュを削除する．Elasticsearch を起動している VM は vCPU 3[Core] と RAM 4[GB] のスペックで実験する．

検索クエリは 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．

表 1 ステータスコードごとに URL を集計した検索結果の例

ステータスコード	URL	件数
200	/inne/informacja_online.php	913,395
200	/javascript/skrypty.php	164,650
502	/inne/informacja_online.php	69
502	/inne/zakladki.php	2

検索結果の例を表 1 に示す．ステータスコードは ResponseCode であり，表 1 は 200 の OK と 502 の Bad Gateway がある．200 の OK はリクエストが成功した

ことを示し，502 の Bad Gateway はゲートウェイとして動作するサーバーが無効なレスポンスを受け取ったことを示している．URL はアクセス先であり，/inne/informacja_online.php と/javascript/skrypty.php，/inne/zakladki.php がある．件数は検索結果の件数を示している．ステータスコードが 200 かつ URL が/inne/informacja_online.php の場合に件数は 913,395 件である．ステータスコードが 200 かつ URL が/javascript/skrypty.php の場合に件数は 164,650 件である．ステータスコードが 502 かつ URL が/inne/informacja_online.php の場合に件数は 69 件である．ステータスコードが 502 かつ URL が/inne/zakladki.php の場合に件数は 2 件である．

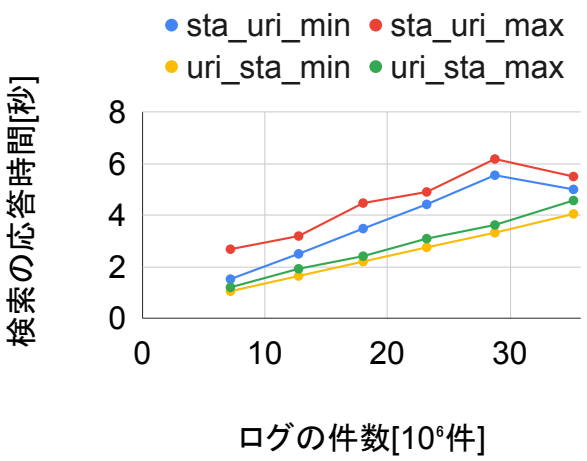

図 1 集計するクエリを変更した際の検索の応答時間の比較

図 1 は基礎実験の結果を示している．X 軸は件数を表し，単位に 1,000,000 件をとる．Y 軸は検索の応答時間を表し，単位に [秒] をとる．凡例は 4 つあり，sta_uri_min, sta_uri_max, uri_sta_min, uri_sta_max がある．sta_uri_min はステータスコードごとの URL の集計の検索の応答時間の最小値を示している．sta_uri_max はステータスコードごとの URL の集計の検索の応答時間の最大値を示している．uri_sta_min は URL ごとのステータスコードの集計の検索の応答時間の最小値を示している．uri_sta_max は URL ごとのステータスコードの集計の検索の応答時間の最大値を示している．

6 ヶ月分の検索結果を除き，検索の対象となるログの件数が多くなるほど検索の応答時間が増加することが分かる．

各章の概要

第 2 章関連研究では関連する既存研究を述べる．第 3 章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 章実装では実装方法について述べる．第 5 章実験と分析は，実験環境と実験結果，分析について説明する．第 6 章議論では提案手法について議論をする．最後に，第 7 章結論では全体を簡潔にまとめている．

2. 関連研究

Lars Arge らは，線形空間を使用した B-tree をベースにしている I/O 効率の良いデータ構造を提案している [6]．このような構造はすべて、構造内の要素に全順序を仮定しているが，本論文ではこの仮定を解除している．

Jon Louis Bentley は，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 [7]．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の応答時間について考慮していない．

Surajit Chaudhuri らは，SQL データベースの物理設計におけるインデックスの選択を自動化するためのツールを提案した [8]．このツールは，SQL クエリのワークロードを入力とし，適切なインデックスのセットを提案する．この論文は提案手法に関して検索の応答時間の評価をしていない．

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，検索の応答時間を削減する．図 2 は提案の全体図である．システム管理者とユーザー，ログサーバー，Web サーバーがあり，システム管理者は Web サーバーとログサーバーの管理者であり，ユーザーは Web サーバーの利用者である．まず，ユーザーから見ていく．ユーザーは Web サーバーへアクセスする．Web サーバーはアクセスされると，ログサーバーへアクセスログを送信する．ログサーバーのアクセスログの受け取

