Vol.2020-DPS-185 No.17
2020/12/22

検索性能に配慮した複製による分散ログ管理

小山 智之[1]　串田 高幸[1]

概要：ソフトウェアから出力されるログは，IT システムにおける障害の原因追求に利用される．システム規模が拡大するにつれ，ログの総量は線形に増加する．障害が発生した際には，大量のログから有用なログのみを高速に発見することが求められる．本研究では，システム構成情報と検索シナリオを統合しログ配置ルールを作成することで，検索時の応答速度の高速化を図る．管理者はシステムで動作するノードの役割ごとにシステム構成情報を記述する．検索シナリオはあらかじめ用意した利用頻度の高い条件を使用する．検証ではコンテナ上で既存システムから収集したログを拡張したログメッセージを対象に配置ルールに基づく配置を行った．検索時の性能を測定するため検索クエリを発行し，その応答時間をランダムにログメッセージを配置した手法と比較した．その結果，提案手法はランダム手法に比べ最大で 10.9 倍高速になった．

1. はじめに

1.1 背景

ログはソフトウェアから出力されるメッセージである．このメッセージは，システムの正常な動作の過程を表したり，システムの障害やその兆候を表したりする．メッセージは OS やミドルウェア，アプリケーションから個別に独自のテキスト形式で出力され，テキストファイルとして保管される．ログのユースケースは，次の 3 つに大別される [1].

- 検知と調査の両面を兼ね備えるセキュリティ
- コンプライアンスや規制 (組織外)，規定 (組織内)
- システムとネットワークのトラブルシューティングと通常のオペレーション

開発や運用の場面において使われるものは，**システムとネットワークのトラブルシューティングと通常のオペレーション**である．例えば Web アプリケーションでは，デバッグ用途で変数値をログに出力したり，エラーの原因を特定する用途でスタックトレースの結果をログに出力したりする．こうして出力されたログは，開発における効率の向上や障害発生時の原因特定に役立つ．

IT システムにおける障害は，企業のビジネスのみならず社会に影響を及ぼす [2]．世界 3 位の規模である東京証券取引所で発生したシステム障害は，世界経済に 6 兆ドルの損失を招いた[*1]．システムにおける障害は，IT のみならずその背後にある社会へ影響を与えるため重要な課題である．こうしたシステムの障害への対処にはソフトウェアから出力されるログが有用である．管理者は，過去に発生した事象をログをもとに時系列で追跡し，因果関係から原因を特定する．こうした作業を行うには，常にログが取り出せることが求められる．また，高速にログを検索できることが求められる．

システムの規模が拡大するにつれ，サーバの台数やデプロイするソフトウェアの種類は増加する．これによりシステム全体で扱うログの総量も増加する．個々のサーバでソフトウェアが出力するログはサーバごとに内部で保管されるため，閲覧や検索のためには個別にログインの手間がかかる．こうした課題を解決するため，ログを管理するための数多くの取り組みが行われてきた．

サーバの増加に伴うログ管理の課題を解決する手法の 1 つは，Syslog によるログ専用サーバ (以下，ログサーバ) への集約である．それぞれのサーバで動作するソフトウェアから出力されたログは，Syslog によりログサーバへ転送される．ログメッセージの基本構造は RFC5424 や RFC3164 により定義された同一のものを使う．Syslog はログメッセージの作成，保存，転送のための単純なフレームワークを提供している [3]．ログサーバでは受信したログをストレージへ記録する．システム管理者はログサーバへログインすることで，個々のサーバから生成されるログを一元的に閲覧や検索を行える．IT システムの規模拡大に伴うログの増加すると，集中型ログ管理手法 (以下，集中型) はロ

[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

[*1] https://edition.cnn.com/2020/09/30/investing/global-stocks/index.html

グサーバへ負荷が集中する．そのため，システム管理者は，ログサーバのスケーラビリティを意識した構築や運用が求められる．したがって，集中型はシステム規模が増加するに連れシステム管理者の負担を増加させる．

ログ管理における課題の別の解決策は，PaaS(Platform as a Service) の利用である．Splunk や LogDNA に代表される PaaS は，ログ管理のための構築済みプラットフォームを提供する．サービスの利用者は自身の管理する専用のエージェントソフトウェアを導入するだけで PaaS を通じた一元的なログの閲覧や検索が行える．PaaS は，ログを管理するための専用サーバの構築や運用を必要としないためシステム管理者の手間を省き効率的な運用を実現する．しかし，一般的に PaaS はシステムの拡大にともなうログ件数やサーバ台数の増加により，サービス利用料が増加する．サービスに保管された目的や要件を満たさず意味をもたないログも利用料を圧迫する．管理者はサービス利用料を抑えるためにログの取捨選択をする必要がある．HP OpsAnalytics に代表される IT ベンダーが開発している製品は，PaaS で発生するログの総量の増加に伴う利用料増加は発生しない [4]．また，PaaS に比べ先進的なデータ分析ツールを提供する．しかし，自社の IT サービスと強い結びつきがありベンダーロックインが発生しやすく汎用性が不十分である．

