分散トレーシングのためのログ検索の高速化

小山 智之[1]　串田 高幸[1]

概要：分散トレーシングは，マイクロサービスアーキテクチャでログによる動作の解析を実現する．Scatter-Gather パターンは，トラフィックや期間が増えるにつれ件数が増加する大量のログを高速に検索する手法である．Scatter-Gather パターンの課題は，検索クエリによるログデータへのアクセス傾向と，Scatter-Gather パターンのリーフノードへのログデータの配置の不一致である．これにより，一部のリーフノードへディスクアクセスが偏り検索の応答時間が遅延する．本研究では，システム管理者による特定の条件による絞り込みや，特定のリクエスト ID をもつログの絞り込みに着目した．提案手法は，条件にもとづき同一の属性をもつログをクラスタリングし，時系列の昇順でソートする．その後，ソートしたデータを固定長のブロックに分割し，属性ごとにリーフノードへまとめて配置する．これにより，分散トレーシングでの検索クエリに着目した高速なログ検索を実現する．

1. はじめに

背景

分散トレーシングは，複数のコンポーネントから構成される分散システムにおいて，リクエスト単位で処理の追跡を実現する [1]．近年，分散トレーシングはマイクロサービスアーキテクチャにおいて採用されている．マイクロサービスアーキテクチャは，モノリシックアーキテクチャに比べコンポーネントが分離され複雑な構造をもつ．そのため，リクエストの追跡や分析，ボトルネックの発見が難しい [2]．分散トレーシングはこれら課題をリクエストに識別子を付与することで実現する．

分散トレーシングに用いられるオープンソースソフトウェアの代表例は，Istio である．Istio は Kubernetes 上で Envoy Proxy を使い分散トレーシングを実現する．Istio のアーキテクチャを図 1 に示す．図では，サービス A コンテナとサービス B コンテナが Istio 上にデプロイされている．サービス A コンテナからサービス B コンテナへリクエストを送信するとき，サービス A の Pod に配置された Envoy は，送信 ログを出力する．このログには，リクエストに含まれる ID が含まれる．同様に，サービス B の Pod に配置された Envoy は，受け取ったリクエストに含まれる ID を受信ログへ出力する．

Envoy が出力する送信ログの形式をソースコード 1 に示す．ログにおいて ID は 0cde9d55-b810-4fc4-9489-4b3ca1694905(以降，0cde9）であり，呼び出し先サービス名は reviews:9080 である．Envoy が出力する受信ログの形式をソースコード 2 に示す．ログにおいて ID は 0cde9 である．また，呼び出し先サービス名は，reviews:9080 である．Pod ごとに配置された Envoy が出力するログに含まれる ID(例: 0cde9) を絞り込むことで，システム管理者はマイクロサービスアーキテクチャでサービスごとにリクエストが処理される流れを追跡できる．

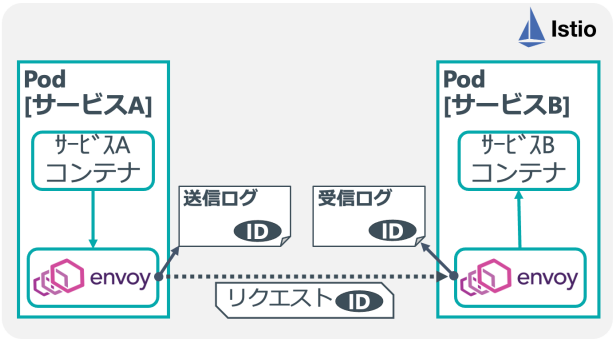

図 1　Istio のアーキテクチャ

Scatter-Gather パターンは，大規模なデータの高速な処理を実現する分散処理アーキテクチャである．このアーキテクチャは，クラウド上のデータセンタにおける Hadoop や BigQuery で頻繁に利用される [3]．利点は，システム利用者からの視点での透過性である．このアーキテクチャは，単一のノードへのアクセスにみせることで，利用者が意識せずとも大規模なデータの処理を実現する．図 2 に Scatter-Gather パターンのアーキテクチャを示す．Scatter-Gather

[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

ソースコード 1　ログの例（呼び出し元）

```
[2021-07-20T04:42:14.654Z] "GET /reviews/0
    HTTP/1.1" 200 - via_upstream - "-" 0
    379 16 16 "-" "Mozilla/5.0 (Macintosh;
    Intel Mac OS X 10_15_7) AppleWebKit
    /537.36 (KHTML, like Gecko) Chrome
    /91.0.4472.164 Safari/537.36"
    "0cde9d55-b810-4fc4-9489-4b3ca1694905"
    "reviews:9080" "172.17.0.8:9080"
    outbound|9080||reviews.default.svc.
    cluster.local 172.17.0.11:41668
    10.101.220.149:9080 172.17.0.11:33134 -
    default
```