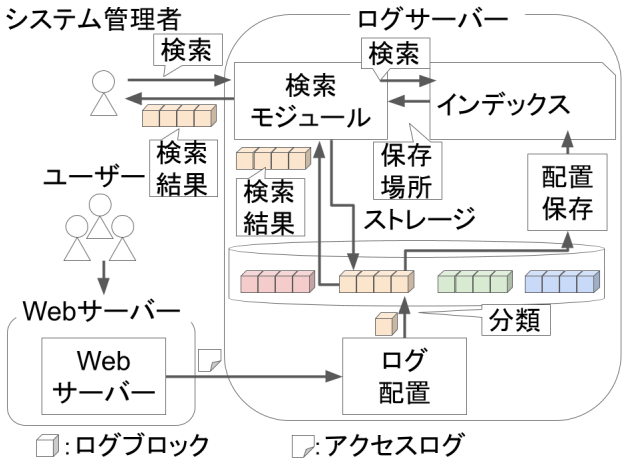

図 2 提案の全体図

りをしているログ配置が 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．配置保存は保存したアクセスログをインデックスに配置場所を記録する．

次に，システム管理者に着目して説明する．システム管理者はステータスコードと URL に条件をつけて検索をする．システム管理者の検索の要求はログサーバーの検索が受け取る．システム管理者の検索の要求から，ログの保存場所を記録しているインデックスへ検索要求をする．インデックスから帰ってきた保存場所からログサーバーの検索はストレージへアクセスして保存場所からログを取得する．取得したログはログサーバーの検索を通してシステム管理者へ検索結果として渡される．

