問い合わせメールに含まれる日時をもちいたログのプリフェッチ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：管理者は障害が発生した際に原因を調査するためにログを検索する．その際に，キャッシュがあると検索の応答時間が速くなる．初めて検索するクエリの場合，プリフェッチをすることで応答時間が速くなる．課題は，既存の履歴ベースとコンテンツベースのプリフェッチ手法では時間の範囲が指定されておらず，キャッシュヒット率が低下することである．本稿では，サイトが閲覧できない場合に，サイト閲覧者の問い合わせメールの受信日時からプリフェッチする対象のログを決定する方法を提案する．提案方式では，問い合わせメールを受信した日時以前のステータスコードが 400 番台か 500 番台のログを検索する．ヒットしたログの中で問い合わせメールを受信した日時に最も近いログに含まれる日時を検索範囲の終了時刻に設定する．ステータスコードが 200 番台のログも同じ条件で検索し，問い合わせメールを受信した日時に最も近いログに含まれる日時を検索範囲の開始時刻に設定する．実験では，ログサーバーの 2024 年 6 月 24 日から 2024 年 10 月 31 日までの 130 日間に収集した 114,336,114 件のコンテナログを使用する．評価では 3 種類の実験を行い，1 つ目は提案ソフトウェアが作成したクエリと 3 人の管理者役の学生が作成した 3 つのクエリの時間差をそれぞれ求めた．3 つのクエリ全てで検索範囲の終了時刻の時間差は 20 分以下，検索範囲の開始時刻の時間差は 5 時間から 12 時間であった．2 つ目はキャッシュヒット率の算出を行った．管理者 A のキャッシュヒット率は 100%，管理者 B は約 29.5%，管理者 C は約 84.3%となった．3 つ目は，提案ソフトウェアのプリフェッチをした場合とプリフェッチをしない場合の検索時間の比較を行った．検索の応答時間の測定では各クエリで 10 回ずつ測定して中央値を求めた．キャッシュがない場合と比べてキャッシュがある場合は管理者 A で 1.5 秒から 0.8 秒になり約 46.7%の短縮，管理者 B で 1.7 秒から 0.7 秒になり約 58.8%の短縮，管理者 C で 1.5 秒から 0.6 秒になり約 60.0%短縮した．上記の結果から，管理者が作成する検索クエリの期間が短い方がプリフェッチするログの件数が少ないためキャッシュヒット率が高くなった．プリフェッチした場合の方がプリフェッチしない場合より検索の応答時間が短くなった．また，プリフェッチした期間によって応答時間の変化は見られなかった．

1. はじめに

背景

ログはシステムやアプリケーションの実行状況を記録する情報源のため，システム管理に利用される [1]. そのため，管理者はログを一元管理する必要がある [2]．ログを収集し管理するデータ分析フレームワークに Elastic Stack がある [3]．Elatic Stack は Elasticsearch，Logstash，Beats，Kibana で構成されている．Elatic Stack を使用してログを収集する流れを図 1 に示す．収集するログは Kubernetes の Pod を使用して構築している WordPress のコンテナログとする．WordPress とは，サイトを作成できるコンテンツ管理システムである [4]．ログをログサーバーに送信するために，コンテナ上で生成されたログは Beats から Logstash に送信される．Beats は，軽量のオープンプラットフォームであり，その 1 つに Filebeat がある [5]．Filebeat はサーバーからログを収集し Logstash に送信する．Logstash はログを解析するためのオープンソースフレームワークで Filebeat から送信されたログを Elasticsearch がログを読み込める形式に変換する [6, 7]．Elasticsearch は，全文検索機能を持ったオープンソースの検索エンジンでログを収集したログの管理や検索を行う [8]．収集されたログは指定されたインデックスごとに保存される．Kibana は Elasticsearch に保存したログを Web 上で検索やログの解析ができるオープンソースの可視化・分析ツールである [9].

管理者は，障害が発生した際に原因の分析や調査を行うためにログを検索する [10, 11]．管理者が障害を認識する

[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

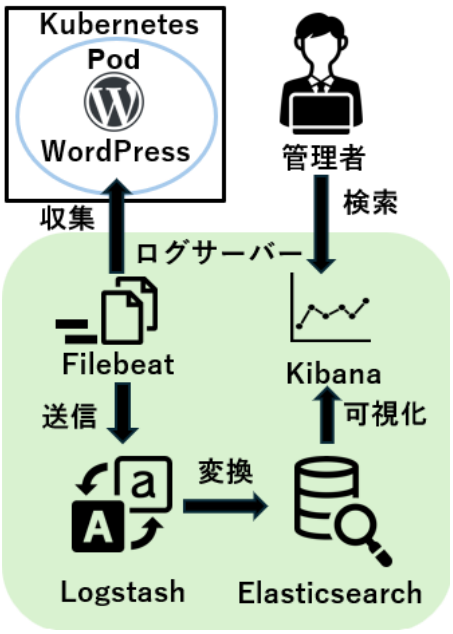

図 1 ログ収集の流れ

コード 1 エンドユーザーからの問い合わせをもとに原因調査を行う際の検索クエリの例

```
"query": {
        "range": {
            "timestamp": {
                "gte": "2024-10-31T17:30:00",
                "lt": "2024-10-31T18:05:00"
            }
        }
    }
```

