問い合わせメールとテンプレートを適用した検索クエリによるキャッシュヒット率の向上

川端 ももの[1]　小山 智之[2]　串田 高幸[1]

概要：ログは，システムの障害発生時に原因を特定するために検索される．その際に，プリフェッチを行うと検索結果の表示が早くなる．しかし，全てのキャッシュを残しておくことはメモリの容量に限界がある．そのため，プリフェッチするログを選別する必要がある．課題は，既存の手法を適用できるが，突発的に発生するに使われるクエリのプリフェッチには対応しきれないことである．提案は，Web サイト利用者の問い合わせメールの内容からクエリを発行しプリフェッチを行う手法である．問い合わせ内容を構造化テキストに変換し，事前に用意したクエリテンプレートに適用させることでクエリを発行する．この提案を使用することで問い合わせメールが届き，原因を特定する際の検索クエリに合わせたプリフェッチを行うことができる．そのため，キャッシュヒット率を向上させることができる．基礎実験は，キャッシュがある場合とない場合のログの検索時間を測定した．EClog のログを使用して Elasticsearch で 50 回ずつステータスコードが 400 番台のログの検索を行った．結果は，キャッシュがある場合の最頻値は，0 秒以上 0.1 秒以下であった．キャッシュがない場合の最頻値は，0.7 秒より大きく 0.8 秒以下であった．したがって，キャッシュがある時の方がキャッシュがない時より検索時間が短い．評価方法は，東京工科大学のプロジェクト実習に関しての問い合わせメールの内容から検索クエリを発行し，提案ソフトウェアが発行した検索クエリと比較する．

1. はじめに

背景

ログとは，システムの実行状態やアクセス履歴，エラーを記録するメッセージでありログステートメントによって生成される [1]．ログは開発者やエンジニアがシステムを監視や解析を行うために利用している．ログを使用することでパフォーマンスの障害の特定，エラーとクラッシュの診断が可能である [2]．

ログはシステムの貴重な情報源となるため，法的記録として管理，保存する必要がある [3]．また，現代ではソフトウェアが複雑化，大規模になったことで出力されるログの量が増加している [2, 4]．そのためログサーバーでログを 1 か所に集めて管理，保存を行っている．

ログを 1 か所に収集するソフトウェア群として，Elastic 社が提供している Elastic Stack がある [5]．Elastic Stack は Elasticsearch，Kibana，Beats，Logstash で構成されている [6]．Elastic Stack を使用してログを送信する流れを図 1 に示す [7]．WordPress は Web サイトを作ることができるオープンソースのコンテンツマネジメントシステムである [8]．この WordPress を使用した Web サイトからログを収集する．Beats はデータの収集，変換，および転送を行うソフトウェアである [9]．Beats の 1 つに Filebeat

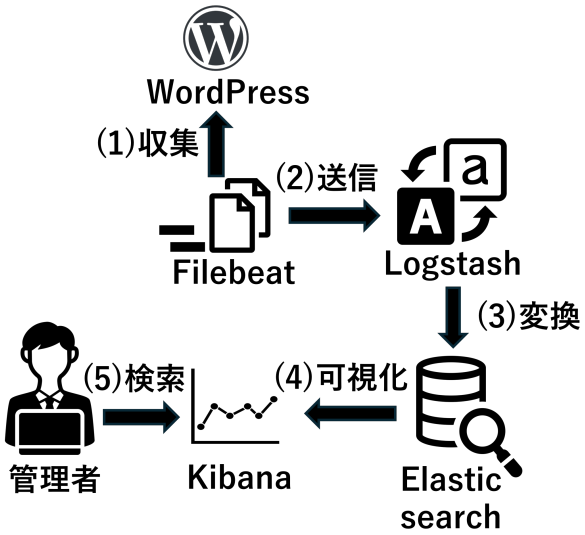

図 1　Elastic Stack を使用してログを送信する流れ

[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

がある [10]．WordPress に存在するログは Filebeat でログサーバーにある Elasticsearch に送信する．ログは非構造化されておりシステムやソフトウェアによって形式が異なる [11]．そのため，Logstash を使用して収集したログを Elasticsearch が処理できるように変換する [12]. 収集されたログは Elasticsearch に保存される．Kibana は Elasticsearch に収集されたログを可視化するグラフィカルユーザーインターフェイスである．管理者はクエリを用いてログの検索やインデックスパターンからダッシュボードを作成することができる．そのためデータの解析や，どのようなイベントが行われているか調査する際に使用される [13].

キャッシュとは同じものを検索, 閲覧した際の応答時間を短縮するためにデータを保存しておくことである．キャッシュしたデータはキャッシュメモリに一時的に保存する．低速メモリよりキャッシュメモリの方が処理が高速である．キャッシュメモリを使用することで，全てのデータが保存されている低速メモリの使用を回避できるため，アクセス時間の短縮ができる [14].

