情報処理学会論文誌 Vol.60 No.3 728–737 (Mar. 2019)

時間軸検索に最適化したスケールアウト可能な高速ログ検索エンジンの実現と評価

阿部 博[1,2,3,4,a] 島 慶一[5,b] 宮本 大輔[6,c] 関谷 勇司[7,d] 石原 知洋[7,e] 岡田 和也[7,f]
中村 遼[7,g] 松浦 知史[8,h] 篠田 陽一[3,i]

受付日 2018年6月25日, 採録日 2018年12月4日

概要：ネットワークのトラブルシューティングやセキュリティインシデントに対応するため，ネットワーク管理者はサーバやネットワーク，セキュリティ機器から出力されるログを蓄積しておき，トラブルの原因を特定するためにその内容を調査することがある．大規模なネットワークでは，出力されるログの量も多く蓄積・検索システムの規模も巨大化する．著者らは，大量に出力される機器のログを後のトラブル解決などで重要となる時間軸を考慮した手法で蓄積し，高速に検索する先行技術として Hayabusa を実装した．本研究では，Hayabusa の検索性能をスケールアウトさせるためのシンプルな分散システムを提案し，その性能を評価した．評価の結果，提案手法は 10 台かつ複数 Worker での分散環境にスケールアウトした状態でスタンドアロン環境での Hayabusa より約 36 倍高速に動作した．これは 144 億レコードの syslog メッセージを約 6 秒で全文検索可能な速度に相当し，実用上十分な速度といえる．

キーワード：syslog，全文検索，GNU Parallel，分散処理，SQLite3，ZeroMQ

Design and Evaluation of Scalable Syslog Search Engine Optimized for Time Dimensional Search Operation

Hiroshi Abe[1,2,3,4,a] Keiichi Shima[5,b] Daisuke Miyamoto[6,c] Yuji Sekiya[7,d]
Tomohiro Ishihara[7,e] Kazuya Okada[7,f] Ryo Nakamura[7,g] Satoshi Matsuura[8,h]
Yoichi Shinoda[3,i]

Received: June 25, 2018, Accepted: December 4, 2018

Abstract: Network administrators usually collect and store logs generated by servers, networks, and security appliances to identify the source of problems by investigating the contents of log information when they find network troubles and/or security incidents. The size of the system to store and search the log messages tends to be larger when the size of the target managed network becomes larger. We proposed a fast log storing and searching system "Hayabusa" which is optimized for time-dimensional search operation in our past work. In this paper, we propose a simple distributed system which adds scalability to the existing Hayabusa system. The evaluation results show that the distributed Hayabusa system consists of 10 servers (with worker processes) is 36 times faster than a standalone Hayabusa system. The time required to perform a full text search over 14.4 billions of recoeds data is just 6 seconds, which is fast enough for daily operations of administrators managing a large scale network.

Keywords: syslog, full text search, GNU Parallel, distributed system, SQLite3, ZeroMQ

© 2019 Information Processing Society of Japan

1. はじめに

ネットワーク管理者は，日々生成されるネットワーク機器からのログを収集して統計的な情報を確認することでネットワークの健全性を評価し，トラブルが発生した場合には実際のログを検索するなどしてその原因を特定し安定したネットワーク運用を実現している．また，セキュリティインシデントに対応するために，トラブルの対応方法と同様にログからどのようなインシデントが発生したかを探ることもある．大規模なネットワークでは，多くのネットワークやサーバ，セキュリティ機器から日々大量の通信を記録したログが出力され，ネットワーク管理者はそれら大量のログをストレージシステムに蓄積し高速にログを検索するシステムを管理している．

大規模なログ検索システムと蓄積システムを扱うために，クラスタリングシステムや専用の管理ソフトウェアを用いることも多く，ネットワーク管理者は本来の業務に加えて検索・蓄積システムの管理に時間を割く必要がある．「ログの蓄積」と「ログの検索」がシンプルに動作し，かつ複雑なクラスタリングシステムを用いずに実現できれば，ネットワーク管理者が検索・蓄積システムの管理に時間を割かれることはなくなり，ネットワークのトラブル対応やセキュリティインシデントの解析といった本来の業務に集中することができる．