IT システムにおいてログはシステムとネットワークのトラブルシューティングと通常のオペレーションのために重要である．これまで Syslog による集中型や PaaS の利用によりログの一元管理は実現されてきた．本稿ではこうしたログ管理手法をふまえ，新たなログ管理手法の提案を行う．

1.2 課題

ログをシステム全体で分散させ管理する手法では，分散させることにより集中型の課題である負荷の集中を分散により軽減できる．従来の手法は，アクセス頻度に基づく配置やラウンドロビンによる配置が行われてきた [5,6]．しかし，従来の手法では実際に動作しているシステムの構成には着目していない．ログを生成するプログラムによりログの種類は異なり，システム構成によって同一のログ形式であってもその役割は異なる．システムを構成するコンポーネントの種類や日時を絞った検索を高速に行うには，システム構成に着目したログの配置が必要となる．ログ管理システムにおけるログ検索時の応答速度を高速にすることは，トラブルシュートや開発における作業効率の向上につながる．作業効率の向上はシステムにおけるトラブルの解決までの時間の短縮や開発効率の向上に伴う開発期間の短縮を実現する．

1.3 各章の概要

まず，2 章では関連研究を紹介する．次に，3 章では本研究における提案内容を説明する．4 章では実装方法と実験環境を示す．5 章では検証方法とその結果を述べる．6 章では現状の課題とその解決方法を議論する．最後に本研究の提案とそれに伴い得られた成果を改めて示す．

2. 関連研究

アクセス頻度をもとにレプリカ配置をする研究では，収集したアクセス頻度に基づきレプリカを動的に再配置している [5]．再配置によりメッセージ通信量の最適化を行うことにより，システムの性能の向上を実現した．この手法では，利用頻度の高い検索条件であっても初期状態ではレプリカが最適な配置にならず，検索の応答速度に課題がある．

グループ志向特性によるポリシーをサポートするタスクフレームワークを実装した研究では，グループ化されたサーバの呼び出しとレプリカ管理用の属性を連携させることで障害が発生したときの応答速度を改善した [7]．この研究ではレプリカの内容に応じた配置が配慮されずログに適用した場合に検索の応答速度に課題がある．

分散システムでのログ管理手法では，分散したホスト上のログの断片を 1 つのログとして扱い，ネットワーク透過的な検索機能をもったログ管理システムを提案した [6]．しかし，この手法では分散させているので検索する時の性能の考慮が課題である．

ログ・ファイル分散管理方式の検討では，ログ情報の内容と冗長性を保証するためにログファイル格納前にログ情報を複数ログファイルに分散保存する方式を提案した [8]．分散時に当該ログ情報のハッシュ値と共に保存することにより，発生順序性を含んでログ内容を保全し，改竄や削除への耐性を実現した．セキュリティの改ざん対策に着目しているため，検索する時の性能の考慮に課題がある．

3. 提案

本研究では，検索時の性能に配慮した分散ログ管理手法を提案する．提案ではログファイルをノード間でログ配置ルールに基づき互いに複製を行う．ログ配置ルールは検索シナリオとシステム管理者の入力したシステム構成情報の統合により作成する．検索シナリオは頻度の高い期間 (例: 最新 3 日分) を想定した上で定義した．システム構成情報はシステムを構成するコンポーネント (ノード) を役割 (例: ロードバランサ) で分類して記述する．検索におけるログの取り出し方を想定して配置ルールを作成することにより，検索時の応答速度の向上を図る．提案は，ログの複製と配置を行う複製フェーズとログの検索を行う検索フェーズの 2 つから構成される．

複製フェーズにおける概要を図 1 に示す．システムではマスターノードと一般ノードの 2 種類から構成される．マスターノードは，一般ノードの情報 (IP アドレスやホスト名，システムにおける役割) を一元管理する．一般ノー

ドは，システムを構成するソフトウェア (アプリケーションやミドルウェア) が動作し，ローカルディスクにログファイルが書き出される．なお，考慮するパラメータを削減するため一般ノードの性能は同一とした．以降では，複製フェーズの手順を説明する．システム管理者は，システム構成情報を設定ファイルに記述しマスターノードで動作する Replication Manager に入力する．Replication Manager では，システム構成情報と検索シナリオをもとにログ配置ルールを作成する．ログ配置ルールをもとに Replication Manager は一般ノードで動作する Replication Agent に複製したいログの種類と複製先ノードの IP アドレスを通知する．Replication Agent では受け取った通知をもとにローカルディスクに保存されたログファイルを開き，複製先ノード宛にログを転送する．転送されたログは Replication Agent で受信されローカルディスクにファイルとして書き込まれる．