プリフェッチとは，アクセスされるデータを予測し，予測されたデータを事前にキャッシュメモリに保存することである [15]．キャッシュは同じものを検索するときにアクセス時間が短くなるのに対して，プリフェッチは一度も検索や閲覧をしたことがないデータのアクセス時間を短縮することができる．

検索時にキャッシュがある時とない時の検索結果表示までの流れを図 2 に示す．(1) はキャッシュありの時，(2) はキャッシュなしの時の流れである．(1) と (2) は最初に管理者が検索クエリを用いて Kibana でログを検索する．次に検索クエリにヒットするログを，メモリから探す．キャッシュがある場合は (1) のキャッシュメモリで検索を行う．キャッシュメモリは保存されているログの量が少ない．そのため，キャッシュなしの時より検索の時間が短い．キャッシュがない場合は，(2) のアクセスメモリから検索を行う．アクセスメモリには全てのログが保存されている．そのため，キャッシュありの時より検索の時間が長い．したがって，キャッシュありの時の方が検索結果表示が速くなる．

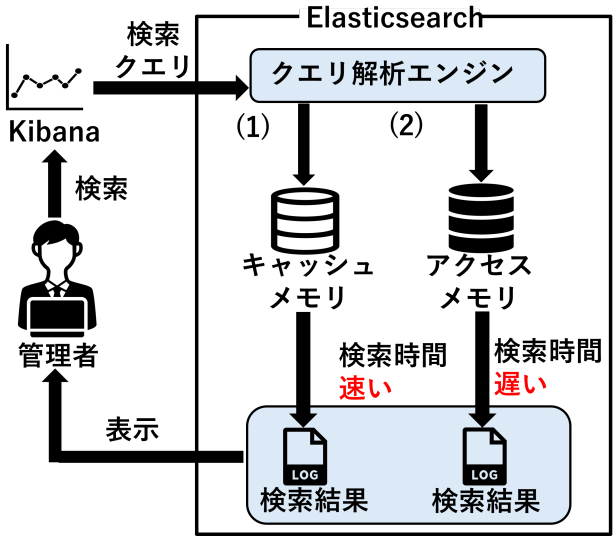

図 2　キャッシュありとなしの検索結果表示までの流れ

Elasticsearch には過去に発行されたクエリをキャッシュする機能はあるが，将来的に発行される検索クエリに対して事前にキャッシュをしたり，プリフェッチしたりする機能がない．そのため，初めて検索するクエリに対しては検索に時間がかかる．したがって，全てのログをプリフェッチすることで検索の時間を速くすることができる．しかし，メモリの容量は上限があるためログを全てプリフェッチすることができない．よってプリフェッチするログを選ぶ必要がある．また，検索でアクセスされるログは検索クエリによって違うためクエリに合わせてプリフェッチするログを選ぶ必要がある．

既存のプリフェッチ方法として，コンテンツベースと履歴ベースが挙げられる [16]．コンテンツベースのプリフェッチはコンテンツの内容を分析して，ユーザーが次に見るコンテンツを予測してプリフェッチする方法である．履歴ベースのプリフェッチは，過去のアクセスパターンにもといて，次に必要となるログデータを予測し，事前に読み込む方法である．

課題

課題は，既存のコンテンツベースと履歴ベースのプリフェッチでは，突発的な障害発生時の原因調査に使用する検索クエリのプリフェッチには対応しきれないことである．そのため，キャッシュヒット率が低下する．Web サイトの障害はサイト閲覧者が見ていたコンテンツの内容に関わらず発生するためコンテンツベースでは Web サイトの障害が発生した場合のプリフェッチには対応できない．Web サイトの障害は突発的に起こるため，Web サイトの障害が発生した場合は閲覧者が過去に見た履歴からは予測が不可能である．既存の方法ではどちらもユーザーが見たコンテンツをもとにプリフェッチを行っている．Web サイトの障害が発生し，検索クエリを発行する際のコンテンツベースと履歴ベースのプリフェッチの流れを図 3 に示す．コンテンツベースはコンテンツ閲覧者が見たコンテンツ内容，履歴ベースは閲覧者がみたコンテンツから次に閲覧者が見るコンテンツを予測する．予測したコンテンツをキャッシュメモリに保存する．障害が起こった場合，障害に気づいた管理者が原因を調べるため，障害が起こったコンテンツに検索をする．そのため，管理者が検索する障害が起こったコンテンツがキャッシュメモリに入っていることで検索をアクセスメモリから検索する時より速くすることができる．管理者はしかし，閲覧者が見たコンテンツはキャッシュメモリに存在するが，キャッシュメモリに障害が起こったコ

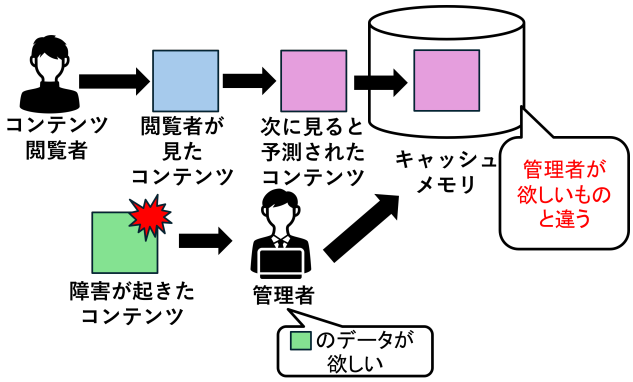

図 3 既存方法でのプリフェッチの流れ

ンテンツが存在しないためプリフェッチのヒットが出来ずキャッシュメモリを検索するときより遅くなる．

各章の概要

2 章は関連研究について述べる．3 章は提案について述べる．4 章は提案ソフトウェアの実装について述べる．5 章は評価実験と基礎実験について述べる．6 章は議論について述べる．7 章はまとめについて述べる．