本論文では，多数のマルチベンダ機器が出力する大量のログを高速に蓄積し，高速に検索可能なシステムの提案を行う．また，システムが扱うログの量が増加した場合にも，システムの検索性能が容易にスケールアウトし，検索速度が飛躍的に向上する分散システムのコンセプトモデルについて提案する．本提案では，多数のマルチベンダ機器で構成される Interop Tokyo 2017 のネットワークで収集された，600 以上のサーバ・ネットワーク・セキュリティ機器から出力された大量の実データをもとに，蓄積・検索性能の検証と評価を行う．

2 章では関連研究の調査と問題点を提示する．さらに本提案のもととなる先行研究「Hayabusa」について解説を行う．3 章では提案手法の分散 Hayabusa のアーキテクチャについて詳細を示す．4 章では実装方法を示し，5 章で性能を評価する．6 章では得られた結果から考察を行い，7 章でまとめと今後の課題を述べる．

2. 関連研究と先行研究

2.1 関連研究

MapReduce アルゴリズム [11] や Apache Spark [20] などの Hadoop エコシステム [2] は全文検索やログ解析によく用いられる．巨大な Hadoop クラスタや Spark クラスタはユーザに高速な検索性能/サービスを提供し，ストレージ容量や処理速度がスケールアウト可能な設計となっている．しかしながら，運用者が Hadoop クラスタを管理するのは難しく，構築でさえ専用ソフトウェアを必要とする．運用者がシンプルにクラスタを運用しようと努めても，規模が大きくなることによりハードウェアの故障率は高まり，故障箇所の特定や安定したクラスタ運用には複雑な知識と経験が要求される．

Hadoop エコシステムで利用される HDFS [15] や，Elasticsearch [3] は分散ストレージとして動作し高可用性を実現している．これらの分散ストレージはデータのコピーを保持し，故障時にデータが完全に失われないように動作するが，信頼性向上のために複雑なプロセスを経由してストレージにアクセスするために処理性能は低下する．

商用製品やサービスとして提供される Splunk [4] や VMware vRealize Log Insight [7] などは，ログ蓄積とインデクシング，高速検索に特化したソリューションである．検索性能は，動作させるクラスタの台数と性能に大きく依存する．しかしながらコスト面を考えると扱うログの量が増加した場合にクラスタの拡大と追加ライセンスが必要となり，高い性能と冗長性を実現するには，莫大なコストが発生する．

grep や awk などの UNIX コマンドもログの検索や集計に利用される．しかしながらこれらのツールを用いて高速な検索や集計を行うには，熟練した知識と専用のデータ構造を事前に定義し実行しなければならない．またこれらコマンドはシーケンシャルに実行され，複数ホストで処理を分散させることは基本的に想定されていない．

Google が開発した Dremel [14] をベースとしたクラウド

[1] 株式会社レピダム
Lepidum Co. Ltd., Shibuya, Tokyo 151–0071, Japan
[2] ココン株式会社
COCON Inc., Shibuya, Tokyo 151–0071, Japan
[3] 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology, Nomi, Ishikawa 923–1292, Japan
[4] 情報通信研究機構
National Institute of Information and Communications Technology, Koganei, Tokyo 184–8795, Japan
[5] 株式会社 IIJ イノベーションインスティテュート
IIJ Innovation Institute Inc., Chiyoda, Tokyo 102–0071, Japan
[6] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology, Ikoma, Nara 630–0192, Japan
[7] 東京大学
The University of Tokyo, Bunkyo, Tokyo 113–8654, Japan
[8] 東京工業大学
Tokyo Institute of Technology, Meguro, Tokyo 152–8550, Japan
[a] abe@lepidum.co.jp/h-abe@jaist.ac.jp
[b] keiichi@iijlab.net
[c] daisu-mi@is.naist.jp
[d] sekiya@nc.u-tokyo.ac.jp
[e] sho@c.u-tokyo.ac.jp
[f] okada@ecc.u-tokyo.ac.jp
[g] upa@nc.u-tokyo.ac.jp
[h] matsuura@gsic.titech.ac.jp
[i] shinoda@jaist.ac.jp

© 2019 Information Processing Society of Japan

サービスである BigQuery [8] は高速に動作するデータベースとして用いられる．BigQuery は 120 億レコードを 5 秒で全スキャンするデモ[*1]が Google のエンジニアによって行われたが，バックエンドで動作するサーバが数千台や数万台といわれる規模で運用されているため，莫大な運用コストが必要になる．

先行研究 [22] では，ベアメタルサーバ上に実装した Hayabusa の性能を評価した．本論文では，他の分散処理環境と比較するために，クラウド環境を検証環境に採用した．

