事前ソートによるWebアクセスログの検索時間の削減

大野有樹[1]　小山智之[2]　串田 高幸[1]

概要：Web サービスにおいてシステム障害が起きたとき，システム管理者はアクセスログを検索することで障害の原因の絞り込みや発生時刻を特定している．障害の原因特定を早くする方法はログの検索の応答時間を削減させることである．障害の対応が遅くなることは，例えば SLA の違約金を含む経済的損失につながる．システム管理者が障害を解消するためのログ検索における課題は検索結果のソートで検索に遅延が生じることである．なぜならアプリケーションノードごとにログがアクセス時刻順に生成されるが，複数のアプリケーションノードからログを集める場合，ログがアクセス時刻順に並ばないためである．また，検索対象が多くなるほど，検索の応答時間は長くなる．本提案は，あらかじめログをアクセス時刻を基準にソートして，検索対象を絞り込むためにステータスコードごとに分割して保存することでログ検索の応答時間を削減する．実験はログの配置と検索クエリを変えたときの検索時間を比較することで評価する．本提案は，障害が解消するまでの時間に含まれるログ検索の応答時間の削減により，経済損失を抑えることに貢献できる．

1. はじめに

背景

ログはシステム実行中の動作の詳細を記録した唯一のデータである [1]．本研究は Web サービスにおけるアクセスログを対象とする．Web サービスは HTTP のアクセスログを出すアプリケーションを Web サービスと置く．本研究は 1 分間に 1 万のリクエストが来る Web サイトを想定する．これは 1998 年のサッカーワールドカップの Web サイトを平均すると 1 分間に 1 万リクエストがあった事例を基準にしている [2]．ログは 1 リクエストにつき 1 件生成されるものとする．本研究で扱うシステム障害は Web サービスの利用者が Web サービスにアクセスできない状態，HTTP レスポンスステータスコード 503 エラーが発生する状態と定義する．

システム管理者はシステム障害を解消するためにログを検索する．システム管理者はログを検索することでシステム障害の原因の絞り込みや発生時刻を特定している．ログ検索の応答時間はシステム管理者がログサーバーに検索クエリを発行してから、検索結果がシステム管理者へ返ってくるまでの時間とする．検索の応答時間はシステム障害が解消するまでの時間に含まれる．したがって，検索の応答時間の増加はシステム障害が解消するまでの時間の増加である．

システム障害解消までの時間が増加することはシステムを管理している組織の経済的損失をもたらす [3]．例えば SLA の違約金の発生における損失である．SLA の保証する水準にはサービス稼働時間があげられる．サービス稼働時間を 99.5%[*1]とすると，1 ヶ月で 3.6 時間以内にシステム障害を収める必要がある．

障害対応の内，ログ検索の時間を用いる時間がどれだけ含まれているかの例を示す．今回、例としてあげる状況は「プログラム不備によりデータ不整合が発生しオンラインサービスでシステムエラーが発生した．暫定対応としてデータパッチを実施して解消した.」場合である[*2]．障害発生から恒久対応を除く暫定対応完了までの間を以下の 4 ステップに分ける．4 ステップは事象確認，調査，暫定対応，暫定対応内容確認とする．事象確認は障害の検知、サービスへの影響が有るか無いかの確認である．調査は障害の影響調査、対象サーバのログ取得・調査である．暫定対応は調査を基にした暫定対応内容検討、暫定対応実施である．暫定対応内容確認は暫定対応後の確認である．そのうち，ログ検索が含まれる対応のステップは調査と暫定対応内容確認である．システム障害の対応時間の全体である 94 分のうち，調査が 8 分で暫定対応の内容確認が 6 分と計 14 分であり，全体の約 15%をログ検索が含まれる対応のステッ

[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

[*1] https://moneyforward.com/pages/premium
[*2] https://atmarkit.itmedia.co.jp/ait/articles/1705/09/news009.html

プが占める．したがって，ログ検索の時間を削減することでシステム障害解消までの時間を削減できる．

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 [4]．並列化手法のひとつである分散手法はひとつのノードにログを配置して検索するのではなく，複数ノードにログを配置して並列で検索する手法である．分散手法の具体的な例は scatter-gather がある．scatter-gather は分散を制御するルートノードと，検索をするリーフノードの 2 種類のノードを使ったリクエスト処理の並列化手法である．

システム管理者はシステム障害を解消するためにログを検索して原因を特定する．このときログはアクセス時刻を基準に並んでいる必要がある．システム管理者は，過去のログを遡ることでシステム障害の根本原因やきっかけを辿ることができる．システム障害は前後関係があり、順序だてて起こる．システム管理者はアクセス時刻の順番に並んでいるログを参照してシステム障害の原因を特定する．