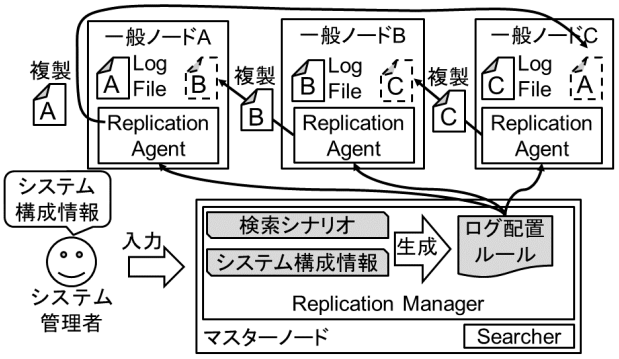

図 1 複製における提案の概要

検索フェーズにおける概要を図 2 に示す．マスターノードではログ配置ルールをもとに検索クエリの受け取りと一般ノードへの検索依頼の実行，検索結果の取りまとめを行う．一般ノードでは，データベースにログを保管しマスターノードからの検索依頼の受け取りと検索の実行, 検索結果の返答を行う．以降では図 2 における検索時の動作手順を述べる．システム管理者は，検索クエリをマスターノードで動作する Searcher へ入力する．Searcher は受け取ったクエリの解析を行い，ログ配置ルールから検索対象のログが配置されているノードを選択し検索条件を検索依頼を送信する．検索依頼を受信した一般ノードは検索依頼に含まれる検索条件をもとに一致するレコードをデータベース (Database) から取り出し Searcher に返す．Searcher は一般ノードから受け取った検索結果を統合しシステム管理者へ検索結果として返答する．

3.1 システム構成情報

本提案では，システムアーキテクチャの構成をもとに，ログの配置を決定する．IT システムは，動作させるアプリケーションの特性に応じてアーキテクチャが設計される．例えば，図 3 は，アプリケーション (Application) とデータ

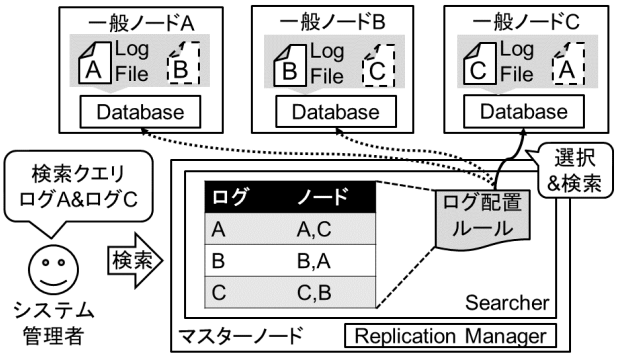

図 2 検索における提案の概要

ベース (Database)，ロードバランサ (LoadBalancer) から構成されるアーキテクチャである．LoadBalancer と Application, Database のそれぞれで生成されるログは Access Logs と App Logs, Query Logs と OS Logs に分類される．Access Logs には，Nginx から出力されるログが含まれる．App Logs は WordPress から出力されるログが含まれる．Query Logs は MySQL の発行クエリが含まれる．OS Logs は，オペレーティング・システム (例: Linux や BSD) から出力されるログが含まれる．こうしたアーキテクチャは

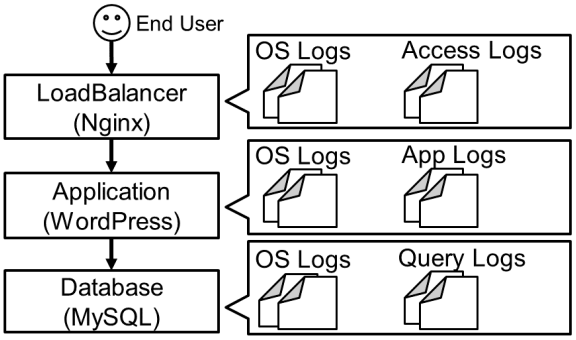

図 3 WordPress のアーキテクチャ

動作させるアプリケーションの性質に応じて異なる．そのため，システムにおいて障害が発生した場合，システムのアーキテクチャをもとに障害の対処を行う必要がある．これはシステムが目的ごとに複数の種類の仮想マシンやネットワーク機器から構成されることにより，システム障害の課題が連鎖的に複雑化しやすいためである．こうした背景から本研究ではシステム管理者が入力したシステム構成情報をログ配置の決定に利用する．