2.2　先行研究である Hayabusa について

Hayabusa [10] は，Interop Tokyo で収集された大量の syslog を高速に検索するためのシステムとして設計された．**図 1** に Hayabusa のアーキテクチャを示す．Hayabusa はスタンドアロンサーバで動作し，CPU のマルチコアを有効に使い高速な並列検索処理を実現する．Hayabusa は大きく StoreEngine と SearchEngine の 2 つに分けられる．StoreEngine は cron により 1 分ごとに起動され，ターゲットとなるファイルから syslog メッセージを取り出し SQLite3 [5] ファイルへと変換する．ログデータは 1 分ごとの SQLite3 ファイルへと分割され，検索時に複数プロセスにより並列処理される．ログが保存されるディレクトリは以下のように時間を意味する階層として定義される．

```
/targetdir/yyyy/mm/dd/hh/min.db
```

時間情報をディレクトリパスの構造に埋め込んでいるため，データベース内部に時間に関する情報を保持する必要がない．これにより，処理に時間がかかる時間のクエリ条件を指定することなく，時間指定でのログ検索が可能になる．ログが保存される SQLite3 ファイルは FTS（Full Text Search）と呼ばれる全文検索に特化したテーブルとして作成され，全文検索のためのインデクシングにより高速なログ検索を実現する．

SearchEngine は，並列検索性能を向上させるために分単位に細分化される FTS フォーマットで定義された SQLite3 ファイルへアクセスを行う．各 SQLite3 ファイルへは GNU Parallel [16] を用いて並列に SQL 検索クエリが実行され，結果は UNIX パイプラインを経由して awk コマンドや count コマンドを用いて集計される．

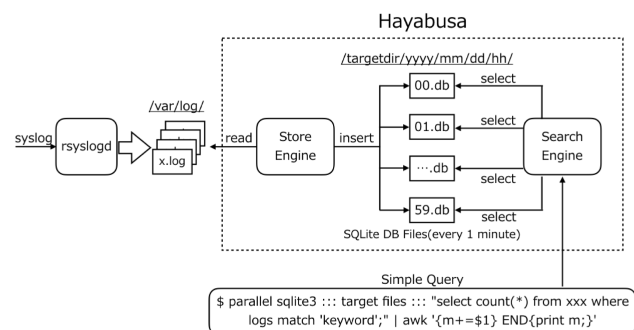


図 1　Hayabusa のアーキテクチャ
Fig. 1　Architecture of Hayabusa.

[*1] https://www.youtube.com/watch?v=swsS12c1VGE

Hayabusa はスタンドアロン環境で動作するが，小規模な分散処理クラスタよりも全文検索性能が高い．しかしながらスタンドアロン環境にはハードウェア限界が存在し，規模が拡大した他の分散処理クラスタにいつかは性能が追い越されてしまう可能性が高い．そこで本提案では，Hayabusa の限界であるスタンドアロン環境という制約を取り払い，複数ホストで Hayabusa の分散処理環境を構築し，検索性能がスケールアウトするアーキテクチャの実現を目指す．

3.　提案手法

本研究では，スタンドアロンで動作する Hayabusa を分散処理システムとして再定義し，検索処理性能をスケールアウトさせることを目標とする．

3.1　アーキテクチャ

分散 Hayabusa を定義するうえでストレージとスケジューラに関して定義を行う．アーキテクチャを定義するにあたり，システムの複雑化を避けるためにスタンドアロン Hayabusa のシンプルさを継承しつつ，処理性能を低下させない設計を目指す．Hadoop のような分散処理システムは，システムが健全・堅牢に動作するためにつねにシステム内部に複雑なデータのやりとりが発生する可能性があるが，分散 Hayabusa では，エラー処理や処理失敗時のリトライ処理を考慮せずにシステムのシンプルさの維持と処理速度を重視する方針で設計を行う．**図 2** に分散 Hayabusa のアーキテクチャを示す．

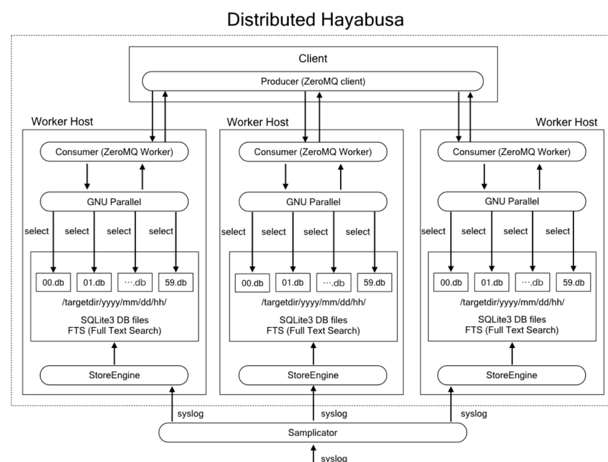


図 2　分散 Hayabusa のアーキテクチャ
Fig. 2　Architecture of Distributed Hayabusa.