方法としてエンドユーザーからの問い合わせメールがあげられる．検索には，JSON にもとづいた独自の検索クエリを使用している [12]．エンドユーザーからの問い合わせメールをもとに原因調査を行う際の検索クエリの例をコード 1 に示す．問い合わせメールは 2024 年 10 月 31 日 18 時 5 分に受信したとする．range は範囲検索をするためのフィールドである．timestamp はログの出力時間を表すフォーマットで，gte は greater than or equal to の略語である．gte を指定するとその時刻を以後で検索を行う．lt は less than or equal to の略語である．lt を指定するとその時刻以前で検索を行う．コード 1 は 2024 年 10 月 31 日 17:30 から 2024 年 10 月 31 日 18:05 に出力されたログを検索する．

ログ検索時にキャッシュがあると応答時間がキャッシュが無い時より速くなる．キャッシュとは情報の一部をアクセスが速いアクセスメモリに一時的に保存することで，次回以降の検索の際にを高速化する機能である [13, 14]．初回検索の応答時間を速くするためにはプリフェッチが効果的である．プリフェッチは検索される情報をあらかじめ予測してアクセスメモリに保存しておくことで，初回検索

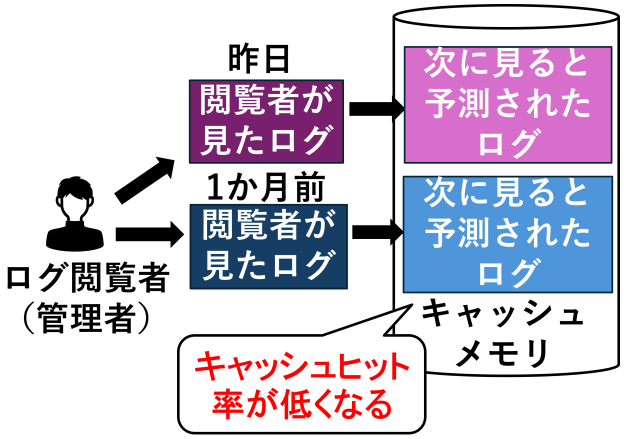

図 2 コンテンツベースのプリフェッチ方法

時もキャッシュがない時より応答時間を速くできる [15]．Elasticsearch にはプリフェッチ機能が無い．プリフェッチ機能を作成することでログ検索の応答時間が速くできるため，原因調査の時間を短くできる．全てのログをプリフェッチするにはメモリを大量に必要でありメモリ容量には上限があるため，プリフェッチするログを選ぶ必要がある．したがって，ログの管理者が検索するクエリに合わせたログをプリフェッチする必要がある．既存のプリフェッチ手法として，コンテンツベースと履歴ベースの 2 種類が存在する [16]．コンテンツベースの手法では，コンテンツの内容を解析することで，ユーザーが次にアクセスする可能性の高いコンテンツを予測し，ログを事前に取得する．履歴ベースの手法では，過去のアクセス履歴をもとに次に必要となるログデータを推測し，ログを先行して取得する．

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方法では，時間の範囲が指定されておらず，プリフェッチ範囲が広いことである．コンテンツベースのプリフェッチ方法を図 2 に示す．コンテンツベースのプリフェッチ方法では，管理者がログを見た際の内容から，次に見るログを予測してキャッシュメモリに入れる．その一方で，時間の指定がされておらず，キャッシュメモリに広範囲のログが入るため，キャッシュヒット率が低くなる．キャッシュヒット率が低下すると検索した際にキャッシュメモリと検索クエリにヒットするログが存在しないことが増えるため検索の応答時間が増え，原因調査に時間がかかる．

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利用したプリフェッチ手法を提案する論文がある [17]．この手法では，テーブルキーによるマッチングと履歴をもとにした代替的なプリフェッチ構造を提案しており，インデックステーブルとグローバル履歴バッファーで構成された FIFO テーブルを使用している．この手法では履歴ベースのプリフェッチを実現しており，閲覧者が閲覧したコンテンツがキャッシュに保存される仕組みである．また，キャッシュミスが発生すると該当するデータがインデックス化され，プリフェッチ候補として登録される．その結果，従来の手法に比べて効率的なプリフェッチが可能となる．その一方で，この提案は障害時の対応を考慮しておらず，障害が利用者のコンテンツ閲覧に依存せず発生するケースでは改善の余地がある．

アクセスマップパターンマッチング（以下，AMPM）プリフェッチ手法を提案する論文がある [18]．この手法では，メモリアドレス空間を固定サイズのゾーンに分割し，最近アクセスされたゾーンをホットゾーンとして検出する．AMPM プリフェッチャーはメモリアクセスパターンの並列マッチングを通じ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仕組みである．評価実験では，PC/DC および C/DC プリフェッチャーと比較し，AMPM プリフェッチャーがパフォーマンスを 42.0%向上させることが示された．その一方で，Web サイトの障害はアクセスされたゾーンに依存せず発生するため，障害が発生するユースケースでは改善の余地がある．

ベイジアンネットワーク理論にもとづくキャッシュプリフェッチ戦略を提案する論文がある [19]．この手法では，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し，応答時間を利益の指標，キャッシュスペースの占有をコストの指標として使用している．評価では，ベンチマークとの比較により，プリフェッチヒット率の向上や応答時間の短縮，さらにはメモリ負荷の大幅な削減が示された．その一方で，この提案は過去のユーザーのアクセス履歴をもとにしているため，障害時にどのログを見るべきか判断することは対象としていない．