アーキテクチャにもとづくシステム構成情報は，システム管理者が設定ファイルに記述しソフトウェアに入力する．システム構成情報には次の情報が含まれている．なお，グループは複数台のマシンから構成されるマシンをまとめた単位である．

- マシンの IP アドレス
- マシンのホスト名
- グループの名前
- グループ間の依存関係

本提案では，ログを種類ごとに同じ役割をもつノードごとのグループに分ける．図 3 では LoadBalancer と Application，Database では OS Logs, Access Logs, App Logs, Query Logs の 4 種類が出力される．この場合，作成されるグループは次に 6 個である．

- LoadBalancer, OS Logs
- LoadBalancer, Access Logs
- Application, OS Logs
- Application, App Logs
- Database, OS Logs
- Database, Query Logs

これらグループごとに所属するノード間で対象ログを複製する．例えば LoadBalancer で生成された OS Logs のグループでは，LoadBalancer ノード群で生成されたログをグループに属するノード間で複製する．

3.2 検索シナリオ

ログの検索は，特定の目的や要件を満たすために行われる．目的の 1 つに IT システムにおける障害にともなうトラブルシューティングがある．システム管理者は．発生した障害の原因をシステムの全てのログから特定し対処する．提案では，こうしたログから障害の原因を特定する作業を想定した検索シナリオを構築する．これによりシステム管理者の目的を満たした高速なログ検索を実現する．

複数のコンポーネントから構成されるシステムにおいて想定されるログ検索シナリオを図 4 に示す．横軸は過去から現在にかけての時間変換を表す．縦軸は低レイヤーから高レイヤーにかけてのレイヤー分布を定義している．ここ

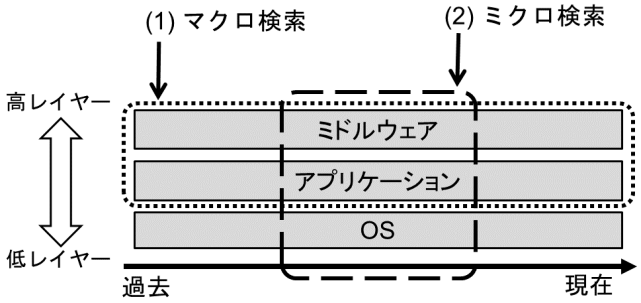

図 4 検索シナリオの分析

では検索シナリオとして (1) マクロ検索と (2) ミクロ検索を定義した．**マクロ検索**は障害が発生した時の初期診断を想定する．具体的には，障害が発生した時間帯の絞り込みや障害と関連性の高いコンポーネントの絞り込みを行う．こうした作業は，ミドルウェアやアプリケーションに代表される高レイヤーのソフトウェアから出力されるログを数日から数週間の広い期間で検索する．レイヤーを分けずに全てのログを検索した場合，対象となるログの件数が多くなり検索結果の取得に時間がかかり障害の原因に時間がかかる．そのため，効率的な障害対応には対象ログを絞った検索が必要になる．**ミクロ検索**は詳細な分析による障害原因の特定を想定する．マクロ検索により絞り込まれた時間帯とノードをもとに高レイヤーから低レイヤーまでのログを対象として検索を行う．具体的には，アプリケーションをはじめとする高レイヤーのログに加え OS を含む低レイヤーのログを時間帯を絞って細かく分析する．これにより OS やハードウェアに起因する障害を含めたログ分析が行える．

以下では，本提案におけるマクロ検索とミクロ検索に含まれる具体的な検索シナリオを示す．なお，検索シナリオは単一での利用に加え，複数による組み合わせを想定する．

時間帯を絞った検索

現在を起点に過去 N 週間 (N=1,2,3)，過去 M 時間 (M=24,48,72) のログを検索する．

グループを絞った検索

同じ役割のマシン (例: LoadBalancer) のログを一括で検索する．

レイヤーを絞った検索

図 4 に示したレイヤー (例: ミドルウェア) に限定して検索する．

3.3 ログ配置ルール

アーキテクチャと検索シナリオに基づくログ配置ルールの作成を説明する．本提案では，システム・アーキテクチャと検索時のシナリオを統合することによりログ配置ルールを生成する．以降では，ログ配置ルールの作成手順を説明する．

まず，システム管理者は，テキスト形式で記述したシステム構成をログ配置ルール生成ソフトウェア (以下，ルール生成ソフト) へ入力する．その後，ルール生成ソフトは入力した構成と検索シナリオを統合してログ配置ルールを生成する．検索シナリオは日時期間を絞ったもの (例: 過去 1 週間，過去 12 時間) を想定する．

ログ配置ルールの作成において，ログチャンク (Log Chunk) を導入した．Log Chunk(以降，チャンク) はログを複製する際のデータのまとまりである．チャンクごとに転送先ノードを決定し，転送を行う．チャンクは次の要素に基づきルール生成ソフトにより作成される．