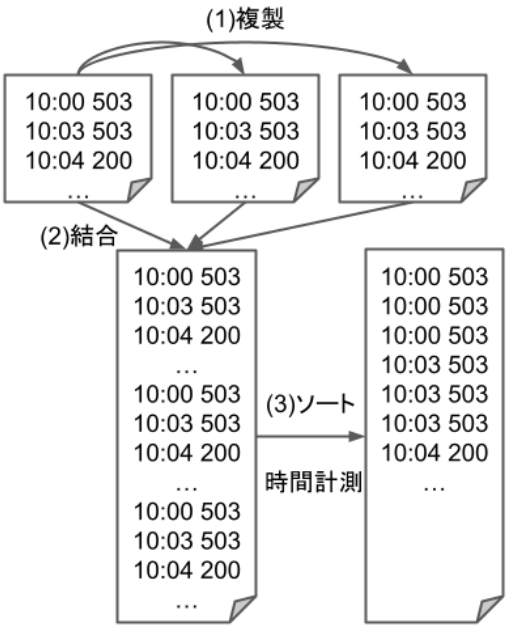

図 1　基礎実験の手順

課題

システム障害発生時のログ検索における課題は，複数のアプリケーションノードからログを集めることで検索以外の処理であるアクセス時刻を基準にしたソートが，検索の応答時間に含まれることである．ソートが含まれることで，ログ検索の応答時間がその分増加する．また，検索の対象となるログが増えるほど検索の応答時間が増加する．

実際にソートによってどれだけ時間がかかるか予備実験をした．予備実験に用いたログは WordPress を使用したブログサイト[*3]のアクセスログである．ログの期間は 13 日，件数は 24,784 件である．予備実験はログ件数とソートの時間の関係性を調べるため，ログ件数を増やしてソートにかかる時間を計測した．図 1 は，予備実験で使用したログの複製方法と予備実験の手順を示している．ソートの対象となるログはアクセスログ 24,784 件を複製して増やした．測定した結果は 3 つであり，アクセスログ 1 つのみをソートした結果，もととなるアクセスログと複製したログを結合してソートした結果，もととなるアクセスログと複製したログ 2 つを結合してソートした結果である．ソートはアクセス時刻を基準に行った．ソート時間はログをソートしはじめてからソートし終わるまでとする．ソート時間はソート処理を 10 回行った平均値から算出した．ソートのアルゴリズムは，Linux の sort コマンドのアルゴリズムである，マージソートを使用した[*4]．計測は Linux の time コマンドを使用した．

図 2 は予備実験の結果を示している．図 2 より，ログ件数とソート時間は比例関係を示していた．ソート時間は検索の応答時間の中に含まれる．すなわち，ログ件数が増えるほどログ検索の応答時間がソートによって増えることを示している．

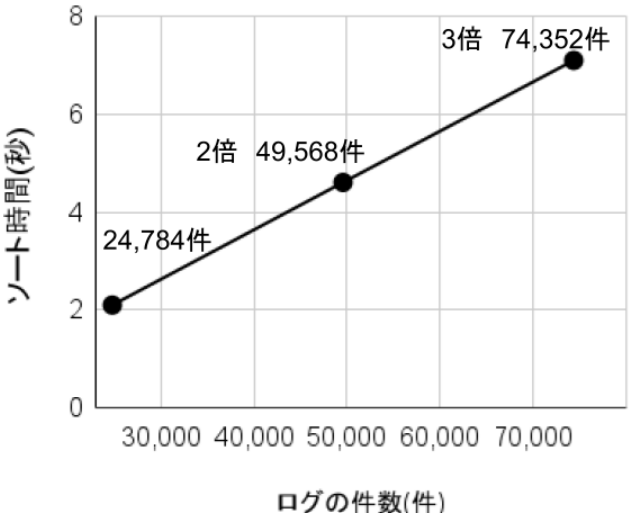

図 2　予備実験におけるログの件数とソートの時間

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

ある産業の売上量といった，特徴量を視覚的に色を用いて示す MAP を作成するアプリケーションを実装した研究がある [5]．563,000 以上のレコードがあるデータセット

[*3] https://ja.tak-cslab.org/
[*4] http://vkundeti.blogspot.com/2008/03/tech-algorithmic-details-of-unix-sort.html