データアクセス履歴キャッシュ（以下，DAHC）を使用した新しいキャッシュ構造を提案する論文がある [20]．この手法では，DAHC は従来の命令キャッシュやデータキャッシュとは異なり，最近の参照情報を格納するキャッシュとして機能する．この構造はデータアクセス履歴テーブル（DAH テーブル）と 2 つのインデックステーブルで構成され，履歴情報の詳細を記録する．SimpleScalar ツールセットを使用したシミュレーションでは，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比較して，キャッシュミス率が大幅に削減されることが示された．その一方で，この手法はデータのアクセス履歴を活用しているため，障害が利用者の行動に依存せず発生するユースケースでは改善の余地がある．

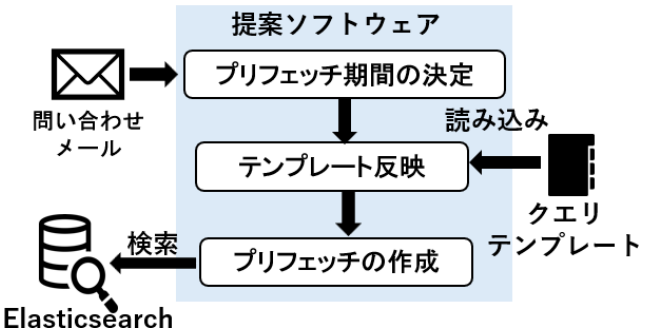

図 3 ソフトウェアの流れ

コード 2 テンプレートクエリ

```
"query": {
   "range": {
      "timestamp": {
          "gte": "(1)",
          "lt": "(2)"
      }
   }
}
```

3. 提案

問い合わせメールの受信日時を使用してクエリを決定し，プリフェッチする方式を提案する．提案手法を適用することで，障害発生時のログ検索の応答時間を短くすることを目的としている．前提条件を以下のように設定する．

- Web サイトを提供するための Web サーバーで障害が発生しており，閲覧者から問い合わせメールが送信されている．
- 閲覧者が問い合わせる前からシステム管理者が検索を行うまで，継続して障害が発生している．

提案手法

提案ソフトウェアの流れを図 3 に示す．提案ソフトウェアは問い合わせメールをテキスト形式で受け取り，形態素解析を行って問い合わせメールを受信した日時を取得する．次にテンプレートクエリに反映する．テンプレートクエリをコード 2 に示す．このクエリは，ログの出力時間で表示するログを絞り込むクエリである．本提案では timestamp の (1) に入る gte の値と (2) に入る lt の値を求める． timestamp の gte と lt の値の決め方を図 4 に示す．t_l は lt の日時，t_g は gte の日時とする．まず，t_l を求める．問い合わせメールを受信した日時以前のステータスコードが 400 番台か 500 番台のログを検索する．検索結果の中でログに含まれる timestamp が問い合わせ時刻より古いログの timestamp の値を t_l に設定する．次に，t_g を求め

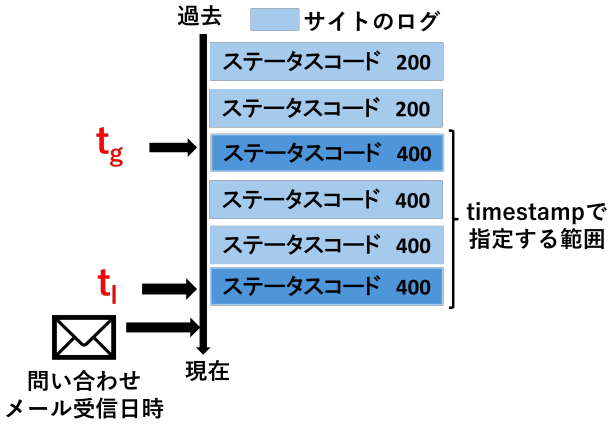

図 4　timestamp の gte と lt の決め方

る．ログのステータスコードが 200 番台のログを検索する．検索結果の中でログに含まれる timestamp が問い合わせ時刻より古いログの timestamp の値を t_g に設定する．最後にテンプレートクエリに求めた t_g と t_l の値を反映し，Elasticsearch で検索を行う．これにより Elasticsearch の機能でキャッシュが作成される．提案ソフトウェアは問い合わせメールが受信されたタイミングで実行される．

ユースケース・シナリオ

企業が Kubernetes クラスタで WordPress を用いて Web サイトを運用している状況を想定する．問い合わせメールから原因調査をする流れを図 5 に示す．Web サイトの 3 つの Pod で構築されており WordPress で作成されている．管理者はログサーバーでコンテナログを収集し，Web サイトの管理を行っている．WordPress のコンテナログは Filebeat で収集を行っている．収集したログは Logstash で変換し，Elasticsearch に送信する．管理者は Elasticsearch でログの検索を行う．閲覧者はサイトがステータスコード 404 で閲覧ができなかった場合，カスタマーサポートに問い合わせのメールを送信する．問い合わせメールを受けたカスタマーサポートは管理者に原因調査を依頼する．管理者は問い合わせメールの内容を確認し，原因調査を行うためログを検索する．その際にプリフェッチを行うことで検索を速くできる．