2. 関連研究

グローバル履歴バッファーを使用してプリフェッチを行う論文がある [17]．この論文はテーブルキーのマッチングとプリフェッチの履歴を代替プリフェッチ構造を提案している．従来のプリフェッチと同様のキーを使用してアクセスするインデックステーブルとグローバル履歴バッファを使用した FIFO テーブルで構成されている．この提案では，履歴ベースのプリフェッチを行っており，使用者が見たコンテンツがキャッシュメモリに保存される．また，キャッシュミスが発生すると，インデックステーブルにインデックスが付けられプリフェッチ候補になる．そのため，従来のプリフェッチ方法より効果的なプリフェッチ方法である．しかし，本論文では障害が起こった際を想定しており，障害は利用者が見たコンテンツに関係なく発生するためこの提案は使用できない．新しいデータプリフェッチ方法であるアクセスマップパターンマッチング (以下，AMPM) プリフェッチを提案している論文がある [18]．AMPM は, メモリアドレス空間を固定サイズの領域に分割し，これを管理単位としてゾーンとしている．最近アクセスされたゾーンはホットゾーンとして検出される．AMPM プリフェッチャー，メモリアクセスマップの並列パターンマッチングによってプリフェッチ候補を検出し，プリフェッチ要求を生成する手法である．評価では，PC/DC，C/DC と AMPM プリフェッチャーの性能を比較している．AMPM プリフェッチャーを使用することでパフォーマンスが 42.0%向上した．この提案は最近アクセスされたゾーンをプリフェッチ候補としているが，Web サイトの障害はアクセスしたゾーンに関係なく発生するため，本稿では使用できない．

ベイジアンネットワーク理論にもとくキャッシュプリフェッチ戦略を提案している論文がある [19]．ベイジアンネットワーク理論とキャッシュのコストと利益に応じてプリフェッチするファイルを選択する．応答時間は利益の尺度として，キャッシュスペースの占有はコストの尺度として用いられている．評価ではベンチマークと比較して，プリフェッチヒット率と要求応答時間を改善し，メモリ負荷を大幅に削減した．この提案では，過去のユーザーによるファイルアクセス操作を用いてベイジアンネットワーク理論でプリフェッチするファイルを決定しているので，Web サイトに問題が発生した際にどのログを見るのか判断できない．そのため，本稿では使用できない．

データアクセス履歴キャッシュ (以下，DAHC) という新しいキャッシュ構造を提案している論文がある [20]．DAHC は，命令やデータ用の従来のキャッシュではなく，最近の参照情報用のキャッシュとして動作する．データアクセス履歴テーブル (以下，DAH テーブル) と 2 つのインデックステーブルで構成されており，DAH デーブルは履歴の詳細を格納している．SimpleScalar ツールセットを使用してシミュレーションを行いストライドプリフェッチ，マルコフプリフェッチ，MLDT アグレッシブプリフェッチとのキャッシュミス率の比較をした．その結果，キャッシュミス率が大幅に削減した．この提案では，データのアクセス履歴を使用しているが，本稿では障害が起こった際を想定しており，障害は利用者が見たコンテンツに関係なく発生するためこの提案は使用できない．

3. 提案

提案方式

Web サイト利用者の問い合わせのメールからプリフェッチするログを判断する方法を提案する．提案ソフトウェアへの入力はテキストパーサーの出力結果とクエリテンプレートである．出力はクエリテンプレートを適用させたクエリである．使用する問い合わせメールは，東京工科大学のプロジェクト実習を履修している学生が Teaching Assistant に送信した問い合わせメールである．クエリテンプレートを適用させたクエリは Elasticsearch で検索しプリフェッチをする際に使用される．提案ソフトウェアを使用した際の流れを図 4 に示す．(1) から (4) は以下のサブセクションで述べる．

原因調査が必要な問い合わせ内容かの判断

問い合わせメールの内容からサーバーの原因調査が必要な場合と必要でない場合を判断する．問い合わせメールは保存した txt 形式の内容をテキストパーサーに API に渡して，問い合わせ内容を判断する．提案ソフトウェアは原

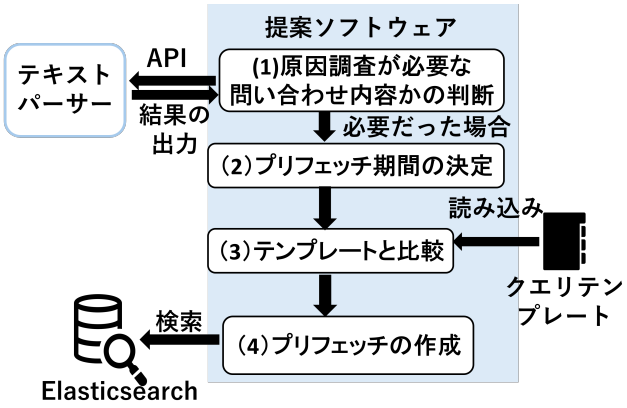

図 4　提案ソフトウェアの流れ

コード 1　原因調査が必要な問い合わせメール例

```
名前：工科 太郎
メールアドレス：tarouk@edu.teu.ac.jp
本文：学籍番号 123456の工科太郎です。
先日いただいた宿題未提出者向けのメッセージについてお伺いしたいことがあります。

停電の日からリバースプロキシ表からサイトにアクセスできず、課題に取り組むことができていません。この件について何度もメールで問い合わせたのですが、メールは届いてませんでしょうか？また、このことを踏まえて、成績、単位への配慮をお願いできませんでしょうか。
```