ソースコード 2　ログの例（呼び出し先）

```
[2021-07-20T04:42:14.655Z] "GET /reviews/0
    HTTP/1.1" 200 - via_upstream - "-" 0
    379 16 16 "-" "Mozilla/5.0 (Macintosh;
    Intel Mac OS X 10_15_7) AppleWebKit
    /537.36 (KHTML, like Gecko) Chrome
    /91.0.4472.164 Safari/537.36"
    "0cde9d55-b810-4fc4-9489-4b3ca1694905"
    "reviews:9080" "172.17.0.8:9080"
    inbound|9080|| 127.0.0.6:52005
    172.17.0.8:9080 172.17.0.11:41668
    outbound_.9080_._.reviews.default.svc.
    cluster.local default
```

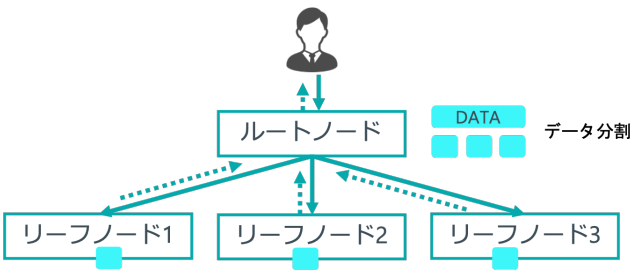

図 2　Scatter-Gather パターンのアーキテクチャ

パターンにおいてノードは，ルートノードとリーフノードの 2 種類から構成される．ルートノードの役割を次に示す．

- ユーザからのクエリの受け取り
- リーフノードへの処理要求を配布 (scatter)
- リーフノードからの処理結果の集約 (gather)
- 処理結果のユーザへの返答

リーフノードには，処理対象のデータが事前に配置される．リーフノードは，ルートノードからの処理要求を受け取り，要求をもとに処理を実行する．また，処理の結果をルートノードへ送信する．

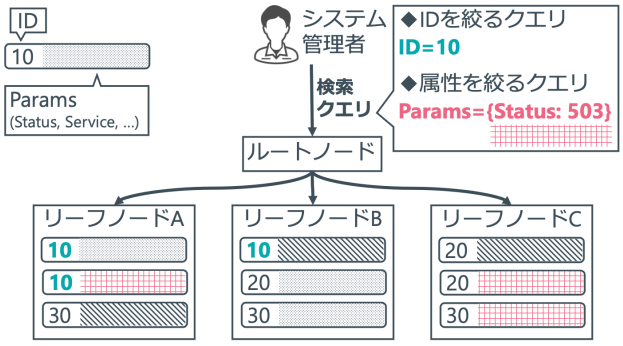

図 3　Scatter-Gather パターンの課題

課題

課題はシステム管理者により発行される検索クエリによる検索の応答時間の遅延である．Scatter-Gather パターンを採用したログサーバは，検索クエリとデータ配置の組み合わせにより応答時間が遅延する．これは，クエリごとにリーフノード上にあるログデータへのアクセスパターンが異なるためである [4].

図 2 は，Scatter-Gather パターンを採用したログサーバにおける課題をあらわす．検索対象のログは，分散トレーシングのために使われるアクセスログである．これらはリーフノードへ配置される．ログには ID と Params が含まれる．例えば，図の左上にあるログは ID が 10 である．また，Params はログに含まれる属性 (Status, Service) を表し，属性の組み合わせが同一である場合に色付けを一致させる．ID は，大量のログの中からサンプリングした追跡のために使う．Params は，大量のログから必要な条件を満たすログを絞り込むときに使う．ルートノードはリーフノードに配置されたデータとリーフノード名が対応付けを把握している．ルートノードとリーフノード間はネットワークにより接続されている．各リーフノードのハードウェアは，同一の性能とする．

システム管理者はルートノードへログを検索するため，検索クエリを発行する．ルートノードはログとリーフノードの対応付けをもとにリーフノードへ検索要求を発行する．リーフノードは，ルートノードから受け取った検索要求にもとづきノード上のログデータを検索する．リーフノードでの検索時間は，リーフノードに配置された検索対象のログ件数により決まる．これは検索対象のログ件数が増えるほど，リーフノードでのディスク I/O が増加するためである．検索の応答時間を高速化には，検索処理によるデータアクセスがリーフノードへ均等に分配される必要がある．一方，検索クエリの種類によりデータへのアクセスパターンは異なる．例えば，図 3 の ID を絞る検索クエリでは，リーフノード A のログ 2 件とリーフノード B のログ 1 件への検索処理が発生する．属性を絞る検索クエリでは，リーフノード A のログ 1 件とリーフノード C のログ