を場所，業界タイプ，主要製品タイプ，売上高，従業員をキューブ化して分類している．この研究はデータセットをキューブ化してシステムが取り出しやすい形にしている．このキューブ化し，分類する考えはログ検索にも取り入れられる．

scatter-gather パターンを情報のアクセスツールとして捉えた研究がある [6]．この研究は scatter-gather におけるドキュメントクラスタリングは効果的な情報アクセスツールとなることを示している．ログ検索をする際に検索条件ごとのログファイルの塊を作ってアクセスする方法を用いれば scatter-gather パターンを使用したログ検索の高速化ができる．

3. 提案方式

提案方式

本提案は，複数のアプリケーションノードから集められたアクセスログをあらかじめアクセス時刻を基準にソートして，ステータスコードごとにログを分けて保存することでログ検索の応答時間を削減する．本提案は，ログを保存する段階で，ルートノードでソートしてステータスコードで分けた後に，リーフノードへ保存する．これによりログを検索するときは，ログをソートした状態でリーフノードに保存してあるため，リーフノードから集めた検索結果をルートノードでソートする時間を削減できる．また，ログをステータスコードごとに分けて保存しているため，検索の対象となるログを削減できる．ソート時間と検索対象を削減することは，検索の応答時間の削減に繋がる．アクセスログは，HTTP レスポンスステータスコードとアクセス時刻が入っている．また，システム管理者が発行する検索クエリは，HTTP レスポンスステータスコードとアクセス時刻順にソートする条件が付いているものとする．

図 3 はログの保存と検索のアーキテクチャを示している．はじめにログの保存について説明する．ログはユーザーのアクセスによってアプリケーションノードで生成される．生成されたログはアプリケーションノードからルートノードへ送信する．アプリケーションノードから送信されたログはルートノード内で結合し，HTTP レスポンスステータスコードと時刻でソートをされる．ログはソートによって HTTP レスポンスステータスコードごとに分けられる．分けられたログはアクセス時刻を基準にソートをした後ブロック化しリーフノードを通してディスクに保存される．HTTP レスポンスステータスコードごとに分ける理由はシステム管理者が検索クエリに条件を指定してログを検索をするためである．検索条件であるステータスコードを基準にソートすることで，検索の対象となるログをステータスコードごとに分けることができる．ステータスコードは障害対応のための異常か正常かの判断基準として利用できる．あらかじめステータスコードでログを分けることで本提案は検索条件に合わせたソートしたログを検索前に用意できる．ブロック化をする理由はログをひとつひとつソートするよりも短い時間でソートできるようにするためである．順番が保証されたログの塊をブロックと置き，ブロックを順番に結合することで，ソートされたログを出力できる．次に検索する場合を説明する．システム管理者はクライアントを通してルートノードに検索クエリを発行する．ルートノードは各リーフノードに検索を要求し返答を受け取る．このとき，ルートノードはリーフノードからの結果を結合するだけでソート処理をすることなく検索結果を出力できる．検索時間はソート処理が不要なため削減できる．

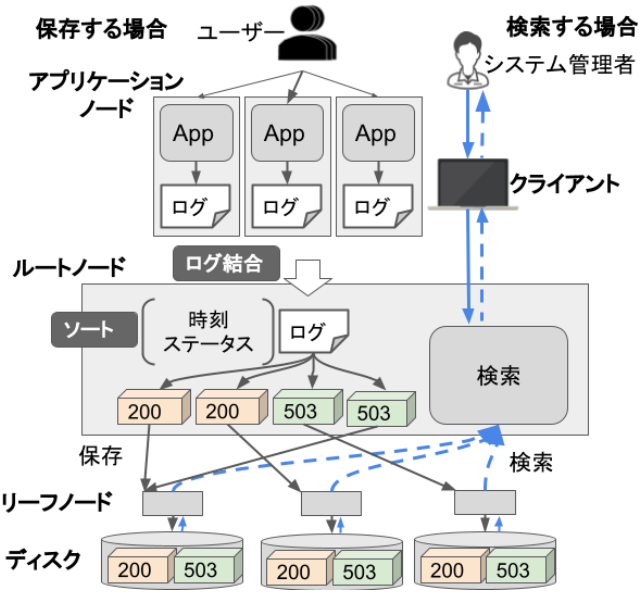

図 3 提案の全体図

図 4 はログの保存方法を示している．ログの保存は検索時にソート処理をしないためにあらかじめソートする．ルートノードで行う動作は 4 つに分けられる．

(1) ログ結合
(2) ソート
(3) 属性分割
(4) ブロック化

まずログ結合はアプリケーションノードから集められたログを結合する．次にソートは結合されたログを HTTP レスポンスステータスコードと時刻でソートをする．ソートによってログは HTTP レスポンスステータスが同じものでまとまり，同じ HTTP レスポンスステータスコードの中でも古いログから順番に並ぶ．また，属性分割は同じ HTTP レスポンスステータスコードのログをまとめて分割する．最後に，ブロック化は分割したログをアクセス時刻が時間と分まで一致するログごとに分ける．分ごとに分ける理由は，システム管理者が検索条件に時間帯を指定するとき，分単位で指定することを想定しているからである．ブロック化したログはリーフノードへ順番に振り分け，ディスク