因調査が必要とテキストパーサーが判断した場合のみ実行する．原因調査が必要な問い合わせメールの内容をコード 1 に示す．問い合わせメールにはプロジェクト実習で使用しているリバースプロキシにアクセスできないことが書かれている．リバースプロキシにアクセスできないという内容はプロジェクト実習で使用するサーバーに障害が発生している場合があり，原因調査が必要である，したがって，提案ソフトウェアを実行する．

プリフェッチ期間の決定

プリフェッチの際の timestamp フォーマットの gte と lte の指定範囲の決め方を図 5 に示す．timestamp は時間を表すフォーマットである．gte は以上を表し，lte は以下を表す．ログ 1 行 1 行にはステータスコードが記載されている．ステータスコードの 200 番台はリクエストの処理が成功したことを表し，ステータスコード 400 番台はクライアントエラーを表す [21]．ソフトウェア 1 は 30 分前に障害が発生し，30 分前から現在までステータスコードが 400 のログが発生している．ソフトウェア 2 は障害が発生せずステータスコードは 200 である．利用者がソフトウェア 1 にアクセスした際，ソフトウェア 1 が使用できず，ステータスコードが 400 のログが発生していた．そのため，利用者は問い合わせメールを作成し送信する．lte は問い合わせメールが到着した時間に一番近いソフトウェア 1 のステータスコードが 400 のログの発生日時を入れる．gte は

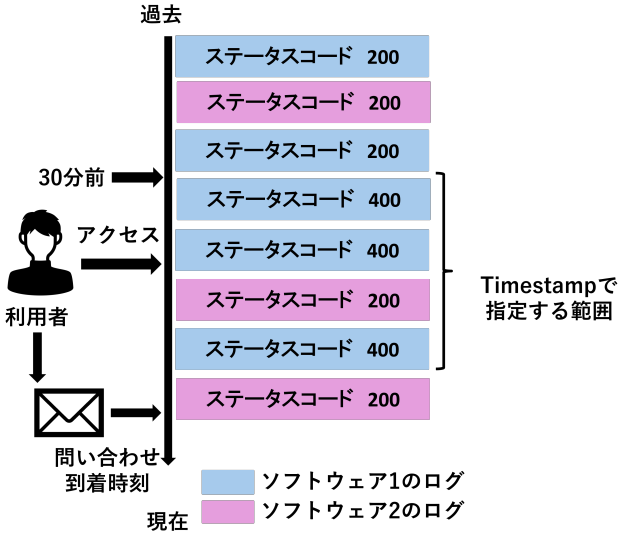

図 5　gte と lte の指定範囲の決め方

コード 2　クエリテンプレート

```
{
  "query": {
    "range": {
      "timestamp": {
        "gte": "",
        "lte": ""
      }
```

ソフトウェア 1 のステータスコードが 200 から 400 に変わった際のログの発生日時を入れる．図 5 では 30 分前にソフトウェア 1 のステータスコードが 200 から 400 に変わっているため，gte には 30 分前に発生したログの発生日時が入る．これは，ソフトウェアが障害が起きた時間を想定している．

テンプレートと比較

次に提案ソフトウェアに保存していたクエリテンプレートに問い合わせメールから求めた gte と lte の値を適用する．事前に管理者はクエリテンプレートを作成する．クエリのテンプレートをコード 2 に示す．このクエリは JSON オブジェクトであり，query はこのコードブロックがクエリであることを示している．range は範囲検索をするためのオブジェクトである．timestamp は時間を表すフォーマットである．gte は以上を表し，lte は以下を表す．ステータスコードが 200 から 400 に変わったログの発生日時が 2024 年 07 月 09 日 14 時，問い合わせメールが到着した日時に一番近いステータスコード 400 のログの発生日時が 2024 年 07 月 09 日 14 時 30 分の場合のクエリテンプレートから問い合わせ内容を適用させたクエリをコード 3 に示す．コード 3 は 14 時 00 分から 14 時 30 分の間のログを検索するクエリである．

コード 3　テンプレートクエリ適用後

```
1  "query": {
2    "range": {
3      "timestamp":
4        "gte": "2024-07-09T14:00:00",
5        "lte": "2024-07-09T14:30:00"
6        }
7    }
```

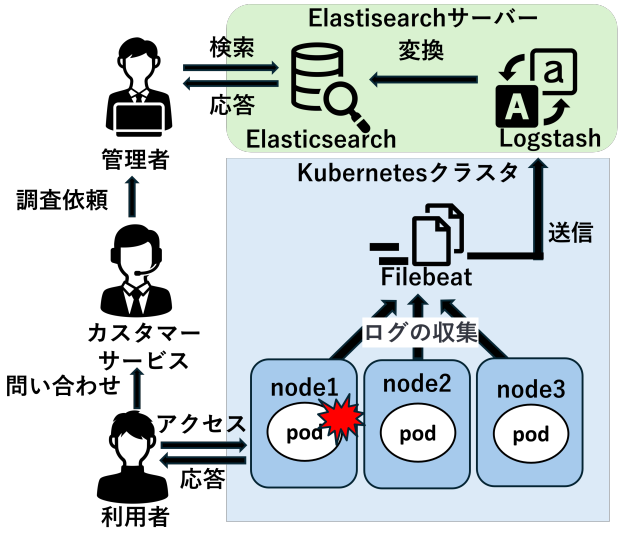

図 6　企業の WordPress の Pod から返答がない状況

プリフェッチの作成

クエリテンプレートから問い合わせ内容を適用させたコード 3 のクエリを Elasticsearch で検索し，プリフェッチを作成する．プリフェッチを作成することで障害が発生し，原因調査でログを検索する際の検索時間を短くすることができる．