4.　実装

提案ソフトウェアの実装について図 6 に示す．setting.py ではテキスト形式にした問い合わせメールを形態素解析し，リストに格納する．格納したリストから日時を取得する．その後，問い合わせメールに 1 番近いステータスコードが 400 番台と 500 番台のログを検索し，timestamp の値を取得し，lt の値として取得する．次に，ステータスコードが 200 番台のログの timestamp を取得し gte の値として取得する．次に取得した値を query.txt に反映させ，newquery.txt として出力する．最後に newquery.txt の検索クエリを使用して，search.py で Elasticsearch にログの検索を行うことでキャッシュが作成される．キャッシュメモリの容量が最大だった場合は，LRU でログが削除される．LRU は最後に検索された日時が 1 番古いログを削除する方式である [21]．

timestamp の値を算出する setting.py，プリフェッチを作成するために検索を行う search.py の 2 つの Python ファイルを作成し実装を行った．使用するモジュール一覧を表 1 に表す．os モジュールは os の機能を扱うために使用する．json モジュールは JSON 形式で保存されたログを扱うために使用する．time モジュールと datetime モジュールは日時を扱うために使用する．subprocess モジュールはファイル内で他の Python ファイルを実行するために使用する．elasticsearch モジュールは Elasticsearch に検索をする際に使用する．logging モジュールはログの出力結果を表示する際に使用する．

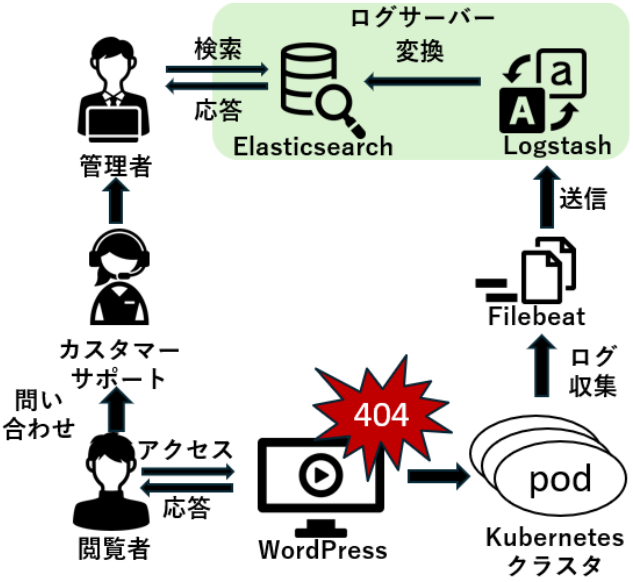

図 5　問い合わせメールから原因調査をする流れの図

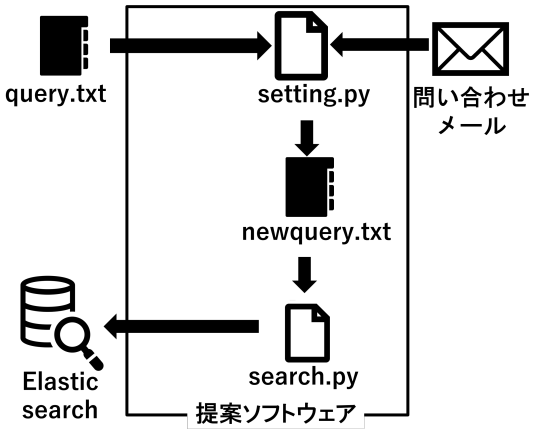

図 6　提案ソフトウェアの実装

5. 評価実験

3 種類の実験を行った．1 つ目は，管理者役の学生が発行した検索クエリと提案ソフトウェアが発行した検索クエリの比較を行った．2 つ目は，管理者役の学生が発行したクエリに対して提案ソフトウェアが発行したクエリがどの程度ヒットするのかをキャッシュヒット率で求める実験を行った．3 つ目は，提案ソフトウェアのプリフェッチがある場合とプリフェッチがない場合の検索の応答時間を比較した．

実験環境

ログサーバーに保存されたログを実験に使用する．ログサーバーには Kubernetes クラスタにあるコンテナログが保存されている．ログサーバーは 6 台の仮想マシンで構成されており，Elasticsearch，Logstash，Kibana が動作している．Kubernetes クラスタを使用しマスターノードが 1 台，ワーカーノードが 5 台で構成されている．全ての Node は，vCPU が 5 コア，メモリが 16GB，ストレージが 25GB で構成されている．ログサーバーには 2024 年 6 月 24 日から 2024 年 10 月 31 日までの 130 日間に収集した 114,336,114 件のログが保存されている．3 つのサーバーからコンテナログを収集している．1 つ目は Kubernetes 共有クラスタの unikube サーバーである．2 つ目は基幹サーバーである．3 つ目は 3 年生向けの Cloud and Distributed Systems Laboratory(以下，CDSL) 研究室紹介サイトが構築されているサーバーである．3 つのサーバーは全て Kubernetes クラスタを使用している．

(実験 1) 検索クエリの比較

管理者役の学生が問い合わせメールを閲覧し，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う．問い合わせメールは東京工科大学のコンピュータサイエンス学部の閲覧者役のの学生が 3 年生向けの CDSL 研究室紹介サイトを閲覧して作成する．メールの書き方は指定しない．前提条件として，閲覧者役の学生はサイトが見れないと確認してすぐにメールを送信したとする．使用する問い合わせメールを図 7 に示す．このメールは 2024 年 10 月 31 日 18:05 に送信されたメールである．3 年生向けの CDSL 研究室紹介サイトの一部ページにアクセスが出来ない旨が書かれている． 想定するシナリオは，3 年生向

表 1　使用するモジュール一覧

モジュール名	用途	使用しているファイル名
os	オペレーティングシステムを扱う	setting.py search.py
json	JSON 形式のデータを扱う	search.py
time	時間を扱う	search.py
subprocess	他のプログラムやコマンドを実行する	setting.py
datetime	日時を扱う	setting.py search.py
elasticsearch	Elasticsearch を扱う	setting.py search.py
logging	ログの出力を行う	search.py