に保存される．

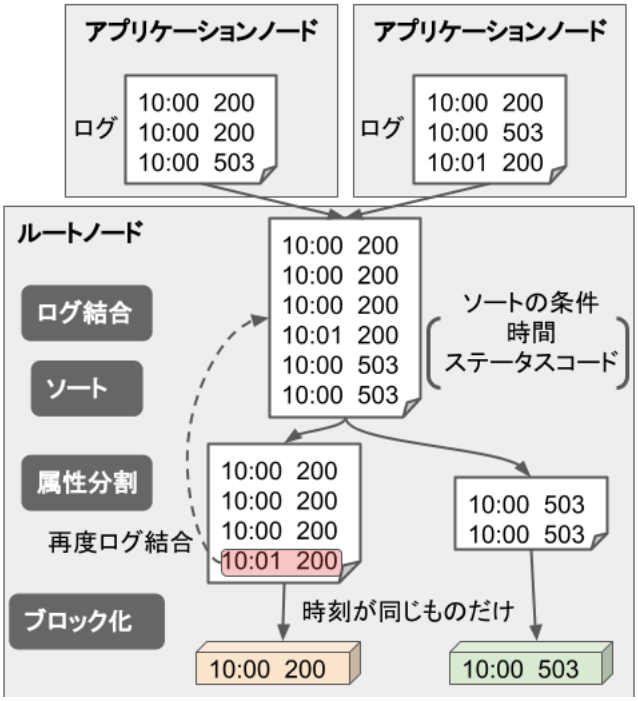

図 4　ログの保存方法

次に，システム管理者は HTTP レスポンスステータスコード 503 のログを検索するときの流れを例として説明する．HTTP レスポンスステータスコード 503 はサーバーがリクエストを処理する準備ができていないことを示している．サーバーがダウンしている障害のとき，システム管理者は HTTP レスポンスステータスコード 503 が出ていると判断して，いつから起きたことなのかを確認するためにログを検索する．図 5 はログの検索方法を示している．ログは保存するときにソートされているため，ルートノードはログを順番に結合するだけで古いログから順番に並んだ検索結果が得られる．検索の流れを以下に説明する．まずシステム管理者はクライアントを通してルートノードに検索クエリを送信する．ルートノードは各リーフノードに検索クエリを発行する．各リーフノードはルートノードから受け取った検索クエリに従い検索する．リーフノードの検索結果はルートノードに送信され結合される．結合した検索結果はクライアントに送信されて出力される．最後にシステム管理者はクライアントに出力された検索結果を確認する．

ユースケース・シナリオ

図 6 は WordPress における HTTP レスポンスステータスコード 503 が発生したときのユースケース・シナリオを示している．検索する目的はシステム管理者が障害の原因を特定することである．保存されるログは WordPress のアクセスログとする．(1)，(2) は Web サービス利用者の

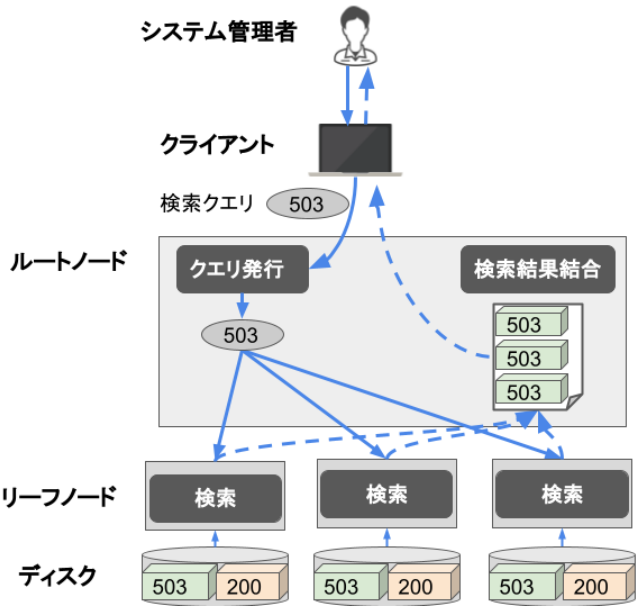

図 5　ログの検索方法

コメント投稿でエラーが発生することを示している．(3) は Web サービスがログをルートノードに送信することを示している．(4) は Web サービス利用者がカスタマサポートに問い合わせすることを示している．(5) はカスタマサポートがシステム管理者へ調査を依頼することを示している．(6) はシステム管理者が原因調査のため，ログをルートノードに検索することを示している．また，システム管理者は検索クエリの条件を HTTP レスポンスステータスコードとアクセス時刻を基準にソート，時間帯を分単位で指定するものとする．