ユースケース・シナリオ

株式会社アイキューブドシステムズは Kubernetes 環境のログを Elastic Stack で収集している[*1]．保存したログはユーザーからの問い合わせがあった事象の調査や Kibana でシステムの稼働状況をダッシュボードに可視化する際に使用されている．

本稿は，企業の WordPress の Pod から返答がない場合をユースケースとして挙げる．サーバーに Kubernetes を用いた WordPres が立っている．企業の WordPress の Pod から返答がない状況を図 6 に示す．node は 3 台あり，WordPress の Replicas は 3 つである WordPress のログは Pod に書き込まれてる．Pod のログを Elasticsearch で集めている．管理者は Pod が落ちた際に Elasticsearch でログの検索を行い原因調査をする．Web サイト利用者がサイトにアクセスしたところ Pod から返答がなかったためサイトが見れなかった．そのため，カスタマーサービスにメールやチャットで問い合わせを行う．Web サイト利用者からの問い合わせを受けてカスタマーサービスは WordPress の管理者に調査を依頼する．管理者は WordPress の Pod のログを確認し原因を調査する．その際，Elasticsearch はプリフェッチが無いためプリフェッチがある時より検索時間がかかる．そのため，プリフェッチを残すことで検索時間を速くすることができる．しかし，原因調査で検索するクエリは既存のコンテンツベースや履歴ベースではプリフェッチされていない．本提案を用いることで，問い合わせの内容から検索するクエリを判断しプリフェッチを行う．そのため，ログを検索する時間を短くすることができる．

[*1] https://www.elastic.co/jp/pdf/case-study-i3-systems-jp.pdf

4.　実装

提案ソフトウェアは Python で作成する．事前にクエリテンプレートを作成し，CSV 形式で VM 上に保存する．テキストパーサーとして ChatGPT を使用する．使用するモジュール一覧を表 1 に表す．Python ファイルはクエリテンプレートに問い合わせメールから決定した日時を適用させて，検索クエリを作成する query.py と作成した検索クエリを使って Elasticsearch に検索し，プリフェッチを作成する search.py の 2 つのファイルを使用する．提案ソフトウェアの処理の流れを図 7 に示す．(1) では，API を使用して問い合わせメールを ChatGPT で原因調査をする必要があるか判断する．必要であると判断した場合は，問い合わせメールを (2) で query.py に渡し，CSV ファイルで保存する．保存した問い合わせメールから送信日時を取得し，gte と lte の値を求める．(3) では，テンプレートクエリを提案ソフトウェアに読み込み (2) で求めた gte と lte を適用させる．(4) で作成した検索クエリを CSV ファイルに保存する．(5) では検索クエリのファイルを search.py で開く．(6) で検索クエリを使用して Elasticsearch で検索を行う．

表 1　使用するモジュール一覧

モジュール名	用途
csv	CSV ファイルの読み書きを行う
time	時間を扱う
datetime	日時を扱う
elasticsearch	Elasticsearch を扱う
logging	ログの出力を行う
basicConfig	ログの出力の際のフォーマットを変更する

5.　評価実験

評価方法

評価方法は，学生の問い合わせメールを実験者が閲覧

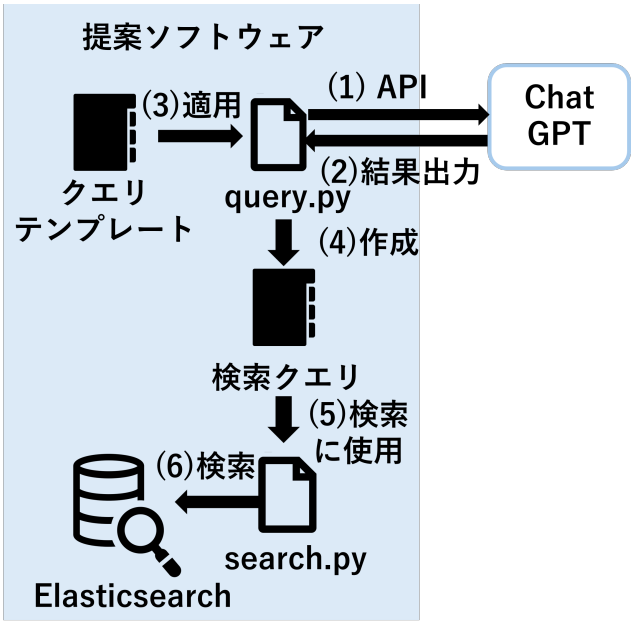

図 7　提案ソフトウェアの処理の流れ

し，発行した検索クエリと提案ソフトウェアが発行したクエリの比較を行いキャッシュヒット率と検索時間を測定する．使用する問い合わせメールは東京工科大学のプロジェクト実習を履修している学生が実際に Teaching Assistant に送信した問い合わせメールである．使用する問い合わせメールは 5 件である．実験者は東京工科大学のクラウド・分散システム研究室の学生（以下，CDSL 生）である．評価方法の流れを図 8 に示す．(1) では問い合わせメールを