図 7　使用する問い合わせメール

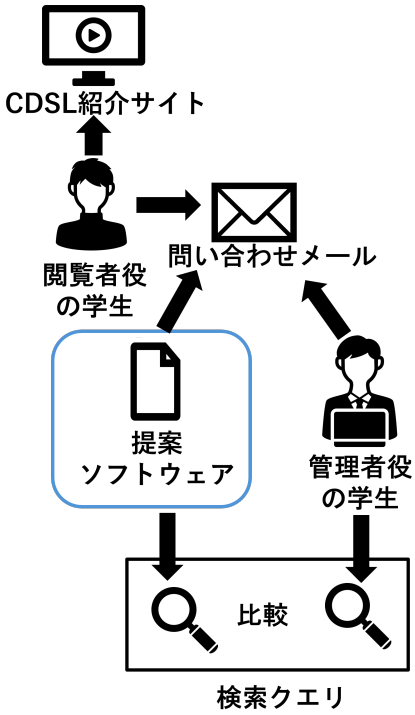

図 8　検索クエリの比較の実験方法

コード 3 提案ソフトウェアが発行した検索クエリ

```
"query": {
   "range": {
      "timestamp": {
         "gte": "2024-10-31T11:16:57",
         "lt": "2024-10-31T18:16:08"
      }
   }
}
```

けの CDSL 研究室紹介サイトにアクセスした際を想定する．ログサーバーに 3 年生向けの CDSL 研究室紹介サイトのコンテナログを収集している．ホームページとクラウドページと IoT ページの 3 ページで構成されている．ホームページにはクラウドページと IoT ページに遷移できるボタンがある．閲覧者がホームページからクラウドページに遷移した際にステータスコード 404 になる．これは，サイト作成者がクラウドページに遷移するボタンの URL を間違えて設定したためである．閲覧者役の学生はこのページを見た際に管理者役の学生に問い合わせを行った．検索クエリの評価実験の方法を図 8 に示す．閲覧者役の学生は 3 年生向けの CDSL 研究室紹介サイトを閲覧し，問い合わせメールを作成する．次に問い合わせメールを管理者役の学生と提案ソフトウェアに渡し，検索クエリを作成する．提案ソフトウェアは問い合わせメールの受信日時から検索クエリを作成する．管理者役の学生は問い合わせメールを読み，timestamp の gte と lt に日時を入力し，検索クエリの作成を行う．管理者役の学生は東京工科大学のコンピュータサイエンス学部の CDSL の学生 A，学生 B，学生 C(以下，管理者 A，管理者 B，管理者 C とする) の 3 名である．提案ソフトウェアと管理者役の学生が発行した検索クエリを比較して timestamp の時間範囲にどの程度，時間差があるのかを算出する．上記のシナリオでの提案ソフトウェアが発行した検索クエリをコード 3 に示す．提案ソフトウェアが発行した検索クエリは 2024 年 10 月 31 日の 11 時 16 分 57 秒から 2024 年 10 月 31 日の 18 時 16 分 8 秒に出力されたログのプリフェッチを行っている．この検索クエリと管理者役の学生が発行した検索クエリの比較を行う．

提案ソフトウェアが作成したクエリと学生が作成したクエリの結果を表 2 に示す．gte と lt は作成したそれぞれの検索クエリの値である．提案ソフトウェアの検索クエリと管理者役の学生の検索クエリの時間差を表 3 に示す．時間差は提案ソフトウェアの発行した検索クエリが管理者役の学生の検索クエリと比べてどのくらいかを表している．時間差は，表 2 の提案ソフトウェアが発行した検索クエリの日時から管理者役の学生が発行した検索クエリの日時を引いた値である．例えば，管理者 A の gte の時間差を求める場合は，提案ソフトウェアの gte の 2024-10-31T11:16:57 から管理者 A の gte の 2024-10-31T17:30:00 を引いた値で

表 2 提案ソフトウェアと管理者役の学生が作成したクエリの結果

	gte	lt
提案ソフトウェア	2024-10-31T 11:16:57	2024-10-31T 18:16:08
管理者 A	2024-10-31T 17:30:00	2024-10-31T 18:05:00
管理者 B	2024-10-31T 00:00:00	2024-10-31T 18:00:00
管理者 C	2024-10-31T 17:00:00	2024-10-31T 18:30:00

表 3 提案ソフトウェアの検索クエリと管理者役の学生の検索クエリの時間差

	gte	lt
管理者 A	6 時間 13 分 3 秒多い	11 分 8 秒多い
管理者 B	11 時間 16 分 57 秒少ない	16 分 8 秒多い
管理者 C	5 時間 43 分 3 秒多い	13 分 52 秒少ない

ある．したがって，提案ソフトウェアが発行した gte は管理者 A が発行した gte より 6 時間 13 分余分にプリフェッチした．gte と lt の時間差はどちらも 0 分になる状態が理想である．結果は，lt は 3 つのクエリで時間差が 20 分以下であったが，gte は 3 つのクエリで時間差が 5 時間から 12 時間であった．gte の時間差が大きい理由として，障害の発生開始の予測が人によって大きく異なることがあげられる．