2 件への検索処理が発生する．リーフノードでのアクセスパターンの偏りは，検索処理にかかる時間のばらつきを生む．これによりシステ全体での検索の応答時間の遅延を生む．分散トレーシングで発行される検索クエリは，ID や属性によりアクセスパターンが異なるため単純な属性によるクラスタリングでは検索の応答時間の高速化が不十分である．本研究では，分散トレーシングにおける 2 種類の検索クエリ（ID による検索，属性による検索）を対象に検索の高速化を行う．

各章の概要

2 章では関連する既存研究を紹介する．3 章では提案手法を説明する．4 章では実装方法および実験方法を述べる．5 章は，評価および分析の手法を説明する．6 章では提案手法の議論をする．最後に，全体を簡潔にまとめ，貢献を明らかにする．

2. 関連研究

Hadoop を対象としたレプリケーションに用いる指標とジョブ実行時間の関係を示した研究がある [5]．この研究は，レプリケーションに用いる指標にディスク空き容量やネットワークスループット，コピー時間を使用した．指標の種類を 3 つから 9 つへ増やすことで，ジョブ実行時間の 18-20%を削減した．ジョブ実行時間の削減は，ログ検索においても有用である．一方，この手法では HDFS 上のデータ内容へ配慮がないため，ログのアクセス傾向にもとづくジョブの実行時間の削減には改善の余地がある．

CDRM は，クラウド上のストレージクラスタにおいてコストに配慮した動的なレプリケーション管理を行う研究である [6]．研究では，可用性とレプリカ数の関係に着目し，要求された可用性を満たす最小のレプリカ数をノードのキャパシティとブロッキング確率をもとに求めた．この提案により，可用性とアクセスレイテンシを向上した．提案は，アクセスレイテンシを向上したが，データ同士の依存関係は考慮されていない．分散トレーシングではデータ間に依存関係へ配慮する必要があり，この提案ではアクセスレイテンシの改善が十分でない．

SQL と NoSQL を組み合わせた高速なデータベースを提案した研究がある [7]．提案では，Elasticsearch と MySQL をミラーリングにより組み合わせ，データの追加と更新を MySQL で行いデータの取得を Elasticsearch で行った．これにより，SQL と NoSQL の相互の強みを活かした高速な検索を実現した．提案システムのもつ性質（トランザクション間の独立性，単一書き込み）は，ログへの利用に適した性質である．一方，データの配置はアクセス特性へ配慮されず高速化に課題がある．

AptStore は，ストレージコストの削減と I/O スループットの改善を行うため，動的なデータの管理システムを

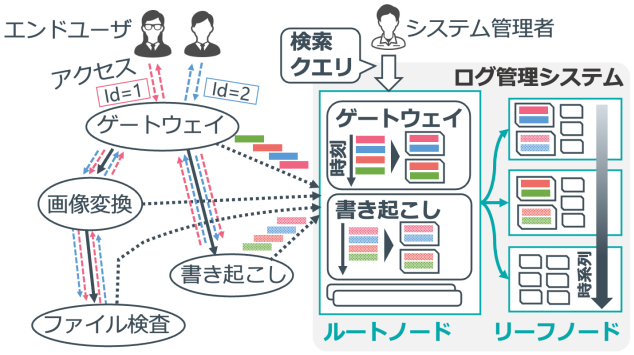

図 4 提案の概要

Hadoop 上に構築した [8]．データへのアクセスがある可能性を算出するアルゴリズムの導入により，スループットの向上とストレージの消費量を削減した．この手法では，アクセスの可能性を算出するために利用した指標がアクセス回数である．過去のアクセス傾向が将来のアクセス傾向と一致する場合はこの手法は有用である．しかし，分散トレーシングにおいてアクセス傾向の一致は僅かであり，この手法では高速化の性能が十分に活用できない．

データ配置により InfiniBand での通信性能を向上する研究がある [9]．研究では，ファイルシステムのページサイズを削減することでメモリ登録によるオーバーヘッドを削減した．この研究ではデータの配置とアクセス傾向には配慮がされず，性能向上に課題がある．

CoHadoop は，Hadoop における柔軟なデータの配置と利用を実現した [10]．この提案では，データノードにおけるディスクの空き容量やファイルサイズ，ファイルの作成順序に着目しデータの配置を決定した．データへのアクセス傾向はデータの内容により決まるため，この手法では検索の性能改善には十分でない．

3. 提案方式

提案の概要を図 4 に示す．図の左側は，マイクロサービスアーキテクチャを採用した Web サービスをあらわす．図の右側は，Web サービスからのログによる分散トレーシングを行うログ管理システムをあわらす．エンドユーザは Web サービスの利用者をあわらす．システム管理者は Web サービスの運用に従事する技術者である．それぞれのサービス（ゲートウェイ，画像変換，書き起こし，ファイル検査）は，実線で接続されたサービスをネットワーク経由で呼び出す．このとき，呼び出し元と呼び出し先の双方で呼び出しに伴いトレースログが出力される．呼び出し元サービスのトレースログは，ソースコード 1 の形式をもつ．呼び出し先サービスのトレースログは，ソースコード 2 の形式をもつ．図 4 の右側は，Scatter-Gather パターンを採用したログ管理システムである．提案方式では，分散トレーシングのログがもつアクセスパターンの特徴に着目