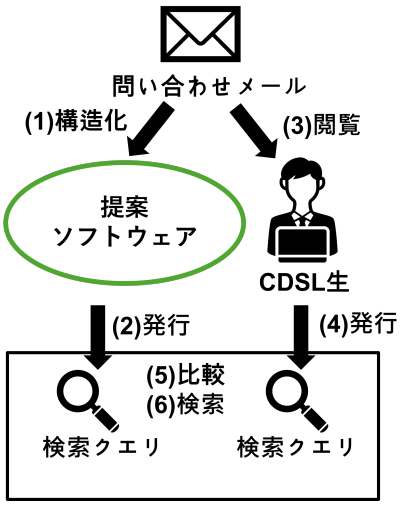

図 8　評価方法の流れ

ChatGPT で構造化する．(2) では提案ソフトを使用し構造化した問い合わせメールの内容から検索クエリを発行する．(3) では CDSL 生が問い合わせメールを閲覧し内容を確認する．(4) では CDSL 生が検索クエリを発行する．(5) では提案ソフトウェアが出した検索クエリと CDSL 生が出した検索クエリが一致しているかの比較を行う．(6) では検索時間を測るため 2 つのクエリを 50 回ずつ検索する．

基礎実験

キャッシュありとキャッシュなしの検索結果表示までの時間を測定した．対象のログは EClog である．EClog を Elasticsearch で検索した際の検索時間を測定した．EClog はハーバード大学のデータベースで公開されているポーランドの Web サイトの Web サーバーアクセスログである [22]．EClog は 2019 年 12 月 1 日から 2020 年 5 月 31 日のログが記録されている．EClog のログをコード 4 に示す．EClog のログには 10 個のフォーマットが含まれている．

コード 4　EClog のログの例

```
11DE,-,637107935860000000,GET,"/p-5586.html",HTTP
    /1.1,200,14566,"-","Mozilla/5.0 (compatible;
    MegaIndex.ru/2.0; +http://megaindex.com/
    crawler)"
```

11DE は IpId フォーマットであり，IP アドレスの識別子である．ISO3166–1Alpha-2 コードを使用しており，11DE はドイツ連邦共和国を表している．UserId フォーマットは Web ユーザーの識別子であり，コード 4 では-になっている．637107935860000000 は TimeStamp フォーマットであり，リクエストの到着時間である．西暦 1 年 1 月 1 日 00:00:00UTC から経過した 100 ナノ秒間隔の数としてコード化されている．HttpMethod フォーマットは，URI に対して行われたアクションを表している HTTP メソッドである．GET はリソースの取得を行っている．/p-5586.html は Uri フォーマットであり，URI の文字列識別子を表している．HTTP/1.1 は HttpVersion フォーマットであり，クライアントが使用する HTTP プロトコルのバージョンが示されている．ResponseCode は HTTP 応答ステータスの数値コードである．200 は，指定されたリクエストがサーバーによって正常に処理されたことを表すコードである．14566 は Bytes フォーマットであり，サーバーが送信した際のデータ量を表している．Referrer フォーマットはクライアントが現在のページにアクセスする前に参照していた Web ページの文字列識別子である．コード 4 では-になっている．UserAgent フォーマットはクライアントのソフトウェアを説明する文字列である．

使用したログの総数は 657,352 件である．検索に使用したクエリをコード 5 に示す．このクエリは，レスポンスコードが 400 番台のログを検索している．これは，管理者がサイトに障害が起こった時間を確認するために検索した状況を想定したものである．ResponseCode は EClog を解析した際のフォーマットである．gte は ResponseCode が 400 以上のものを検索することを示している．lte は ResponseCode が 499 以下のものを検索することを示している．このログをキャッシュありとキャッシュなしで

コード 5　400 番台のステータスコードを検索するクエリ

```
"query": {
    "range": {
        "ResponseCode": {
            "gte": 400,
            "lte": 499
        }
    }
}
```

50 回ずつ測定し，検索時間を度数分布表で表した．検索は Python ファイルで行われている．この基礎実験は全数調査とする．キャッシュを消す関数をコード 6 に示す．request を True にすることで request-cache を消してい

コード 6　キャッシュを消す関数

```
es.indices.clear_cache(
    request=True, # Purge request-cache
    fielddata=True, # Purge field-data-cache
    query=True, # Purge query-cache
)
```

る，fielddata を True にすることで field-data-cache を消している．query を True にすることで query-cache を消している．キャッシュなしの場合は 1 回の検索ごとにこの関数を呼び出している．キャッシュありの場合はこの関数を呼び出さず検索を行う．