(実験 2) キャッシュヒット率

提案ソフトウェアの検索クエリが管理者役の学生の検索クエリに対してどのくらいヒットしているかの評価を行う．検索クエリの比較の際に管理者役の学生が発行したクエリ 3 つと提案ソフトウェアが発行したクエリを使用する．キャッシュヒット率を式 (1) で求める．r_1 はキャッシュヒット率を表している．h_1[件] はキャッシュヒットしたログの件数である．管理者役の学生の検索クエリと提案ソフトウェアの検索クエリどちらにもヒットしたログをキャッシュヒットとする．m[件] はキャッシュミスしたログの件数を表している．管理者役の学生のクエリのみにヒットし，提案ソフトウェアの検索クエリではヒットしなかったログをキャッシュミスとする．

$$r_1 = \frac{h_1}{h_1 + m} \tag{1}$$

r_1　キャッシュヒット率
h_1　キャッシュヒットしたログの件数 [件]
m　キャッシュミスしたログの件数 [件]

提案ソフトウェアがプリフェッチしたログに対してヒットした割合を式 (2) で求める．r_2 は提案ソフトウェアがプリフェッチしたログに対してヒットした割合を表している．h_2[件] はキャッシュヒットしたログの件数である．管理者役の学生の検索クエリと提案ソフトウェアの検索クエリど

ちらにもヒットしたログをキャッシュヒットとする．p[件]は提案ソフトウェアがプリフェッチしたログの件数を表している．

$$r_2 = \frac{h_2}{p} \tag{2}$$

r_2　提案ソフトウェアがプリフェッチしたログに対してヒットした割合
h_2　キャッシュヒットしたログの件数 [件]
p　提案ソフトウェアがプリフェッチしたログの件数 [件]

キャッシュヒット率とその内訳を表 4 に示す．キャッシュヒット率は式 (1) を使用して算出する．キャッシュヒット率は管理者 A は 33,670/(33,670+0) より 100%，管理者 B は 492,618/(492,618+1,177,554) より約 29.5%，管理者 C は 75,608/(75,608+14,052) より約 84.3%となった．キャッシュヒットしたログの件数は管理者 A が 33,670 件，管理者 B が 492,618 件，管理者 C が 75,608 である．キャッシュミスしたログの件数は管理者 A が 0 件，管理者 B が 1,177,554 件，管理者 C が 14,052 件である．管理者 A は全てのログをプリフェッチすることができたが，提案ソフトウェアが余分にプリフェッチしたログの件数が 1 番多かった．管理者 B はキャッシュミスのログの件数が多いためキャッシュヒット率が低いが，提案ソフトウェアが余分にプリフェッチしたログの件数が 1 番少なかった．管理者 C は，キャッシュヒット率と提案ソフトウェアがプリフェッチしたログに対してヒットした割合が 2 番目に高く，時間差も 2 番目に少なかった．検索の応答時間の結果から提案ソフトウェアと管理者役の学生が発行した検索クエリの時間差がなかった管理者 A はキャッシュヒット率が 1 番高かった．時間差が大きかった管理者 B はキャッシュヒット率が 1 番低かった．したがって，提案ソフトウェアと検索クエリの時間差が大きいほど，キャッシュヒット率が低くなる．

表 4　キャッシュヒット率と内訳

	キャッシュヒット率	キャッシュヒットしたログの件数	キャッシュミスしたログの件数
管理者 A	100%	33,670	0
管理者 B	29.5%	492,618	1,177,554
管理者 C	84.3%	75,608	14,052

提案ソフトウェアがプリフェッチしたログに対してキャッシュヒットした割合と内訳を表 5 に示す．提案ソフトウェアがプリフェッチしたログに対してヒットした割合は式 (2) を使用して算出する．p は 510,378 件である．提案ソフトウェアがプリフェッチしたログに対してキャッシュヒットした割合が高いほど，余分にプリフェッチしたログの件数が少なく望ましい．提案ソフトウェアがプリフェッチしたログに対してヒットした割合は管理者 A は 33,670/510,378 より約 6.6%，管理者 B は 492,618/510,378 より約 96.5%，管理者 C は 75,608/510,378 より約 14.8%となった．キャッシュヒットしたログの件数は管理者 A が 33,670 件，管理者 B が 492,618 件，管理者 C が 75,608 件である．キャッシュヒット率と提案ソフトウェアがプリフェッチしたログに対してヒットした割合の結果から管理者 A と管理者 C はキャッシュヒット率が高い．一方で余分にプリフェッチしてしまったログの件数が管理者 B より多い．管理者 B はキャッシュヒット率が低いが提案ソフトウェアがプリフェッチしたログに対してヒットした割合が高いため，提案ソフトウェアが発行した検索クエリが管理者役の学生が発行した検索クエリよりプリフェッチした timestamp の時間範囲が狭い．

表 5　提案ソフトウェアがプリフェッチしたログに対してヒットした割合と内訳

	提案ソフトウェアがプリフェッチしたログに対してヒットした割合	キャッシュヒットしたログの件数
管理者 A	6.6%	33,670
管理者 B	96.5%	492,618
管理者 C	14.8%	75,608

表 4 のキャッシュヒット率と表 5 の提案ソフトウェアがプリフェッチしたログに対してヒットした割合がどちらも 100%になることが理想である．表 4 から管理者 A と管理者 C が発行した検索クエリはキャッシュヒット率が 80%以上であり，管理者 B は 29.5%であった．5 から管理者 A と管理者 C が発行した検索クエリは提案ソフトウェアがプリフェッチしたログのに対してヒットした割合が 20%以下であり，管理者 B は 96.5%であった．以上の結果から，管理者 A と管理者 C はキャッシュヒット率が 80%以上と高く，提案ソフトウェアがプリフェッチしたログのに対してヒットした割合が 20%以下と低い．したがって，管理者 A と管理者 C は提案ソフトウェアが発行した検索クエリが，管理者役の学生が発行した検索クエリに対して余分にプリフェッチしたログの件数が多い．一方で，管理者 B はキャッシュヒット率が 29.5%と低く，提案ソフトウェアがプリフェッチしたログのに対してヒットした割合が 96.5%と高い．したがって，管理者 B は提案ソフトウェアが発行した検索クエリが，管理者役の学生が発行した検索クエリに対してプリフェッチしたログの件数が少ない．