提案の説明を以下の 3 つに分けて説明する．

- ログブロックの作成方法
- インデックスの作成方法
- 検索方法

まずはログブロックの作成方法について説明する．

ログブロックの作成方法

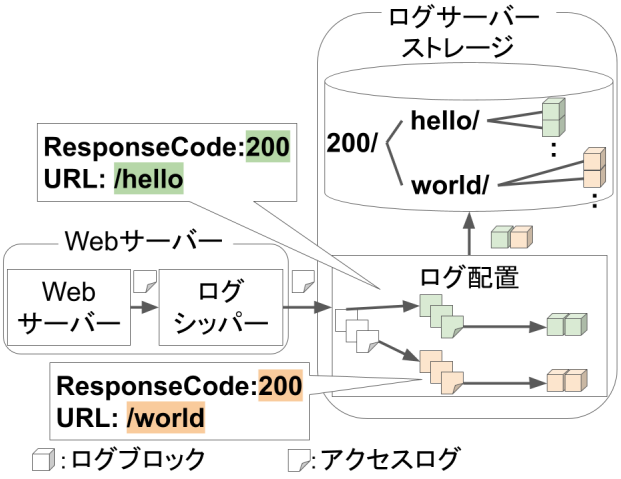

図 3 ログの分類

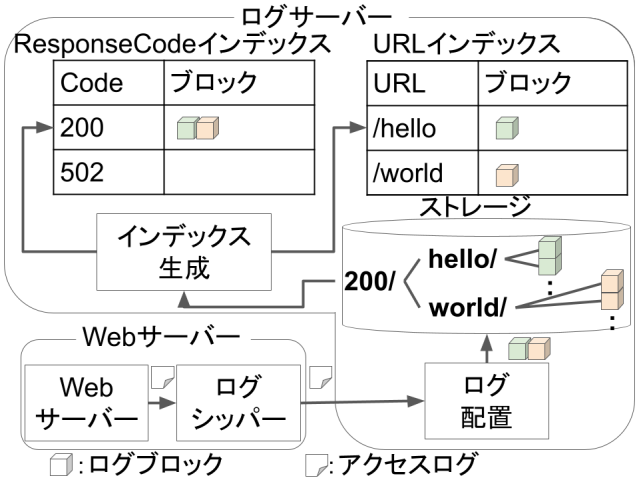

図 4 インデックスの作成

図 3 はログの分類を示している．ログの分類は Web サーバーでアクセスログが生成するたびに行われ，ログサーバーのストレージにログブロックとしてアクセスログが保存される．Web サーバーには，Web サーバーを起動しているソフトウェアとアクセスログを送信するログシッパーがある．ログサーバーにはログ配置があり，アクセスログを受け取り分類してからストレージに保存する．まず，Web サーバーがアクセスログを生成する．生成したアクセスログはログシッパーが読み取り，ログサーバーのログ配置へ向けて送信する．ログサーバーのログ配置は Web サーバーからのアクセスログを受け取る．ログ配置はログサーバーにアクセスログをストレージに保存する前にログブロックにしている．ログブロックはステータスコードとアクセス先の URL が一致しているアクセスログごとにまとめて作成される．図 3 には例として緑と橙の 2 つに分けている．緑は ResponseCode が 200 で URL が/hello のアクセスログを示している．橙は ResponseCode が 200 で URL が/world のアクセスログを示している．緑と橙は ResponseCode が 200 で同じだが，URL が/hello と/world で異なっているため異なるログブロックとして作成する．ログ配置は作成したログブロックをストレージに保存する．ストレージは 2 つ分のディレクトリの深さを持っている．1 つ目はステータスコードごとに分かれている．2 つ目はアクセス先の URL ごとに分かれている．すなわち，緑と橙は 1 つ目で同じ 200 のディレクトリへ保存されるが，2 つ目の URL で異なるディレクトへ保存される．ログ配置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．URL は完全一致ではなく，URL のスラッシュで区切られた 1 番最初の階層のアドレスで判定する．例えば，URL が/hello/ono と/hello/coyama の場合，同じブロックとして保存される．

インデックスの作成方法

図 4 はインデックスの作成方法を示している．図 3 と同様に Web サーバーには，Web サーバーを起動しているソフトウェアとアクセスログを送信するログシッパーがある．ログサーバーにはログ配置があり，アクセスログを受け取り分類してからストレージに保存する．インデックスの作成は図 3 のログの分類をするたびに行われ，ログブロックの保存場所の記録先としてインデックスが作成される．インデックスは ResponseCode インデックスと URL インデックスの 2 つをさす．図 4 ではストレージに保存したアクセスログをまとめたログブロックがストレージのどの場所へ保存したかを示すインデックスを作成する．インデックスはログ検索のときに検索の探索範囲を削減するためにアクセスログの配置場所を記録している．インデックス生成はログサーバーのストレージに新しくログブロックが保

存されたことを検知して起動する．インデックス生成はストレージに新しく保存したログブロックの保存場所をストレージから読み取る．インデックス生成は読み取った保存場所をインデックスへ保存する．ResponseCode インデックスはブロックの配置場所をステータスコードごとに記録している．URL インデックスも同様にブロックの配置場所をアクセス先の URL ごとに記録している．すなわち，ResponseCode インデックスでは緑と橙は ResponseCode が 200 のアクセスログであるため，どちらも 200 のインデックスへ保存場所を記録する．同様に URL インデックスでは緑は URL が/hello のアクセスログであるため，URL インデックスの/hello のインデックスへ保存場所を記録する．対して橙は URL が/world のアクセスログであるため，URL インデックスの/world のインデックスへ保存場所を記録する．

検索方法

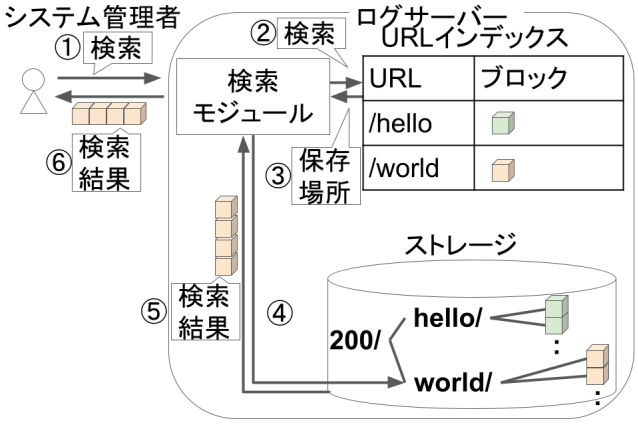

図 5 検索の流れ

図 5 は検索の流れを示している．まず，①でシステム管理者はログサーバーの検索モジュールに検索クエリを発行する．次に②で検索モジュール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．例として URL が/world のアクセスログを検索したいとき，URL インデックスを使用し URL に/world があるアクセスログの保存場所を明らかにする．図 5 では橙のログブロックが/world の保存場所であるため，橙のログブロックを検索結果として取得すればよい．③では検索モジュールがインデックスから保存場所を取得する．④では検索モジュールが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．⑤で検索ソフトウェアは検索結果となるブロックを取得し検索結果としてまとめる．⑥で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