- ログの作成日時
- ログの生成元ノード
- ログの生成元ソフトウェア

ログの作成日時は，一定期間ごとでログを 1 つのチャンクとして扱う．ログの生成元ノードは，ログが出力されたノード名をもとにログを 1 つのチャンクとして扱う．ログの生成元ソフトウェアは，Nginx や MySQL に代表されるソフトウェア名をもとにログを 1 つのチャンクとして扱う．例えば，LoadBalancer1 と LoadBalancer2, LoadBalancer3

で動作する Nginx から出力されたアクセスログが 3 日分 (2020/10/24〜2020/10/26) ある．このとき，1 日ごとにチャンクを作成すると，次の 9 種類のチャンクが作成される．

(1) LoadBalancer1, Nginx, 2020/10/24
(2) LoadBalancer1, Nginx, 2020/10/25
(3) LoadBalancer1, Nginx, 2020/10/26
(4) LoadBalancer2, Nginx, 2020/10/24
(5) LoadBalancer2, Nginx, 2020/10/25
(6) LoadBalancer2, Nginx, 2020/10/26
(7) LoadBalancer3, Nginx, 2020/10/24
(8) LoadBalancer3, Nginx, 2020/10/25
(9) LoadBalancer3, Nginx, 2020/10/26

上記のチャンクを 3 台のノードに配置した様子を図 5 に示す．図では LoadBalancer1 から LoadBalancer3 までの 3 台で 2020/10/24 から 2020/10/26 までの Nginx のアクセスログを分散配置させた．図の上部のチャンクはノード内部で生成されたログを表し，図の下部のチャンク複製ログを表す．ログを日付ごとにノードを分けて配置することにより，日付を絞った全ノードのログを検索する際の応答速度の向上を図る．

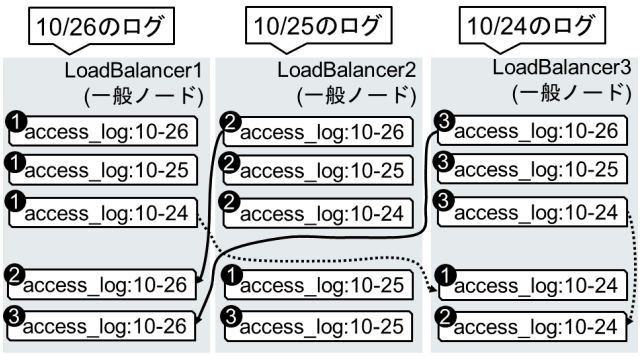

図 5 ログレプリカの配置 (3 ノード，3 日分)

4. 実装と実験環境

4.1 実装

ノードはマスターノードとそれ以外 (以下，一般ノード) の 2 つに分けられる．マスターノードでは，一般ノードのログ複製先の管理や検索クエリの集約を行う．こうした処理を実現するため Replication Manager を実装し配置した．一般ノードでは，ログの転送と受信を担う Replication Agent を実装し配置した．マスターノードは 1 台，一般ノードは N 台 ($N >= 1$) とする．

システム管理者が入力するシステム構成情報はソースコード 1 の YAML 形式で表現した．コード内の loadbalancer ではシステムにおけるロードバランサの役割をもつノード郡であることを示す．logfile_basename ではソフトウェアから出力されるログファイル名の形式を表現している．depends_on では loadbalancer と接続するシステム内の他のノード群の名前をリスト形式で記述する．nodes では loadbalancer に属するノードの IP アドレスとホスト名を記述する．

ソースコード 1 システム構成情報

```
loadbalancer:
  logfile_basename: "access_log"
  depends_on:
    - application
  nodes:
    - ipaddr: 10.1.0.8
      name: replica-agent-dep-b79d4d6dc-55
          shn
    - ipaddr: 10.1.0.15
      name: replica-agent-dep-b79d4d6dc-9
          gm68
application:
  logfile_basename: "app.log"
```

Replication Manager は Python 3.8.5 により実装し，ログ配置ルールは JSON 形式の中間ファイル rule.json(ソースコード 2) として出力した．ファイル内の nodeAddr はログが生成されたノードの IP アドレスを表す．logFilename はログファイルのファイル名を，loDate はログファイルのチャンク単位 (期間) を表す．この JSON ファイルを Replication Manager から読み取り Replication Agent へ HTTP 経由でログ複製先を通知する．Replication Agent は Python 3.8.5 によるプログラムと Fluentd(td-agent 1.10.2), Grafana Loki v2.0.0 により実装した．Replication Agent における Replication Manager との通信には Python 用 Web フレームワークである FastAPI(v0.61.1) を利用した．Fluentd はログシッパーとよばれるログの処理や転送を行うソフトウェアである．Loki はログの検索や取り出しをはじめとする一括管理を実現するインメモリ方式を採用するソフトウェアである．