(実験 3) 検索の応答時間

提案ソフトウェアが作成した検索クエリのプリフェッチをした場合とプリフェッチをしない場合の検索の応答時間の比較を行う．使用する問い合わせメールは実験 1 の検索クエリの比較の際に使用する問い合わせメールと同じである．検索クエリは管理者役の学生 3 名が作成した検索クエ

リを使用する．プリフェッチした場合とプリフェッチしない場合でそれぞれ 10 回ずつ測定する．プリフェッチありの場合は提案ソフトウェアを実行しプリフェッチを行ってから，管理者役の学生が作成した検索クエリを使用して検索を行う．プリフェッチなしの場合は，検索の際に提案ソフトウェアの実行を行わない．どちらの検索時も全てのキャッシュを削除してから行う．

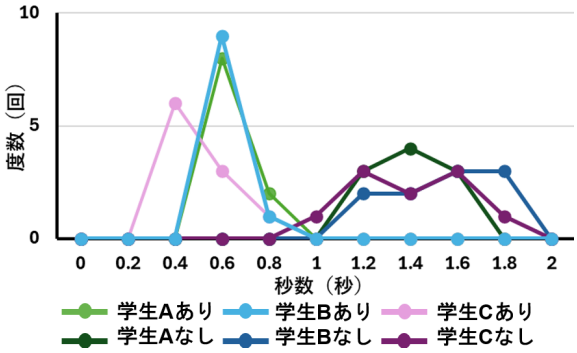

図 9 プリフェッチをした場合とプリフェッチをしない場合の検索の応答時間

プリフェッチをした場合とプリフェッチをしない場合の検索の応答時間を図 9 に示す．度数分布図で縦軸が度数，横軸が秒数である．それぞれ 10 回ずつ計測を行った．管理者 A のプリフェッチありの最頻値は 0.4 秒以上 0.6 秒未満で 8 回，プリフェッチなしの最頻値は 1.2 秒以上 1.4 秒未満で 4 回であった．管理者 B のプリフェッチありの最頻値は 0.4 秒以上 0.6 秒未満で 9 回，プリフェッチなしの最頻値は 1.4 秒以上 1.6 秒未満と 1.6 秒以上 1.8 秒未満で 3 回ずつであった．管理者 C のプリフェッチありの最頻値は 0.2 秒以上 0.4 秒未満で 6 回，プリフェッチなしの最頻値は 1.0 秒以上 1.2 秒未満と 1.4 秒以上 1.6 秒未満でそれぞれ 3 回ずつであった．管理者 A のプリフェッチありの中央値は 0.75 秒，プリフェッチなしの中央値は 1.48 秒であった．管理者 B のプリフェッチありの中央値は 0.70 秒，プリフェッチなしの中央値は 1.67 秒であった．管理者 C のプリフェッチありの中央値は 0.57 秒，プリフェッチなしの中央値は 1.49 秒であった．管理者 A は 35 分の範囲で 33,670 件のログ，管理者 B は 18 時間の範囲で 1,670,172 件のログ，管理者 C は 30 分の範囲で 89,660 件のログをプリフェッチしている．プリフェッチをしない場合はプリフェッチした時間範囲が 1 番長い管理者 B が 1 番検索の応答時間が遅い．プリフェッチをした場合はプリフェッチした時間範囲が 7 時間 30 分差がある管理者 A と管理者 B の最頻値が同じであった．原因としてプリフェッチした時間範囲が狭いことがあげられる．そのため，プリフェッチの範囲を変えて検索の応答時間を比較する必要がある．プリフェッチをした場合と比べてプリフェッチをしない場合は管理者 A で 1.5 秒から 0.8 秒になり約 46.7%の短縮，管理者 B で 1.7 秒から 0.7 秒になり約 58.8%の短縮，管理者 C で 1.5 秒から 0.6 秒になり約 60.0%短縮した．これらの結果から提案ソフトウェアで作成した検索クエリのプリフェッチをした場合の方がプリフェッチをしない場合より検索の応答時間が速くなる．したがって，プリフェッチが機能しているといえる．

6. 議論

本提案では，メールの受信日時のみを使用しており，メール本文の内容を使用していない．そのため，メールの本文に事象の発生日時やエラーの発生する URL が書いてあるとしても使用できない．解決方法として，テキストパーサーでメール文を解析し，必要な情報を抽出する．例として，サイトの URL をあげる．テキストパーサーを使用して形態素解析を行い，各単語ごとにリストに格納する．格納したリストの中に http から始まる文字列やサイトのページ名や名前があった場合は，その値を取得し，テンプレートに追記する．これにより，問い合わせメールに書かれている内容を含む検索クエリを作成できる．

本提案では，問い合わせメールが送られてきた際にソフトウェアが実行される．そのため，障害に関係ないメールが送られてきた場合もクエリを作成してしまう．解決方法として，テキストパーサーを使用して問い合わせメールの内容が障害が起きている可能性があるかないかを判断する方法をあげる．メールを取得したら API でテキストパーサーに問い合わせメールの内容が障害に関係するか判断し，障害に関係があった場合のみ検索クエリを作成する．これにより，不必要なログがメモリを圧迫を防げる．