生活雑貨を販売している EC サイトである生活雑貨サイトには，例えば無印良品がある．図 6 はユースケースシナリオを示している．ユーザーは Web サーバーに生活雑貨サイトから商品を購入するためにアクセスする．アクセスされた際に Web サーバーはコンテンツをユーザーに送り返す．このとき，Web サーバーにアクセスしたことを示すログが生成される．生成されたログはログサーバーに送られて保存される．アクセス数は商品を販売している EC サイトのアクセスログを記録している EClog を参考に 8,000 [件/時] とする [5]．

このとき，外部の API の決済サービスが原因で決済できなかったときを想定する [4]．外部 API の決済サービスのエラーはレスポンスが返って来ないため 500 番台のステータスコードが出力される．ユーザーはカスタマーサポートに商品を購入できないことを問い合わせる．カスタマーサポートはいつ購入できなかったかを聞く．カスタマーサポートはシステム管理者へ商品が購入できないシステム障害が発生したと報告をする．このときシステム管理者は障害対応の足がかりを掴むためにログサーバーへログを検索する．

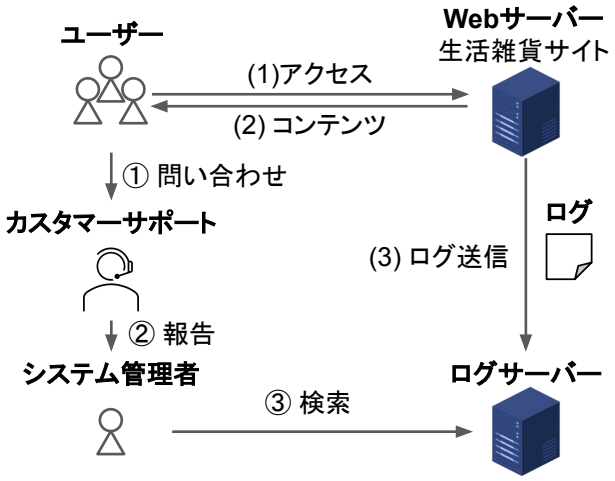

図 6 ユースケースシナリオ

図 7 はユースケースシナリオの構成を示している．Web サーバーは Kubernetes で作成している．Web サーバーは 3 台構成で，Kubernetes を制御する Master を 1 台と，アプリケーションを起動する Worker を 2 台で構成する．Web サーバーのアプリケーションは WordPress と MySQL を使用する．WordPress と MySQL はコンテナとして起動し，Worker1 台あたり 1 つずつ起動する．SVC は Service を表し，ユーザーのアクセスを WordPress のコンテナへ伝える．ストレージは Kubernetes 内で PersistenceVolume，PV として NFS サーバーのものを使用する．ログサーバーは WordPress から出力されたログを取得するサーバーである．