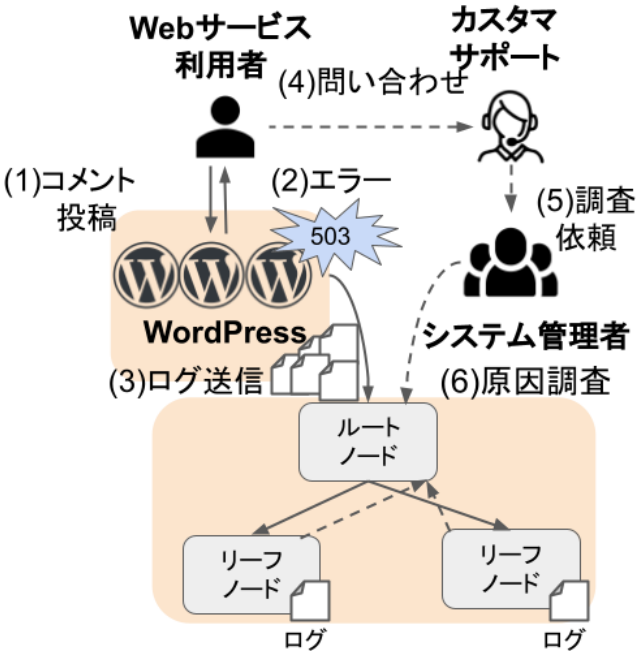

図 6　ユースケース・シナリオ

4. 実装と実験方法

実装

図 7 はログの保存方法を示している．実装はアプリケーションノード，ルートノード，リーフノードの 3 つに分けられる．作成するソフトウェアはログファイル生成，ブロック生成，ブロック配置の 3 つである．ログファイル生成はログファイルを生成するソフトウェアである．ルートノードへログファイルを送信する機能もある．ブロック生成はログファイルをブロック化するソフトウェアである．ブロックはログを一定容量にまとめたものである．ブロック配置はブロック配置先のリーフノード決定するソフトウェアである．

まずアプリケーションノードにあるログファイル生成がログファイルを生成する．生成されたログはアプリケーションノードからルートノードに送信する．送信されたログはブロック生成によってブロックにする．生成されたブロックはブロック配置によって複数あるリーフノードのディスクに振り分けられる．