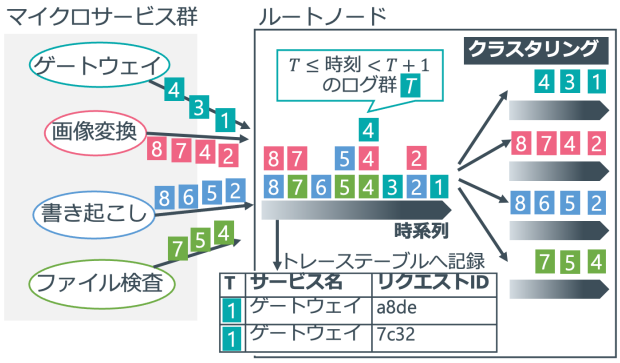

図 5　クラスタリングの方法

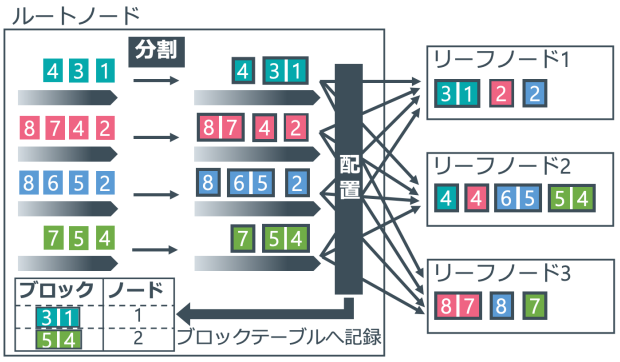

図 6　分割と配置の方法

する．具体的には，パターンのもつ特徴にもとづきログのクラスタリングと分割を行う．さらに，分割したログをアクセスパターン（時系列）にもとづきリーフノードへ配置する．これにより分散トレーシングにおける検索の応答時間を高速化する．以降では，クラスタリングと分割，リーフノードへの配置を説明する．

クラスタリング

それぞれのマイクロサービス (例:画像変換) は，トレースログをログ管理システムのルートノードへ送信する．ルートノードでは，マイクロサービスから送られたログをサービスごとにクラスタリングする．これは，検索する際のアクセス対象となるログデータ量を削減するためである．クラスタリングの方法を図 5 に示す．図の左側にはそれぞれのマイクロサービス（ゲートウェイサービス，画像変換サービス，書き起こしサービス，ファイル検査サービス）がある．各サービスからは，一定時間ごと（例:1 分）にバッファリングされたログがルートノードへ転送される．例えば，ゲートウェイサービスは時刻 1,3,4 でのバッファリングしたログをルートノードに転送する．ルートノードでは受信したログの時刻 (T) とサービス名，リクエスト ID をトーレステーブルに記録する．これは，リクエスト ID でログを検索する場合の検索対象を削減するためである．図 5 の表では，時刻 1 のときにゲートウェイサービスから出力されたログ (リクエスト ID: a8de) をあらわす．ルートノードは，受信したログに含まれるサービス名をもとにログをクラスタリングする．例えば，ゲートウェイサービスから受信したログは，時刻 1,3,4 のログをゲートウェイサービスとしてクラスタリングされる．

分割・リーフノードへの配置

ログの分割とリーフノードへの配置を図 6 に示す．ルートノードはそれぞれのクラスタリングしたログを期間で分割し，ブロックを作成する．例えば，ゲートウェイサービスから出力されたログ 1,3,4 は，ブロック 1,3 とブロック 4 に分割される．ブロックの期間 r は，エンドユーザからのリクエストの処理を終えるまでの時間とする．これはリクエスト ID で絞り込む検索クエリによりアクセスされるログの件数から算出した．図は，$r = 3$ の場合をあらわす．すなわち，時刻 1 から 3 までのログから 1 つのブロックを作成する．