本提案では，管理者によって検索クエリが異なることを考慮していない．そのため，管理者が発行する検索クエリによっては，時間差が大きくなり，キャッシュヒット率が下がる，または余分なログをプリフェッチする．解決方法として，管理者が過去に原因調査時に検索したクエリの時間範囲をもとにクエリを決定する．原因調査時に使われたクエリを保存する．保存された検索クエリから timestamp の時間範囲がどのくらいかの平均を出し，gte の値を変更する．管理者ごとに時間差が少ない検索クエリの発行を行うことができる．

7. おわりに

Web サイトが閲覧できなくなった際，サイト閲覧者は Web サイトの問い合わせメールアドレスに問い合わせメールを送信する．管理者は障害発生時に原因を調査するために検索クエリを使用してログを検索する．その際にキャッシュがあると検索応答時間が速くなる．初めて検索する検索クエリはプリフェッチをすることで検索応答時間を速くできる．課題は既存のプリフェッチ方法は時間の範囲が指定されていないため，障害時はキャッシュヒット率が

下がる．そのため，問い合わせメールの受信日時からプリフェッチするログの時間範囲を絞る手法を提案する．評価実験では，3 種類の実験を行った．1 つ目は管理者役の学生が問い合わせメールを閲覧して作成したクエリと提案ソフトウェアが作成したクエリの比較を行った．検索クエリの比較の結果，lt は 3 つのクエリとも時間差が 20 分以下であったが，gte は時間差が 5 時間から 12 時間であった．2 つ目はキャッシュヒット率の算出を行った．キャッシュヒット率の結果は管理者 A が 100%，管理者 B が約 29.5%，管理者 C が約 84.3%となった．3 つ目は提案ソフトウェアで作成した検索クエリのプリフェッチをした場合とプリフェッチをしない場合の検索の応答時間の測定を行った．検索の応答時間の測定の結果は，キャッシュがない場合と比べてキャッシュがある場合は，管理者 A で 1.5 秒から 0.8 秒になり約 46.7%の短縮，管理者 B で 1.7 秒から 0.7 秒になり約 58.8%の短縮，管理者 C で 1.5 秒から 0.6 秒になり約 60.0%短縮した．上記の結果から，管理者が作成する検索クエリの期間が短い方がプリフェッチするログの件数が少ないためキャッシュヒット率が高くなった．プリフェッチした場合の方がプリフェッチしない場合より検索の応答時間が短くなった．また，プリフェッチした期間によって応答時間の変化は見られなかった．

謝辞

本稿の執筆にあたり東京工科大学コンピュータサイエンス学部の西村克己さんに問い合わせメールの作成を行って頂きました．また，東京工科大学コンピュータサイエンス学部の近藤悠斗さん，三上翔太さん，加藤健吾さんに検索クエリの作成を行って頂きました．また，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平尾真斗さんにご助言を賜りました．上記の方々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] Rochim, A. F., Aziz, M. A. and Fauzi, A.: Design Log Management System of Computer Network Devices Infrastructures Based on ELK Stack, *2019 International Conference on Electrical Engineering and Computer Science (ICECOS)*, pp. 338–342 (online), DOI: 10.1109/ICECOS47637.2019.8984494 (2019).

[3] Chen, L., Liu, J., Xian, M. and Wang, H.: Docker Container Log Collection and Analysis System Based on ELK, *2020 International Conference on Computer Information and Big Data Applications (CIBDA)*, pp. 317–320 (online), DOI: 10.1109/CIBDA50819.2020.00078 (2020).

[4] 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).

[5] Lertwuthikarn, T., Barroso, V. C. and Akkarajitsakul, K.: Resource Optimization for Log Shipper and Preprocessing Pipeline in a Large-Scale Logging System, *2022 IEEE 5th International Conference on Knowledge Innovation and Invention (ICKII)*, pp. 196–200 (online), DOI: 10.1109/ICKII55100.2022.9983590 (2022).

[6] Doan, D. N. and Iuhasz, G.: Tuning Logstash Garbage Collection for High Throughput in a Monitoring Platform, *2016 18th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 359–365 (online), DOI: 10.1109/SYNASC.2016.063 (2016).

[7] 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).

[8] 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).

[9] Bhatnagar, D., SubaLakshmi, R. J. and Vanmathi, C.: Twitter Sentiment Analysis Using Elasticsearch, LOGSTASH And KIBANA, *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*, pp. 1–5 (online), DOI: 10.1109/ic-ETITE47903.2020.351 (2020).

[10] 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)*, pp. 121–130 (online), DOI: 10.1109/ICSE-SEIP.2019.00021 (2019).

[11] 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).

[12] Akdal, B., Çabuk Keskin, Z. G., Ekinci, E. E. and Kardast, G.: Model-Driven Query Generation for Elasticsearch, *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pp. 853–862 (2018).

[13] 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).

[14] 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).

[15] Byna, S., Chen, Y. and Sun, X.-H.: A Taxonomy of Data Prefetching Mechanisms, *2008 International Symposium on Parallel Architectures, Algorithms, and Networks (i-span 2008)*, pp. 19–24 (online), DOI: 10.1109/I-SPAN.2008.24 (2008).

[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] Li, Z.-s., Liu, D.-w. and Bi, H.-j.: CRFP: A Novel Adaptive Replacement Policy Combined the LRU and LFU Policies, *2008 IEEE 8th International Conference on Computer and Information Technology Workshops*, pp. 72–79 (online), DOI: 10.1109/CIT.2008.Workshops.22 (2008).