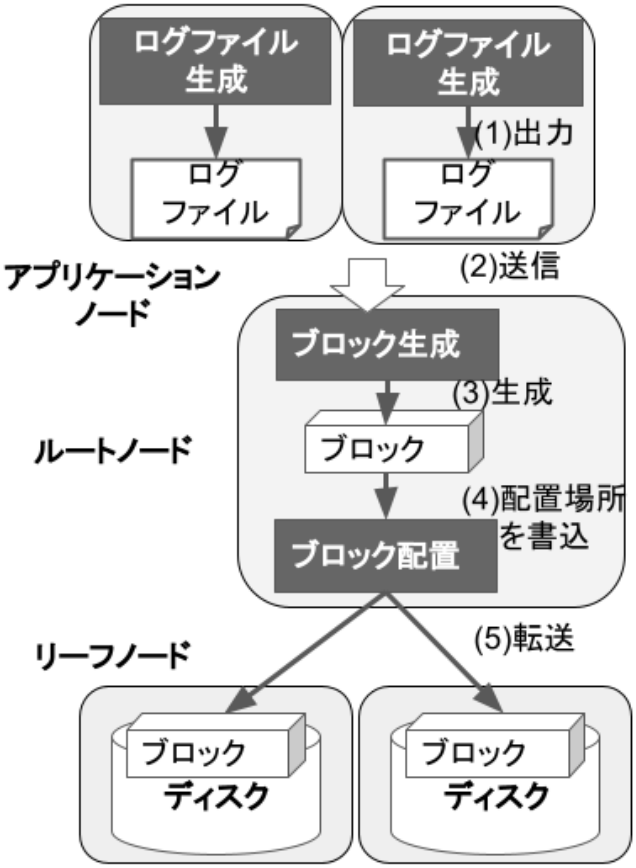

図 7 ログの保存

図 8 はログの検索方法を示している．使用するノードはルートノード，リーフノードである．作成するソフトウェアはクエリ発行とクエリ検索，検索結果結合の 3 つである．クエリ発行は分散先のリーフノードへ検索クエリを発行する．クエリ検索は発行された検索クエリを条件にディスクに保存されたブロックを検索する．検索結果結合は各リーフノードからの検索応答を結合する．本提案の検索ソフトウェアはシステム管理者，ルートノード，リーフノード，ルートノード，システム管理者の順番で検索する．はじめにシステム管理者はルートノードへ検索要求をする．検索要求はルートノード内のクエリ発行で受け取る．クエリ発行は分散先のリーフノードへ受け取った検索要求を検索クエリとして発行する．リーフノードに送られた検索クエリはクエリ検索で受け取る．クエリ検索はディスクに保存されたブロックをブロック単位で検索する．検索クエリに当てはまるブロックは検索応答としてルートノードに送信する．最後に各リーフノードから集められたブロックを結合してシステム管理者に検索応答を返す．

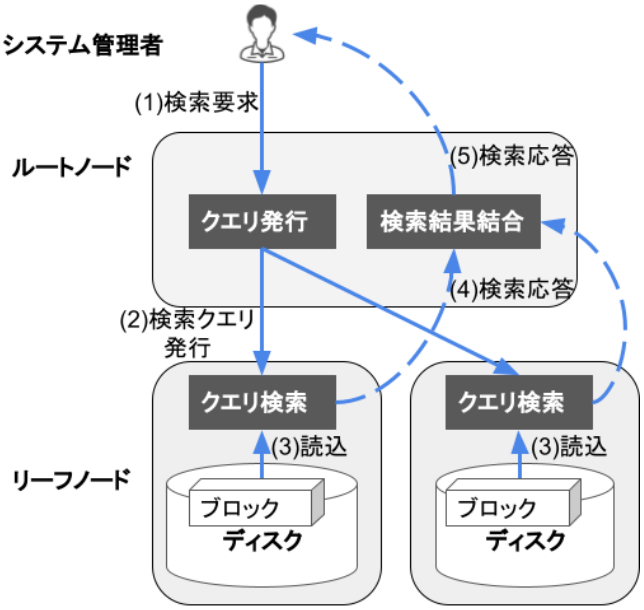

図 8 ログの検索

実験環境

図 9 は実験環境を示している．実験はハードウェア上にインストールしたハイパーバイザー (VMware ESXi) で行う．ハイパーバイザーに配置する仮想マシン (Ubuntu 20.04.2 LTS) はアプリケーションノード，ルートノード，リーフノードである．仮想マシンはネットワーク経由でストレージに接続する．全ての仮想マシンは同一のハードウェア性能 (CPU: 1[コア], RAM: 1[GB], Storage: 30[GB]) を持つ．各ノードの台数はルートノードを 1 台，アプリケーションノードとリーフノードをそれぞれ 5 台作成する．

5. 評価手法と分析手法

評価はログ検索の応答時間を比較する．評価手法は，生成したブロックの配置を変更や，検索クエリを変更するこ

図 9 実験環境図

とによって検索の応答時間を評価する．評価で使用するログは，実際にブログから取得したアクセスログを用いて生成する．

まずはじめに評価で使用するログの説明をする．評価に使用するログはアクセス傾向を考慮して生成する．アクセスログは CDSL ブログ（本研究室のブログサイト）[*3]の 13 日間で 24,784 件を使用する．このログを用いてアクセス傾向を維持したままログを生成する．

本提案は HTTP アクセスログを対象とする．アクセスログはユーザーがサーバーにアクセスすることで生成される．アクセスログはフォーマットに従ってログを出力する．例として，Nginx のログのフォーマットをソースコード 1 に，Nginx のログをソースコード 2 に示す．ソースコード 1 の$time_local に対応するのはソースコード 2 の 11/Mar/2021:06:25:47 +0900 である．$time_local はクライアントがサーバーにアクセスした時刻を示している．ソースコード 1 の$status に対応するのはソースコード 2 の 200 である．$status は HTTP レスポンスステータスコードを示している．

ソースコード 1 Nginx のログのフォーマット

```
log_format '$remote_addr -
    $remote_user [$time_local] "
    $request" $status $body_bytes_sent
    "$http_referer" "$http_user_agent"'
```

ログの生成方法を以下に示す．1 時間あたりの中央値の中でもっとも小さい値を**最小中央値**と定義する．例として，1 時間ごとのアクセス数の中央値が 40，30，70，90，40，30，20 とすると 20 が最小中央値と定義できる．2 週間のアクセスログを用いる予定である．

以下に手順を示す．

(1) 月，火，水，木，金 (以下，平日) と土，日 (以下，休日) に分けて (2) から (4) をする．
(2) アクセス数を時間帯（例：1 時間）ごとに集計し中央値をとる．