ソースコード 2 ログ配置ルールの例

```
[
  {
    "nodeAddr": "10.1.0.8",
    "logFilename": "access_log",
    "logDate": "20201020"
  },
  {
    "nodeAddr": "10.1.0.8",
    "logFilename": "access_log",
    "logDate": "20201021"
  },
]
```

ログ複製の実装を図 6 に示す．以下ではログ複製時の各コンポーネントの動作を説明する．Replication Manager では生成したログ配置ルールをもとに一般ノードで動作する Replication Agent に複製先を HTTP 経由で通知する．Replication Agent は受け取った通知をもとに監視対象のログファイル (Log File) の中から条件に一致するログを選び別ノードに転送する．実験・評価にあたりログをノードごとに異なるものにするため，ログ生成プログラム (Log Generator) を作成し，動的にログファイルを生成した．生成するログは外部公開された Web サイトのアクセスログをもとに拡張している．他のノードで動作する Replication Agent から転送されてきたログは，Fluentd で受信した後にノードのローカルにノード，日付で分類され保存される．

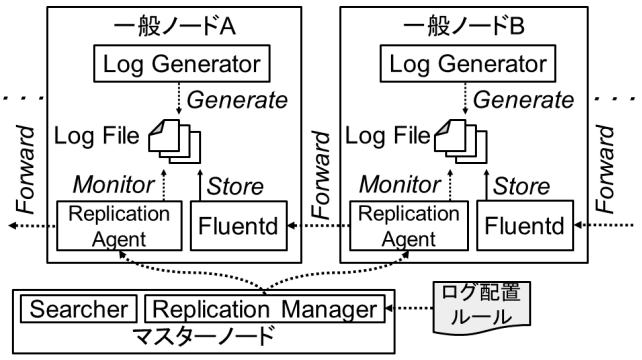

図 6 ログ複製の構成

ログ検索の実装を図 7 に示す．Fluentd で受信されローカルに保存されたログを，Fluentd を使い Loki に追加する．オペレータ (Operator) はログの検索を行うには，検索クエリをマスターノードの Replication Manager へ HTTP 経由で発行する．Replication Manager は検索クエリを解析し，複製されたログが配置されているノードを検索しクエリを各ノードで動作する Loki に検索クエリを発行する．各ノードで動作する Loki は Replication Manager からのリクエストをもとに格納されたログを検索してその結果を Replication Manager へ返答する．Replication Manager は検索クエリを発行した Loki から受け取った返答を統合しオペレータに返答する．

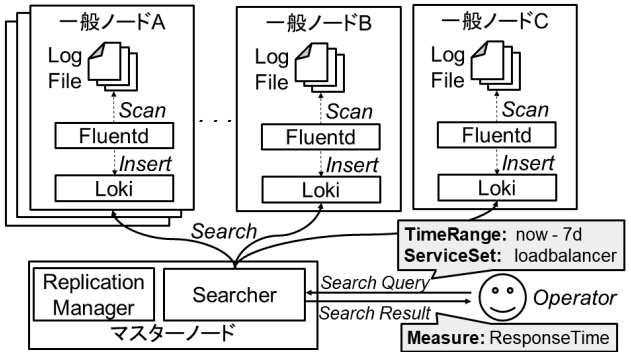

図 7 ログ検索の構成

4.2 実験環境

実装したソフトウェアを図 8 の環境にデプロイした．検証環境は Ubuntu 18.04 をインストールした 3 台の仮想マシン上に K3s による Kubernetes クラスタ (v1.18.3+k3s1) を使用した．仮想マシンはいずれも同一のハードウェア性能 (CPU: v1[core], RAM: 4[GB], SSD: 50GB) で作成した．Kubernetes はコンテナオーケストレーションツールでありコンテナの作成が容易に行えるため採用した．Kubernetes 上にマスターノードに相当するコンテナ 1 台 (以下，マスターコンテナ) と一般ノードに相当するコンテナ (以下，一般コンテナ) を作成し配置した．マスターコンテナは独自に作成した Replication Manager(Replica Manager) と構成情報ファイル (Manifest File) を Python3 のコンテナイメージ (python:slim) 内に配置し起動した．一般コンテナは，次の 4 つを Debian ベースの Python3 イメージ (python:3.7.9-stretch) 上に配置し作成した．なお，ログ・ファイル (Log File) はコンテナ起動時に Log Generator が起動して自動生成する．Log Generator は環境変数により生成するログの件数 (初期値:1000) と日数 (初期値:5) を設定可能となっている．