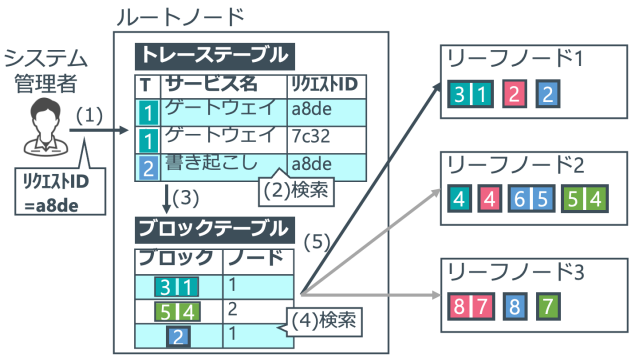

図 7　検索の動作

ルートノードは，作成したブロックをリーフノードへ配置する．それぞれのブロックは，ブロック ID をもつ．時刻 t におけるログを含むブロック ID は $floor(t/r)$ で示される．リーフノードの総数が L のとき，それぞれのリーフノードは 1 から L までのノード ID をもつ．ブロック ID が i のログを配置するリーフノードの ID は，$mod(i, L)+1$ である．ブロックテーブルは，ブロックごとに配置されたリーフノードを記録する．これは，ブロック ID からブロックの配置されたリーフノード ID を特定するためである．図 6 では，時刻 3,1 を含むブロックがノード 1 に配置されていることをあらわす．

ログ検索

提案手法における検索の動作を図 7 に示す．以下は，システム管理者により発行された検索クエリが処理される手順を述べる．

(1) システム管理者は，ルートノードへ検索クエリ（例: リクエスト ID=a8de) を発行する．

(2) ルートノードは，トレーステーブルからリクエスト ID に対応する時刻とサービス名の組み合わせ（例: 1,

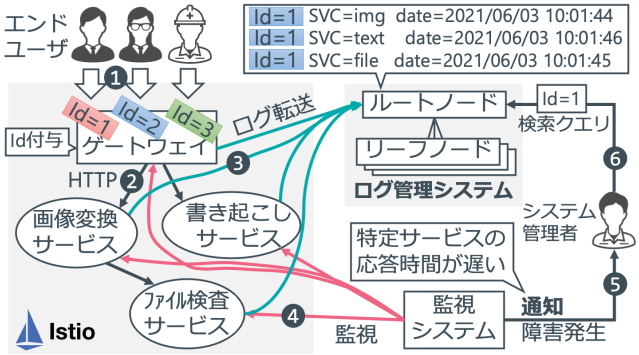

図 8　マイクロサービスアーキテクチャにおける障害対応

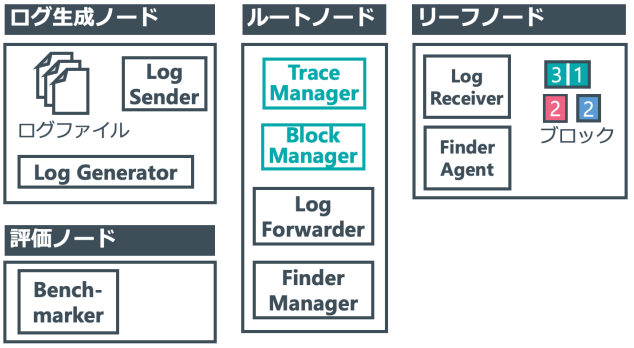

図 9　実装の概要

ゲートウェイ) を探す.

(3) トーレステーブルは，時刻とサービス名の組み合わせをブロックテーブルへ渡す.

(4) ルートノードは，ブロックテーブルから時刻とサービス名の組み合わせを含むブロック (例: 3,1) を探し，対応するノード (例: 1) を探す.

(5) ルートノードは，リーフノードへブロックテーブルから求めたノードへブロックの検索処理を発行する.

ユースケース・シナリオ

図 8 は，マイクロサービスアーキテクチャを採用した Web サービスにおける障害対応を示す．Istio 上に API Gateway と各マイクロサービス（画像変換サービス，書き起こしサービス，ファイル検査サービス）を配置する．各サービスのトレースログ（呼び出し元，呼び出し先）は，ログサーバへ集約され保管される．監視システムは，それぞれのマイクロサービスへアクセスすることで，サービスの応答時間を監視する．ここでは，監視システムが画像置換サービスにおける応答時間の遅延による障害を検出したときを考える．監視システムは，システム管理者へ障害発生の通知を送る．システム管理者は，障害の原因を突き止めるためにログサーバへ検索クエリを発行する．