[*3] https://ja.tak-cslab.org/

ソースコード 2 Nginx のログの例

```
110.249.202.162 - - [11/Mar
    /2021:06:25:47 +0900] "GET /
    archives/650 HTTP/2.0" 200 6226 "-"
     "Mozilla/5.0 (Linux; Android 5.0)
    AppleWebKit/537.36 (KHTML, like
    Gecko) Mobile Safari/537.36 (
    compatible;Bytespider; https://
    zhanzhang.toutiao.com/)"
```

(3) (2) の中央値を最小中央値で割る．
(4) 目的のログの件数（1 分間に 1 万のリクエスト）に合わせるようにアクセス数を N 倍する．

以下にログを生成する手順の理由を示す．平日と休日に分ける理由は，平日と休日でアクセス傾向が異なるためである．まずはじめにアクセス数を一定時間（例：1 時間）ごとに集計して中央値をとる．次に各時間帯ごとの中央値を最小中央値で割る．(3) はアクセス傾向を得るためにする．1 時間あたりに最小中央値の何倍のアクセス数があるかでアクセス数を表現する．目的のログの件数（1 分間に 1 万のリクエスト）に合わせるようにアクセス数を倍にすることで，アクセス傾向を維持したログ件数の変移を再現できる．例として，1 日 2000 アクセスのブログサイトを基に 1 分間に 1 万のリクエストのログを生成する場合，7200 倍することでアクセス傾向を維持したログ件数の変移を再現することができる．最後に求めたログの傾向に従い，負荷テストツール JMeter を用いて対象となるアプリケーションノードにアクセスしてログを出力させる．出力されたログは実際のアクセス傾向に沿ったものになるため，実際に検索に使うログと類似したログになる．このログを評価実験で使用する．

検索の応答時間をブロックのリーフノードへの配置場所と検索クエリを変えて計測する．図 10 はログ検索の応答時間を示している．検索の応答時間はルートノードで検索要求をしてからルートノードで各リーフノードからの検索応答が完了するまでの時間とする．リーフノードは CPU:1[core]，RAM:1[GB] の VM を 5 台使用する．前提として，Web サービス利用者は障害が発生した時刻，エラー文，障害が発生した場所の URL，動作をカスタマーサポートに報告しているものとする．システム管理者の目的はいつから障害が発生したかを特定するためである．

ソースコード 3 は評価で使用する検索クエリの例を示している．HTTP レスポンスステータスコード 400 以上は異常が発生していることを示すので，検索クエリの status の条件は 400 以上と指定できる．次に，検索クエリの path は Web サービス利用者の報告より明らかになる．例えば，「/comment」をコメント投稿フォームの path とす

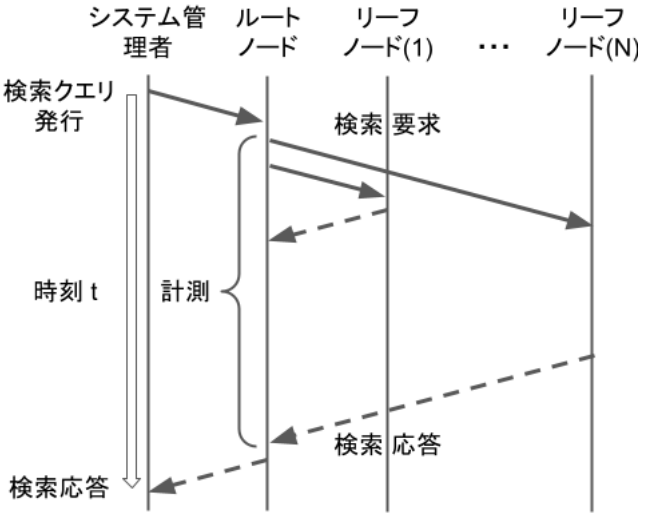

図 10 ログ検索の応答時間

ると，検索クエリの条件はコメント投稿時の障害の場合「/comment」と指定できる．最後に検索クエリの method は Web サービス利用者の報告がコメントを投稿したときとしたとき，POST と指定できる．status，path，method の 3 つの条件を指定して検索する．ログをブロックで保存しているため，ブロックの配置場所を変更や，検索クエリを変更することによって，検索をしたときに応答時間がどの様に増減するかを評価する．

ソースコード 3 検索クエリの例

```
date-time：11/22 0:00 ~ 11/24 9:16 AND
     status >= 400 AND path=/comment
    AND method = POST
```