システム管理者は何人のお客様がアクセスできなかったか，どのページにアクセスできないのかをログ検索で調べ

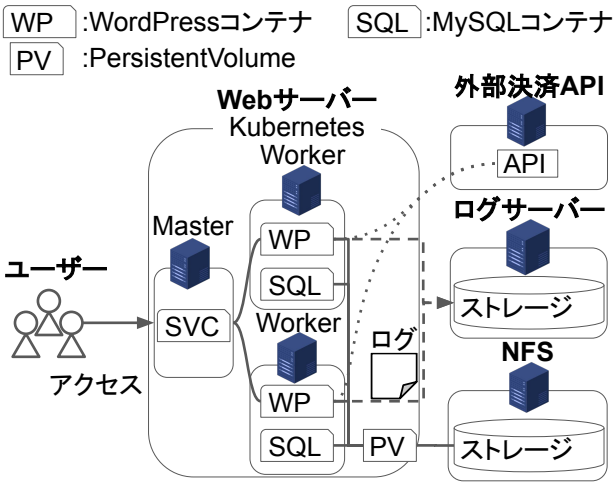

図 7 ユースケース

る．商品を購入できなくなる原因は複数あるため，システム管理者はまずページを閲覧できない原因を絞り込む．アクセスログに HTTP ステータスコードが含まれる．HTTP ステータスコードは 400 以上 600 未満がエラーを示す．生成されたアクセスログの内，異常を示すステータスコードが何件あるのか，ステータスコードをカウントすることで調べることができる．システム管理者はステータスコードをカウントをすることで，一番出力されている異常を示すステータスコードを知ることができる[*2]．ここでは 503 が一番多い異常を示すステータスコードとする．次にステータスコード 503 が発生している URL を調べることでどこが閲覧できなくなっているかを調べることができる．図 8 はステータスコード 503 に対応するときにどの URL から対処するかの方法を示している．図 8 にはシステム管理者とログが保存されているログサーバーがある．システム管理者はステータスコード 503 が発生している URL をログ検索で明らかにする．このときシステム管理者は原因調査のためステータスコード 503 が出力されたアクセスログから URL をカウントする．カウントすることでどの URL が閲覧できなくなっているか，ユーザーのアクセス数順にすることができる．ステータスコード 503 の URL をカウントした検索結果はシステム管理者がどの URL から障害対応すればよいかの優先順位付けに用いることができる．

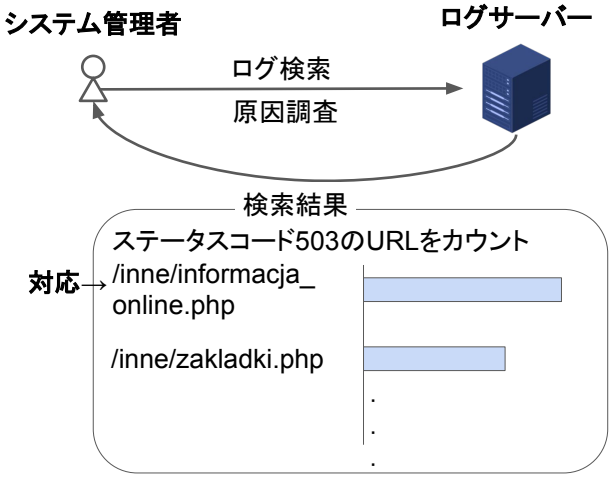

図 8 ユースケースシナリオの流れ

[*2] https://www.loggly.com/ultimate-guide/troubleshoot-with-apache-logs/

4. 実装

実装するソフトウェアはログ配置とインデックス生成，ログ配置，検索ソフトウェアである．このうちログ配置は既存のソフトを使用する．

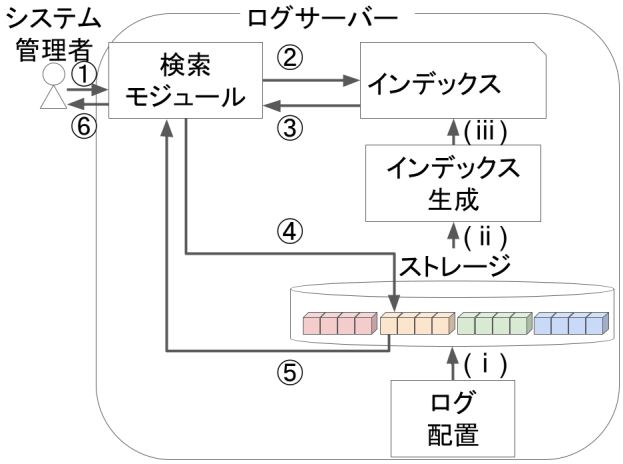

図 9 実装の全体図

図 9 は実装の全体図を示している．実装するソフトウェアは 3 つあり，ログ配置とインデックス生成と検索モジュールである．実装は Python3 で行う．以下に実装するソフトウェアの流れを説明する．ログファイルの保存を (i) から (iii) に示す．(i) はログ配置がストレージに保存している．ログ配置は Web サーバーから受け取ったログファイルをステータスコードと URL ごとに分類する．分類したログファイルはステータスコードと URL が一致するアクセスログがあるディレクトリに保存される．ログブロックはステータスコードと URL は完全に一致しているものをまとめる．(ii) はインデックス生成が (i) でストレージに書き出されたログブロックを読み出し，保存場所を確認する．(iii) はインデックス生成がログブロックの保存場所をインデックスとして保存する．保存したインデックスは JSON 形式で保存し，検索のときに呼び出すことでどの場所へログブロックを保存したかを明らかにする．