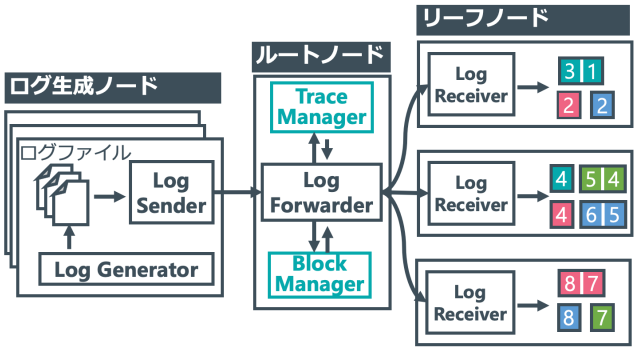

図 10　ログ保存の実装

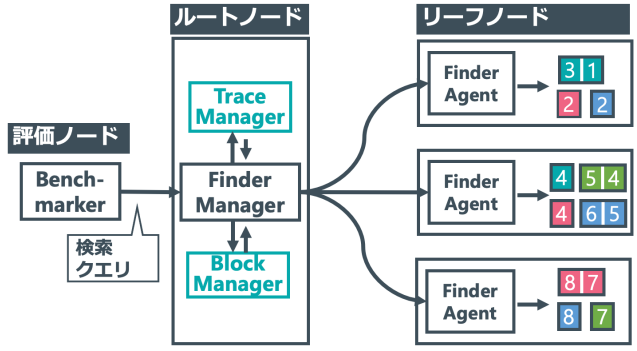

図 11　ログ検索の実装

4.　実装と実験方法

実装

図 9 に実装の概要を示す．実装でのノードは，ログ生成ノードと評価ノード，ルートノードとリーフノードの 4 種類から構成される．ログ生成ノードには，Log Generator と生成されたログ，Log Sender が配置される．評価ノードには Benchmarker を配置する．ルートノードには，Trace Manager と Log Forwarder，Block Manager, Finder Manager が配置される．リーフノードは，Log Receiver と Finder Agent，ログファイルから生成したブロックが配置される．

ログの保存時のアーキテクチャを図 10 に示す．ログ生成ノードの Log Generator は事前に定義した条件に基づきログファイルを生成する．Log Sender は，マシンのローカルに配置されたログをルートノードの Log Forwarder へ転送する．Log Sender ではログのバッファリングを行う．Log Forwarder は，Log Sender からログを受信し，ログエントリに含まれる時刻とサービス名，リクエスト ID を Trace Manager へ送る．Trace Manager は，受け取った時刻とサービス名，リクエスト ID をトレーステーブルに保存する．Log Forwarder は，サービスごとにログをクラスタリングする．クラスタリングしたログは，期間で分割されアルゴリズムによりノードが割り当てられる．Block Manager は，ブロックと割当リーフノードの組み合わせ

を記録する．ブロックは Log Forwarder により割当リーフノードの Log Receiver へ転送される．リーフノード上の Log Receiver は受信したログをローカルボリュームに保存する．Log Sender と Log Forwarder，Log Receiver は Fluentd を利用する．Trace Manager と Block Manager を新たに開発する．

ログの検索時のアーキテクチャを図 11 に示す．評価ノードの Benchmarker は定義された検索クエリをルートノードの Finder Manager へ送信する．Finder Manager は Trace Manager と Block Manager へ問い合わせを行い，検索クエリに対応するブロックとリーフノードの対応関係を解決する．Finder Manager は，リーフノードごとの Finder Agent へ検索要求を並列で送信する．Finder Agent は検索要求に対応するログをブロックから検索し，検索結果をルートノードへ返答する．ルートノードは全てのリーフノードの検索結果を結合し，評価ノードへ返答する．

トレーステーブルの設計: 提案手法は，リクエスト ID による検索を高速に行うためトレーステーブルによるリクエスト ID とブロックの対応付けを行った．こうしたデータアクセスの高速化戦略は，リレーショナル・データベース (以降，RDB) におけるインデックスの構築と類似する．提案ではトレーステーブルの実装にハッシュテーブルを使用した．ハッシュテーブルでは検索後のディスクアクセス回数は，$O(1)$ である．

RDB で代表的な B-tree によるインデックス構造は，データを木構造により分類することでアクセスを高速化している [11]．B-tree において 1 回あたりのデータ検索で発生するディスクへのアクセス回数は，$O(\log N)$ である．これは，データ量が増えた場合もディスクアクセスへの回数の増加は緩やかである．ビットマップインデックスは，カーディナリティの小さいデータ群において検索が高速である [12]．特に性別に代表されるデータの種類（カーディナリティ）が少ない場合に検索がはやい．

本論文では，リクエスト ID を条件としたログの検索を想定した．Istio に内包された Envoy から出力されるリクエスト ID は UUID 形式である．UUID はトランザクション単位で生成されるため重複が極めて少なく，カーディナリティが大きいデータである．したがって，ビットマップインデックスは適さない．また，ハッシュテーブルは B-tree に比べてアクセスが少ない．したがって，トレーストレーステーブルはハッシュテーブルによる実装が適する．

リクエスト ID 以外の属性による検索を高速化するには，データのカーディナリティに配慮したインデックスの構築を考慮すべきである．これにはデータのカーディナリティを「ユニークなデータ件数/全データ件数」で求め，しきい値によりインデックス構造を選択する．