- ログ生成プログラム (Log Generator)
- ログ転送プログラム (Replica Agent)
- ログ受信ソフトウェア (Fluentd)
- ログ検索ソフトウェア (Loki) を配置

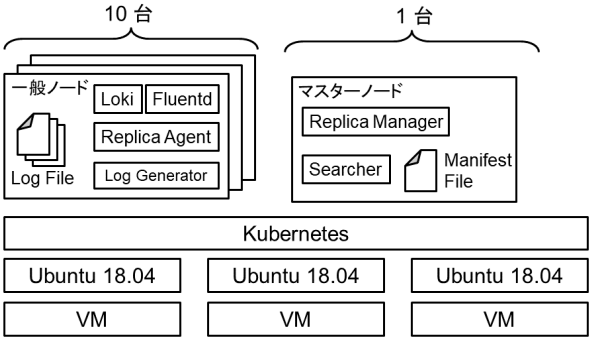

図 8 検証環境の構成

5. 評価と分析

実装および実験環境をもとに以下の手順で評価実験を行う．評価にあたり検索時の応答速度について提案手法とランダムに配置した手法で比較を行う．評価は次の手順で行った．

(1) Kubernetes 上に Nginx を想定したコンテナ (以下，一般コンテナ) を 10 個作成し，マスターノードを想定したコンテナ (以下，マスターコンテナ) を 1 個作成した．

(2) kubectl コマンドを使いコンテナの IP アドレス一覧を取得し，シェルのワンライナーを組み合わせることでシステム構成情報をあらわす arch.yaml を作成した．

(3) マスターコンテナに arch.yaml をコピーする．
(4) マスターコンテナの内部で配置ルール (提案手法の配置，ランダム配置) を指定して Replication Manager を起動する．
(5) Replication Manager は arch.yaml をもとにログ配置ルールを作成し，それをもとに一般コンテナで動作する Replication Agent にログ配置先の通知を行う．
(6) Replication Agent は通知をもとに別ノードで動作する Fluentd へログを転送する．
(7) Fluentd は受信したログをディスクに書き出し，Loki へ格納する．
(8) Searcher に検索クエリを発行し，検索の応答速度を測定する．
(9) コンテナ環境を kubectl コマンドで再作成し初期化する．
(10) 上記の手順を一般コンテナで生成するログの件数を 5000 行，50000 行のそれぞれに設定して提案手法とランダム手法での検索の応答速度を計測する．

実験では，Nginx が動作するロードバランサを 10 台想定した．Log Generator から生成されるログは，Nginx のアクセスログを想定した．ログは 1 日あたり 1,000 件と 10,000 件を 5 日分で用意した．全ノードのログを合計するとそれぞれ 5,000 件と 50,000 件になる．応答速度の計測にあたり検索条件を設計した．検索条件は，全ノードの 3 日分の Nginx アクセスログから GET を含む行とした．これらを提案手法とランダムに配置した手法のそれぞれで行い比較する．

図 9 に実験結果を示す．横軸は 1 台の一般ノードあたりで生成するログの件数をあらわし，縦軸は検索時の応答速度を秒数であらわした．5000 行のログを生成した場合，提案手法では検索時の応答速度は 0.12 秒，ランダムに配置した手法では 0.77 秒であった．50000 行のログを生成した場合，検索時の応答速度は提案手法で 0.12 秒，ランダム手法で 1.33 秒であった．これらの結果から，提案手法はランダム手法に比べ 5000 行のログでは 6.6 倍速く，50000 行のログでは 10.9 倍速い．ログの件数を 10 倍にしたとき，提案手法では検索の応答速度に変化がみられなかった．ランダム手法では，検索の応答速度が 1.7 倍遅くなった．

提案手法とランダム手法に差が生じた理由の 1 つに検索対象ノード数の削減がある．ログの検索では，検索クエリの条件を満たすノードへログ配置ルールに基づき検索依頼を送信する．提案手法ではログの配置を想定される検索条件に基づいて行った．そのため，日付を限定した検索において検索依頼の送信先となる一般ノード数が少なくなる．これにより，マスターノードと一般ノードの間での通信により発生する遅延が削減につながった可能性がある．

検索の応答速度に差が生じた理由として検索依頼の送信アルゴリズムがある．現状の実装では，検索時の検索依頼を直列で送信しており検索依頼を送信するノード数が増えるにつれ全体での検索結果を得るために通信するノード数が増加する．その結果，検索依頼を一般ノードへ送信する回数の多いランダム手法が提案手法に比べ応答時間が遅くなった可能性がある．