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モジュールへ検索クエリを発行している．②は検索モジュールがインデックスへ検索クエリの条件に一致するブロックがどこに保存しているかを検索している．ここでは，URL に関する検索条件を/world を含むものと指定

して URL インデックスに検索をしている．③はブロックの保存場所を URL インデックスから読み取っている．④は検索モジュールが③で読み取ったログブック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で検索モジュールは検索結果をブロックで集め，結合してシステム管理者へ返している．

5. 実験と分析

実験環境

ESXi を使って VM を作成し，実験する．OS に Ubuntu 22.04 を使用する．VM に Elasticsearch をインストールする．実験には EC サイトのアクセスログを使用する．EC サイトのアクセスログは HARVARD Dataverse より EClog を使用する [5].

検索クエリはステータスコードごとの URL をカウントして集計をする．ログサーバー内のみの検索の応答時間を計測する．検索の流れは，システム管理者が検索クエリを発行して，ログの配置場所からログを取得する．取得したログは検索実行で検索結果としてまとめてシステム管理者へ検索の応答を渡す．実験で計測する検索の応答時間は，検索の命令をしてからログ取得が完了するするまでの時間とする．時間の計測方法は Linux の time コマンドを使用する．

実験結果と分析

図 10 と図 11 は検索の応答時間の比較を示している．横軸はログ検索の件数を示していて，単位は [10^6 件] である．縦軸は検索の応答時間を示していて，[秒] である．凡例として，co_min，co_max，wc_min，wc_max，elas_min，elas_max の 6 つある．co_min と co_max は検索結果を 1 行ずつカウントした実装の検索の応答時間を示している．wc_min と wc_max は検索結果を wc コマンドで行数をカウントした実装の検索の応答時間を示している．elas_min と elas_max は Elasticsearch で検索した際の検索の応答時間を示している．co_min と wc_min，elas_min は検索の応答時間の最小値を，co_max と wc_max，elas_max は検索の応答時間の最大値を示している．

図 10 は Python3 で 1 行ずつカウントした提案手法と wc コマンドで実装した提案手法，Elasticsearch の検索の応答時間を比較したものである．co_min と co_max は Pyhon3 で該当するログを 1 行ずつカウントして検索結果を出力している．7,175,241[件] である 1 ヶ月分が検索対象のとき co_min は 49.5[秒] であった．35,157,691[件] である 6 ヶ月分が検索対象のとき co_min は 245.8[秒] であった．検索対象が多くなるほど検索の応答時間が増加していることが分かる．カウントの処理に時間がかかっていると予測し，行数をカウントすることの出来るコマンド wc -l に置き換えることで検索の応答時間を削減した．wc_min と wc_max は上記の co_min と co_max の行数カウントの実装を wc -l コマンドに置き換えたものである．