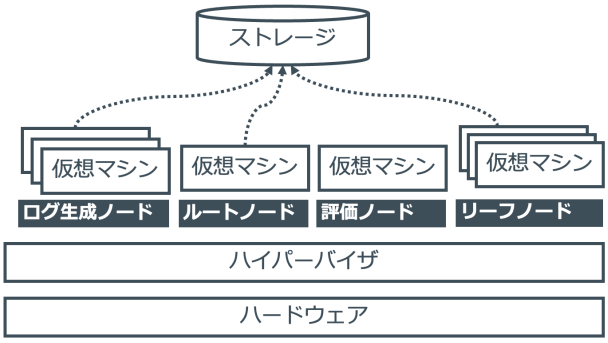

図 12 実験の環境

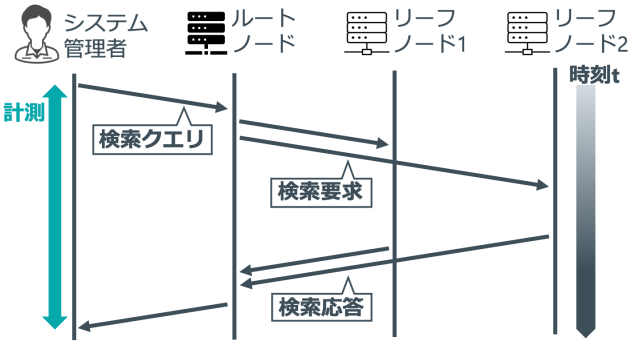

図 13 応答時間の計測方法

実験環境

実験環境を図 12 に示す．ハードウェア上にハイパーバイザ (VMware ESXi) をインストールし，仮想マシンを配置する．ログ生成ノードやルートノード，リーフノードや評価ノードはそれぞれ仮想マシンで作成する．仮想マシンはネットワーク経由でストレージに接続される．すべての仮想マシンは，同一のハードウェア性能 (CPU: 1[コア], RAM: 1[GB], Storage: 30[GB]) とする．各ノードの台数は，ログ生成ノードを 5 台，ルートノードと評価ノードを 1 台，リーフノードを 5 台とする．

5. 評価手法と分析手法

検索クエリに対応する検索結果が得られるまでの時間（応答時間）を計測する．図 13 に応答時間の計測方法を示す．応答時間は，システム管理者がルートノードへ検索クエリを発行してから，検索結果が得られるまでの時間とする．評価では，以下の条件設定を変更し検索の応答時間を測定する．

- **ブロックのサイズ:** ブロックに含めるログの期間 r(単位: 秒) を範囲 $[1, 300]$ で変更し，同一の検索クエリでの応答時間を計測する．ブロックのサイズが検索の応答時間に与える影響を分析する．
- **検索クエリ:** ルートノードへ発行する検索クエリの条件を 2 種類の中で変更する．条件 1 は，特定の URL エンドポイント (Method, Path) における障害 (ステー

ソースコード 3　検索クエリの例

```
date_from=2021-07-01 date_to=2021-07-08
rid=0cde9d55-b810-4fc4-9489-4b3ca1694905
```

タスコード=50x) の絞り込みである．条件 2 は，特定の URL エンドポイント (Method, Path) における障害のリクエスト ID によるサンプリングである．検索クエリの条件と検索の応答時間を比較する．

- **リーフノードの台数:** リーフノードの台数を範囲 [3,13] で変更し検索の応答時間を測定する．スケーラビリティを評価するため，リーフノード数が増えたときの検索の応答時間を比較する．
- **ログの件数:** ログ生成ノードからルートノードへ送信するログの件数を範囲 [1000,10000] で変更し，検索の応答時間を測定する．ログ件数の増加と検索の応答時間を比較する．

ログの生成: 評価で利用するログデータは，独自に作成したテクニカルレポート公開サイトのトレースログである．ログ形式は，ソースコード 1 とソースコード 2 である．ログ件数が少ないことから，既存ログをもとに Log Generator でトレースログを生成する．ログの生成ではトラフィックの性質を考慮する．1 時間あたりで生成するログの件数を過去のトラフィック傾向と乱数を組み合わせて決定する．具体的には，トラフィックを 1 時間ごとのアクセス数で集計し，各アクセス数の 0〜10%の範囲で乱数を生成する．これにより 1 時間あたりに生成するログの件数を決定する．ログの内容は過去のトレースログを属性ごとに分割し，属性をランダムに選択して組み合わせる．

検索クエリ: トレースログにより障害の原因をもつサービスを特定するクエリを用意する．検索クエリの例をソースコード 3 に示す．このクエリは，リクエスト ID を絞り込む検索で使われる．クエリを発行するシステム管理者は，日時やステータスコード (例: 503) により絞り込むクエリを発行する．次に個々のログをリクエスト ID によりサンプリングし，処理をトレースする．5 種類のこうしたクエリを用意し Benchmarker から発行する．