実験環境

実験環境を図 9 に示す．EC サーバーの/home/eclog 内

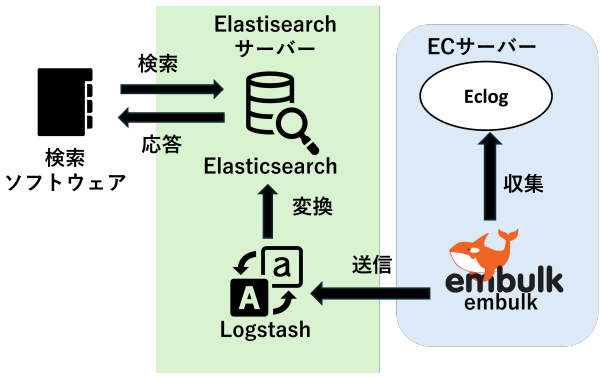

図 9　実験環境

にログが存在しており，拡張子は log である．このログを EC サーバーに構築した embulk を使用して Elasticsearch サーバーの Logstash に送信している．embulk とは大量のデータの取り込み用オープンソースデータコネクタである [23]．Logstash に送信されたログは Elasticsearch が読み込めるように構造化される．管理者は Elasticsearch に Python ファイルを使用して検索をする．Elasticsearch は検索結果と検索時間を返す．

実験結果と分析

657,352 件のログを検索しヒットしたログは 4,416 件であった．測定結果を図 10 に示す．階級は検索時間が何秒

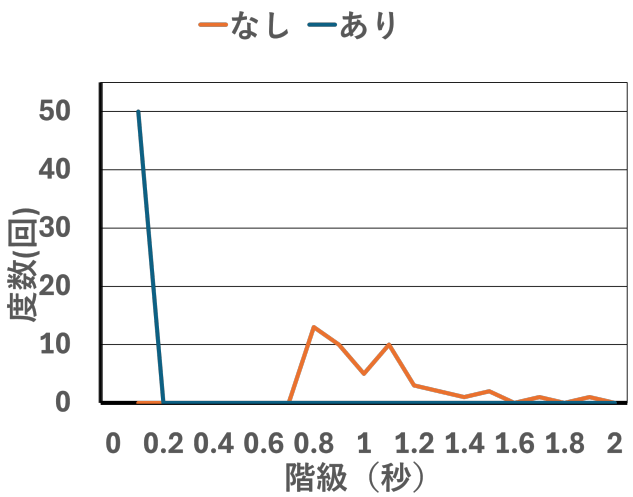

図 10　キャッシュありとキャッシュなしのログ検索の計測結果

の範囲にいるかを表した度数分布表である．測定結果から，キャッシュがありの場合は 50 回すべてが 0 秒以上 0.3 秒以下であることが分かった．キャッシュなしは 0.7 秒より大きく 0.8 秒以下が一番多いことが分かった．また，キャッシュなしは 0.7 秒より速い測定結果はなかった．したがって，キャッシュありの時の方がキャッシュない時より検索時間が短いことが分かった．

6.　議論

本稿では，問い合わせメールに障害発生した日時が記入してある場合を考慮していない．そのため，問い合わせメールに障害発生した日時が記入してある場合も timestamp の lte の値として，問い合わせメールが到着した日時を使用している．しかし，問い合わせメールが到着する日時には利用者がソフトウェアにアクセスを行い，ソフトウェアが利用できないことを確認し，問い合わせメールを作成するまでの時間がかかっている．したがって，問い合わせメールに書かれている日時の方が障害が発生している日時に近い値となる，解決方法として，テキストパーサーで原因調査が必要か判断するだけではなく，問い合わせメールの内容からフォーマットの値として使えるものを抽出する指示を追加する．例として，問い合わせメールに障害が発生した日時やアクセスした日時があった場合は timestamp フォーマットとして抽出する．日時は検索クエリで使用する ISO8601 形式に変換する．

本稿では，手動でクエリテンプレートを作成しており，クエリテンプレート作成は自動化されていない．自動化する際のフォーマットの決め方として，過去に管理者が使用したクエリを使用する方法がある．キャッシュメモ

リにある検索クエリから管理者が頻繁に検索の際に使用しているフォーマットを使用する．フォーマットに入れる値の決め方の例として 400 番台と 500 番台のログにある uri フォーマットの値や Kubernetes 環境の場合 kubernetes.pod.name フォーマットの値をあげる．原因調査が必要な問い合わせメールがきた場合，提案ソフトウェアで 400 番台と 500 番台のログを検索し，uri フォーマットまたは kubernetes.pod.name の値を取得し，検索クエリに反映させる．

7. おわりに

ログは，システムの貴重な情報源であるため Elasticsearch を用いてログを一箇所に集めて管理，検索を行う．プリフェッチを行うと検索が速くなる．しかし，全てのキャッシュを残しておくことはメモリの容量に限界がある．そのため，プリフェッチするログを選別する必要がある．課題は，既存のコンテンツベースと履歴ベースのプリフェッチは，適用できるが，突発的に発生するアラートの対応に使われるクエリのプリフェッチには対応しきれないことである．提案で問い合わせ内容から検索クエリを作成しプリフェッチを行う方法を提案した．基礎実験では，キャッシュがある場合とない場合のログの検索時間を 50 回ずつ測定した．結果は，キャッシュがある場合の最頻値は，0 秒以上 0.1 秒以下であった．キャッシュがない場合の最頻値は，0.7 秒より大きく 0.8 秒以下であった．したがって，キャッシュがある時の方がキャッシュがない時よりログの検索時間が速いことが分かった．評価方法は，東京工科大学のプロジェクト実習に関しての問い合わせメールの内容から検索クエリを発行し，提案ソフトウェアが発行した検索クエリと比較する．

参考文献

[1] Li, H., Shang, W. and Hassan, A. E.: Which log level should developers choose for a new logging statement?, *Empirical Software Engineering*, Vol. 22, pp. 1684–1716 (2017).