6. 議論

scatter-gather を含む分散手法は落ちこぼれ問題がある．落ちこぼれ問題は分散先のノードの中でもっとも遅い返答時間が全体の返答時間となる問題である．この問題を解決するために，分散先の応答は同じ時間で返ってくる必要がある．ブロックはログの出力が同じものをまとめているため，同じ出力のブロックを均等にリーフノードへ振り分けることで分散先の応答時間を均等にする．したがって，ブロックの配置場所を決める手法を提案する．図 11 はブロックの配置先を決定する例である．前提として，リーフノードには通し番号が振り分けられているものとする．振り分け先の条件は (1) から (3) の順番で優先順位が低くなるものとする．

(1) 同じ属性のブロックがもっとも少ないリーフノードに振り分ける．
(2) ブロックの総数がもっとも少ないリーフノードに振り分ける．
(3) 通し番号の一番小さいリーフノードに振り分ける．

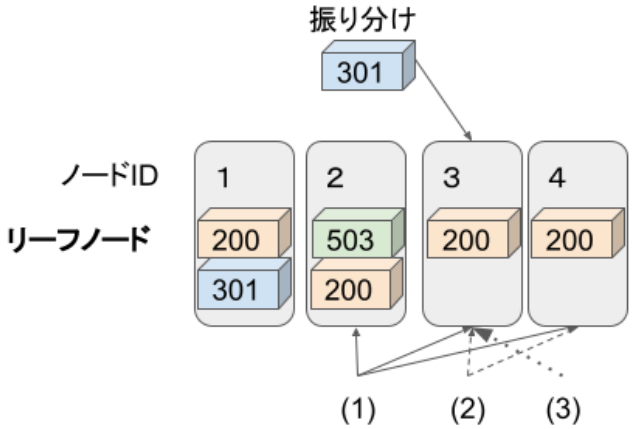

図 11 ログを振り分けるリーフノードの決定

(1) の条件より，ブロック 301 は 2，3，4 のリーフノードにはないため，振り分け先の候補はリーフノード 2，3，4 である．(2) の条件より，リーフノード 2 はブロックが 2 つだが，リーフノード 3，4 はブロックが 1 つであり，振り分け先の候補はリーフノード 3，4 である．(3) の条件より，候補の中でもっとも小さい番号は 3 であり，振り分け先はリーフノード 3 に決まる．

また本提案において，ログをリーフノードへ保存した後に，アプリケーションノードからログがルートノードへ送信される場合がある．このときリーフノードへ保存されたログは，ログがアクセス時刻を基準に並んでいない．アクセス時刻を基準に並んでいる状態でリーフノードへ保存しなければ，ログを検索するときに検索結果はアクセス時刻を基準に並んでいないことになる．したがって，ログがアクセス時刻を基準に並んでいる状態を維持する手法を提案する．ログがリーフノードへ保存した後にルートノードへ送信された場合，本提案と同じようにログを結合してソートして属性分割した後，ブロック化を行わず，リーフノードへ保存してあるブロックへログを送信する．後から来たログをリーフノードで結合し，ソートすることでブロック内のログがアクセス時刻を基準に並んでいる状態を維持する．

7. おわりに

課題は，検索結果をソートすることで検索結果の件数が増えるほどログ検索の応答時間が増加することである．本提案は，検索をするときに，ソート時間を削減したログの配置をする．本提案は検索条件として HTTP レスポンスステータスコードに着目してログを分割した．分割したログは時刻順にソートして保存することで，検索時にソートの時間を削減したログのブロックを作成した．ブロックはリーフノードへルートノードから振り分ける．検索時はブ

ロックをルートノードに送信し，結合することでソートの時間を削減したログ検索システムを実現した．評価は，仮想マシン上にログを配置して，ブロックの配置や検索結果を変えながら，応答時間を計測する．本提案は，障害が解消するまでの時間に含まれるログ検索の応答時間の削減により，経済損失を抑えることに貢献できる．

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] 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).

[3] Ali, S. K. and Sadik, S.: Web Services Monitoring, Analysis for Future Usage and Failure Prediction.

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

[5] Guo, D., Chen, J., MacEachren, A. and Liao, K.: A Visualization System for Space-Time and Multivariate Patterns (VIS-STAMP), *IEEE Transactions on Visualization and Computer Graphics*, Vol. 12, No. 6, pp. 1461–1474 (online), DOI: 10.1109/TVCG.2006.84 (2006).

[6] Cutting, D. R., Karger, D. R., Pedersen, J. O. and Tukey, J. W.: Scatter/Gather: A Cluster-Based Approach to Browsing Large Document Collections, *SIGIR Forum*, Vol. 51, No. 2, pp. 148 – 159 (online), DOI: 10.1145/3130348.3130362 (2017).