6. 議論

提案手法ではブロックの期間 r は固定長としている．ブロックの期間はブロックのファイルサイズに対応する．ブロックの期間が増加するほど，ブロックのファイルサイズは増加する．同時に読み書き可能なファイルサイズは，ディスクのもつ I/O 性能により異なる．したがって，ブロックのサイズはディスクの I/O 性能による算出が必要である．これには，iostat でディスクの読み書き性能 (IOPS) を測定し，ブロックの期間を決定する．

MySQL(InnoDB) や PostgreSQL では，ブロックサイズの初期値はそれぞれ 16[KB], 8[KB] である．これはファイルシステムにおけるキャッシュのページサイズが 4[KB] の倍数であるためだと考える．例えば，16[KB] のブロックにソースコード 2 のログを含める場合を考える．このとき，400 文字を 400[B] と換算すると 40 件 (16000/400 = 40) が含められる．ブロックサイズの初期値 (4KB, 8KB, 16KB, 32KB, 64KB) で変更し，検索の応答時間を計測する．これにより適したブロックサイズを決定する．

7. おわりに

課題は，Scatter-Gather パターンを採用したログ管理システムにおいて，分散トレーシングを目的としたログ検索での応答時間の遅延である．提案では，分散トレーシングにおいてシステム管理者が発行する 2 種類のクエリ（ステータスコード，リクエスト ID）を高速化する．提案手法は，検索条件にもとづき同一の属性（期間，ステータスコード，サービス名）をもつログをクラスタリングし，時系列の昇順でソートした．その後，ソートしたデータを固定長のブロックに分割し，属性ごとにリーフノードへまとめて配置した．これにより，分散トレーシングでの検索クエリに着目した高速なログ検索を実現した．評価では，仮想マシン上にログを配置し条件（ブロックのサイズ，検索クエリ，リーフノードの台数，ログの件数）を変えながら検索の応答時間を計測する．提案により分散トレーシングによるシステム管理者の障害対応にかかる時間を削減し，システムにおける信頼性の向上が期待できる．

参考文献

[1] Shkuro, Y.: *Mastering Distributed Tracing: Analyzing performance in microservices and complex systems*, Packt Publishing Ltd (2019).
[2] Mengistu, D. M.: Distributed Microservice Tracing Systems: Open-source tracing implementation for distributed Microservices build in Spring framework (2020).
[3] Alvarez, C., He, Z., Alonso, G. and Singla, A.: Specializing the network for scatter-gather workloads, *Proceedings of the 11th ACM Symposium on Cloud Computing*, pp. 267–280 (2020).
[4] Dan, A., Philip, S. Y. and Chung, J.-Y.: Characterization of database access pattern for analytic prediction of buffer hit probability, *The VLDB Journal*, Vol. 4, No. 1, pp. 127–154 (1995).
[5] Ciritoglu, H. E., Batista de Almeida, L., Cunha de Almeida, E., Buda, T. S., Murphy, J. and Thorpe, C.: Investigation of replication factor for performance enhancement in the hadoop distributed file system, *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pp. 135–140 (2018).
[6] Wei, Q., Veeravalli, B., Gong, B., Zeng, L. and Feng, D.: CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster, *2010 IEEE international conference on cluster computing*, IEEE, pp. 188–196 (2010).
[7] Taware, U. and Shaikh, N.: Heterogeneous database system for faster data querying using elasticsearch,

2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), IEEE, pp. 1–4 (2018).

[8] Krish, K., Khasymski, A., Butt, A. R., Tiwari, S. and Bhandarkar, M.: Aptstore: Dynamic storage management for hadoop, *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, Vol. 1, IEEE, pp. 33–41 (2013).

[9] Rex, R., Mietke, F., Rehm, W., Raisch, C. and Nguyen, H.-N.: Improving communication performance on InfiniBand by using efficient data placement strategies, *2006 IEEE International Conference on Cluster Computing*, IEEE, pp. 1–7 (2006).

[10] Eltabakh, M. Y., Tian, Y., Özcan, F., Gemulla, R., Krettek, A. and McPherson, J.: CoHadoop: flexible data placement and its exploitation in Hadoop, *Proceedings of the VLDB Endowment*, Vol. 4, No. 9, pp. 575–585 (2011).

[11] Comer, D.: Ubiquitous B-tree, *ACM Computing Surveys (CSUR)*, Vol. 11, No. 2, pp. 121–137 (1979).

[12] Wu, K., Ahern, S., Bethel, E. W., Chen, J., Childs, H., Cormier-Michel, E., Geddes, C., Gu, J., Hagen, H., Hamann, B. et al.: FastBit: interactively searching massive data, *Journal of Physics: Conference Series*, Vol. 180, No. 1, IOP Publishing, p. 012053 (2009).