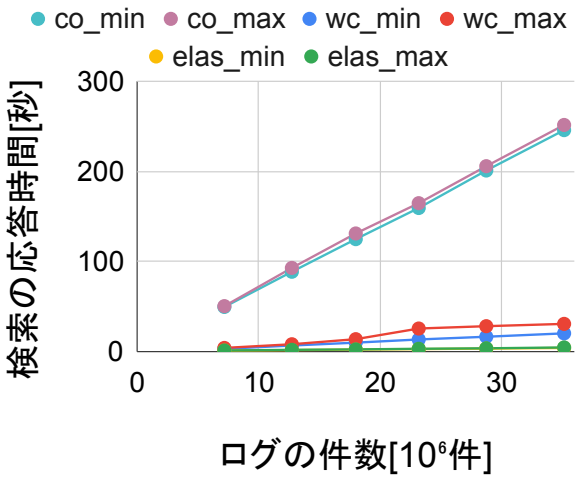

図 10 提案と既存の検索の応答時間の比較

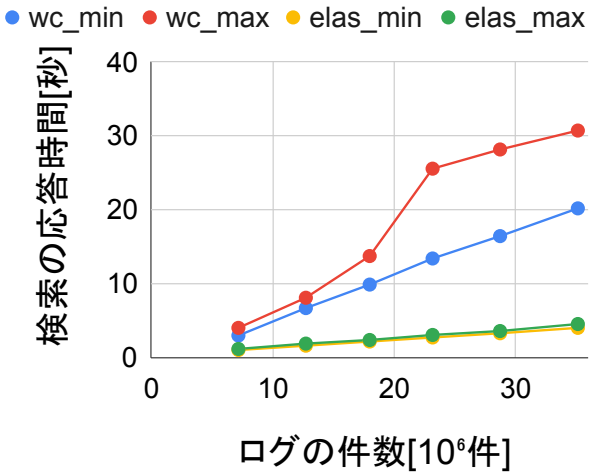

図 11 実装を改善した提案と既存の検索の応答時間の比較

図 11 は wc コマンドで実装した提案手法と Elasticsearch の検索の応答時間を比較したものである．7,175,241[件] である 1 ヶ月分が検索対象のとき wc_min は 3.0[秒] であった．35,157,691[件] である 6 ヶ月分が検索対象のとき wc_min は 20.2[秒] であった．co_min より検索の応答時間が短いことが分かる．elas_min の結果を以下に載せる．7,175,241[件] である 1 ヶ月分が検索対象のとき elas_min は 1.0[秒] であった．35,157,691[件] である 6 ヶ月分が検索対象のとき elas_min は 4.0[秒] であった．wc コマンドで実装した提案手法と Elasticsearch は同様に検索の対象の件数が増えるほど検索の応答時間は増加していた．実験結果から検索の対象が増えるほど wc コマンドで実装した提案手法と Elasticsearch の検索の応答時間の差は広がることが分かった．

また，wc_max に着目すると 20 [10^6 件] を超えた辺りから検索の応答時間が上がっている．原因として VM で実験していたため，他の使用者が VM を起動していたことによる影響を受けたことがあげられる．再度実験をすることで他の使用者による影響か検索の処理の仕方による影響かが分かる．

6. 議論

検索にかかる時間が長いクエリは複数ノードを用いることで検索の応答時間を短くすることができる．複数ノードに処理を分散する分散処理はノード数を増やすほど処理時間は早くなる．しかし，複数ノードに分けた処理を通信で受け渡すために時間がかかる．そのため，分散する仕事に対して通信がどれだけかかるかを見積もる必要がある．通信の時間を l と置き，処理全体にかかる時間を t と置き，分散するノード数を n と置く．このとき，条件式 $t/n > l$ を満たすのであれば分散処理をしても処理全体にかかる時間は早くなるといえる．実験で l と t を求めて n を算出することで複数ノードを用いた検索の応答時間全体を削減することができる．

アクセスログに載っている URL は同じページにアクセスしていたとしても HTTP の GET のクエリによって文字列として前方のみが一致している URL が記録されているときがある．どこからアクセスしてきたかをアクセスログに保存し，マーケティングの目的で使用される．エラーが出力された URL の数をカウントする際に異なる場所からアクセスしたリクエストを異なるサイトと認識することになる．GET リクエストの始まりの記号である，?より後ろの URL を無視してカウントすることで，どこからアクセスしたかに左右されずに一番エラーが発生している URL を特定することが出来る．

7. おわりに

課題はシステムの原因調査のために月単位の範囲で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．提案として ResponceCode インデックスと URL のインデックスを作成し保存場所を分けることで，検索の応答時間を削減する．評価方法は提案手法でログを分類して保存場所分けて検索した場合の検索の応答時間と既存のソフトウェアである Elasticsearch の検索の応答時間を比較した．結果として，検索の応答時間は Elasticsearch と比較して 438[%] 上昇した．

謝辞 ご指導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木 優希さん，伊藤 佳城さん，杉本 一彦さん，中川 翔太さん，飯島 貴政さん，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 真斗さんに御礼申し上げます．

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 (online), DOI: 10.1109/TDSC.2017.2762673 (2018).

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

[3] 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).

[4] 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).

[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] Arge, L., Danner, A. and Teh, S.-M.: I/O-Efficient Point Location Using Persistent B-Trees, *ACM J. Exp. Algorithmics*, Vol. 8, p. 1.2–es (online), DOI: 10.1145/996546.996549 (2004).

[7] 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).

[8] Chaudhuri, S. and Narasayya, V. R.: An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server, *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., p. 146–155 (1997).