[2] Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z. and Lyu, M. R.: Tools and benchmarks for automated log parsing, *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, IEEE, pp. 121–130 (2019).

[3] Anusooya, R., Rajan, J. and SatyaMurty, S.: Importance of centralized log server and log analyzer software for an organization, *International Research Journal of Engineering and Technology (IRJET)*, Vol. 2, No. 3, pp. 2244–2249 (2015).

[4] He, S., He, P., Chen, Z., Yang, T., Su, Y. and Lyu, M. R.: A survey on automated log analysis for reliability engineering, *ACM computing surveys (CSUR)*, Vol. 54, No. 6, pp. 1–37 (2021).

[5] Uday, D. V. and Mamatha, G. S.: An Analysis of Health System Log Files using ELK Stack, *2019 4th International Conference on Recent Trends on Electronics, Information, Communication Technology (RTEICT)*, pp. 891–894 (online), DOI: 10.1109/RTEICT46194.2019.9016706 (2019).

[6] Calderon, G., Del Campo, G., Saavedra, E. and Santamaria, A.: Management and Monitoring IoT Networks through an Elastic Stack-based Platform, *2021 8th International Conference on Future Internet of Things and Cloud (FiCloud)*, pp. 184–191 (online), DOI: 10.1109/FiCloud49777.2021.00034 (2021).

[7] Patel, S. K., Rathod, V. and Parikh, S.: Joomla, Drupal and WordPress - a statistical comparison of open source CMS, *3rd International Conference on Trendz in Information Sciences Computing (TISC2011)*, pp. 182–187 (online), DOI: 10.1109/TISC.2011.6169111 (2011).

[8] Patel, S. K., Rathod, V. and Prajapati, J. B.: Performance analysis of content management systems-joomla, drupal and wordpress, *International Journal of Computer Applications*, Vol. 21, No. 4, pp. 39–43 (2011).

[9] Vethanayagam, S.: Threat Identification from Access Logs Using Elastic Stack (2020).

[10] Subramanian, K. and Meng, W.: Threat Hunting Using Elastic Stack: An Evaluation, *2021 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pp. 1–6 (online), DOI: 10.1109/SOLI54607.2021.9672347 (2021).

[11] Du, M., Li, F., Zheng, G. and Srikumar, V.: Deeplog: Anomaly detection and diagnosis from system logs through deep learning, *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 1285–1298 (2017).

[12] Bajer, M.: Building an IoT Data Hub with Elasticsearch, Logstash and Kibana, *2017 5th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*, pp. 63–68 (online), DOI: 10.1109/FiCloudW.2017.101 (2017).

[13] Ahmed, F., Jahangir, U., Rahim, H., Ali, K. and Agha, D.-e.-S.: Centralized Log Management Using Elasticsearch, Logstash and Kibana, *2020 International Conference on Information Science and Communication Technology (ICISCT)*, pp. 1–7 (online), DOI: 10.1109/ICISCT49550.2020.9080053 (2020).

[14] Laoutaris, N., Syntila, S. and Stavrakakis, I.: Meta algorithms for hierarchical Web caches, *IEEE International Conference on Performance, Computing, and Communications, 2004*, pp. 445–452 (online), DOI: 10.1109/PCCC.2004.1395054 (2004).

[15] Ishii, Y., Inaba, M. and Hiraki, K.: Access map pattern matching for high performance data cache prefetch, *Journal of Instruction-Level Parallelism*, Vol. 13, No. 2011, pp. 1–24 (2011).

[16] Ali, W., Shamsuddin, S. M., Ismail, A. S. et al.: A survey of web caching and prefetching, *Int. J. Advance. Soft Comput. Appl*, Vol. 3, No. 1, pp. 18–44 (2011).

[17] Nesbit, K. and Smith, J.: Data Cache Prefetching Using a Global History Buffer, *10th International Symposium on High Performance Computer Architecture (HPCA'04)*, pp. 96–96 (online), DOI: 10.1109/HPCA.2004.10030 (2004).

[18] Ishii, Y., Inaba, M. and Hiraki, K.: Access map pattern matching for data cache prefetch, *Proceedings of the 23rd international conference on Supercomputing*, pp. 499–500 (2009).

[19] Li, C., Song, M., Du, S., Wang, X., Zhang, M. and

Luo, Y.: Adaptive priority-based cache replacement and prediction-based cache prefetching in edge computing environment, *Journal of Network and Computer Applications*, Vol. 165, p. 102715 (online), DOI: https://doi.org/10.1016/j.jnca.2020.102715 (2020).

[20] Chen, Y., Byna, S. and Sun, X.-H.: Data access history cache and associated data prefetching mechanisms, *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*, pp. 1–12 (2007).

[21] Fielding, R. and Reschke, J.: RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): semantics and content (2014).

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

[23] Saito, T. L., Takezoe, N., Okada, Y., Shimamoto, T., Yu, D., Chandrashekharachar, S., Sasaki, K., Okumiya, S., Wang, Y., Kurihara, T. et al.: Journey of Migrating Millions of Queries on The Cloud, *Proceedings of the 2022 workshop on 9th International Workshop of Testing Database Systems*, pp. 10–16 (2022).