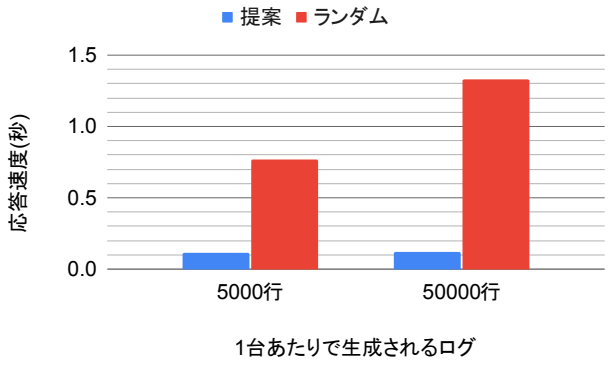

図 9 ログの検索における応答速度の比較

6. 議論

システム構成情報の入力を現在は手動により行っている．しかし，システム規模が大きくなるに連れ管理の手間も大きくなる．そのため，システム構成をプログラムにより自動生成することにより手動による作業量の削減が必要である．

検索シナリオの構築では，ユースケースを想定して検索条件を作成した．しかし，実際の検索クエリを使用していないため，検索条件の現実的であるか検討が必要である．改善にはログ基盤の検索条件のログや外部公開された資料の利用が考えられる．

ログチャンクの分割において現在は単位を 1 日としている．このサイズの変更 (例: 3 時間，1 週間) による応答速度の変化が十分に計測できていない．今後はサイズの変化に伴う応答速度の最小となるチャンクサイズを調べる必要がある．

現在の提案ではマスターノードと一般ノードの 2 種類のノードが存在する．しかし，マスターノードは単一障害点になるため冗長化が必要になる．冗長化はシステム構築や運用においてコストとなるため，最終的にはマスターノードの役割を一般ノードにもたせマスターノードを廃止したい．実現にはマスターノードの情報を分散して保持するための分散合意アルゴリズムの利用が必要である．

提案では一般ノードのハードウェア性能を同一 (ホモジニアス) とした．しかし，実際のシステムではハードウェア性能は異種 (ヘテロジニアス) である場合が多い．これには，配置ルールの作成時に個別の一般ノードの性能を変数として含めることで実現する．

検証ではノード数とログの件数が少ないためパラメータを増加させた場合にボトルネックがどこで発生するのか十

分に計測できていない．今後は，提案手法全体のパラメータ数を変更しながらシステム全体の性能にどう影響があるか計測を行う必要がある．

7. おわりに

本研究では，ログの分散管理手法における検索の応答速度を課題とした．この課題を解決するため，システム構成情報と検索シナリオに基づくログ複製の配置を提案した．検索を事前に想定したログ複製の配置を行うことで，検索の応答速度の高速化を行った．検証は 10 台のノードにおいて対象に検索の応答速度を提案手法とランダム手法を比較した．その結果，ランダム手法に比べ提案手法は最大 10.9 倍の高速化を実現した．提案によりログの分散管理における検索速度の改善を図った．これによりトラブルシューティングや開発において効率的なログ検索を実現した．

参考文献

[1] Chuvakin, A.: The complete guide to log and event management, *White Paper* (2010).
[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I. and Zaharia, M.: A View of Cloud Computing, *Commun. ACM*, Vol. 53, No. 4, p. 50–58 (online), DOI: 10.1145/1721654.1721672 (2010).
[3] Kent, K. and Souppaya, M.: Guide to computer security log management, *NIST special publication*, Vol. 92, pp. 1–72 (2006).
[4] Li, T., Jiang, Y., Zeng, C., Xia, B., Liu, Z., Zhou, W., Zhu, X., Wang, W., Zhang, L., Wu, J., Xue, L. and Bao, D.: FLAP: An End-to-End Event Log Analysis Platform for System Management, *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2017).
[5] Xu, Z., Xianliang, L., Mengshu, H. and Jin, W.: A Dynamic Distributed Replica Management Mechanism Based on Accessing Frequency Detecting, *SIGOPS Oper. Syst. Rev.*, Vol. 38, No. 3, p. 26–34 (online), DOI: 10.1145/1035834.1035838 (2004).
[6] 藤田元三郎，西尾　学：分散システムでのログ管理方式，全国大会講演論文集，Vol. 55, pp. 729–730 (1997).
[7] Morgan, G. and Ezilchelvan, P. D.: Policies for Using Replica Groups and Their Effectiveness over the Internet, *Proceedings of NGC 2000 on Networked Group Communication*, COMM '00, New York, NY, USA, Association for Computing Machinery, p. 119–129 (online), DOI: 10.1145/354644.354661 (2000).
[8] 江藤文治，高橋健一，堀　良彰，櫻井幸一：ログファイル分散管理方式の検討，コンピュータセキュリティシンポジウム 2009　(CSS2009) 論文集，No. 2009, pp. 1–6 (2011).