© 2019 Information Processing Society of Japan

3.2 ストレージ

分散 Hayabusa のストレージは，スタンドアロン版と同様に SQLite3 の FTS とディレクトリ階層に時間をマッピングし，時間のクエリ条件を指定しなくても，時間範囲の検索ができる形とする．

さらに検索処理をスケールアウトさせるために，どの処理ホストに処理リクエストが届いても検索可能なようにすべての処理ホストに同一のデータを保持させる．これはすべての処理ホストへと syslog データを複製して配送することを意味する．これにより，どの処理ホストへと処理リクエストが渡っても同じ結果が返ることが保証される．また syslog が複製されることにより，データの保全性が高まり処理ホストが故障しデータが消失したとしても他の処理ホストにデータが残り対故障性が向上する．

3.3 スケジューラ

分散 Hayabusa は，検索処理のスケジューリングに RPC（Remote Procedure Call）を用いたロードバランシングと GNU Parallel によるプロセス実行機構を用いる．分散したホストへの検索処理投入に関して，Producer/Consumer モデルを用いた RPC と処理を均等に振り分けるロードバランシングにより，各ホストへと順番に検索処理が投入される．各ホストで受け取った検索処理がスタンドアロン版の Hayabusa と同等に GNU Parallel の並列検索として実行され，結果を Worker 経由でクライアントへと返す．

4. 実装

4.1 並列蓄積

4.1.1 データの複製

syslog をすべての処理ノードへと複製するために本研究ではオープンソースソフトウェアである，UDP Samplicator [6] を利用した．UDP Samplicator は，受信した UDP パケットを送信元アドレスを変更せずに，指定した対象ホストへと転送する．これにより転送先のホストは，あたかも自身が直に送信元からデータを受信したかのように UDP パケットを受信することができる．図 2 に示すように，すべての処理ホストは複製された同じ syslog パケットを受信する．

4.1.2 マルチプロセス化による負荷軽減

UDP Samplicator は 1 プロセスで UDP の転送処理を行う．そのため，大量の syslog を受信し負荷が上昇した際にプロセスのコア使用率が 100%となりパケット転送処理が追いつかなくなる場合があり，データが破棄される可能性がある．そこで本提案では socket のオプションに「SO_REUSEPORT」を利用して UDP Samplicator へパッチを当て，マルチプロセスとして動作するようにソースコードに修正を行った．これにより大量に syslog を受信したときに，1 CPU コアではボトルネックになりがちな syslog パケットの複製と転送処理を，複数の CPU コアを利用し複数プロセスを起動することで解消した．

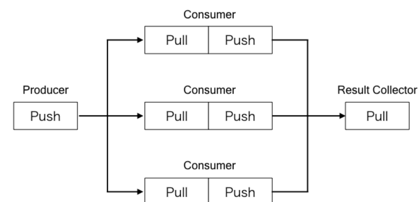


図 3 Push/Pull パターン
Fig. 3 Push/Pull pattern.

4.2 分散検索

検索処理リクエストはキューイングされ，Producer/Consumer モデルで処理される．Consumer にあたる処理ホストはキューイングされた処理リクエストを取得するが，このときに Producer は各ホストに均一に処理リクエストが行き渡るようにロードバランスを行う．

Producer/Consumer モデルは多くのソフトウェアで実装可能であるが，本研究では処理が高速に実行可能で，ライブラリとしてクライアントと Worker プロセスを実装可能な ZeroMQ [12] を用いた．ZeroMQ は高速に動作する分散メッセージキューとして利用され，「Request/Response」「Publish/Subscribe」「Push/Pull」など多様なメッセージングパターンを容易に実装することができる．本提案では，「Push/Pull」パターンを用いて Producer/Consumer モデルを実装する．

4.2.1 ZeroMQ の Push/Pull

図 3 で示すように ZeroMQ の Push/Pull パターンは以下の順序で処理が行われる．

1) Producer がリクエストをキューイング（Push）
2) Consumer が Producer からリクエストを取得（Pull）
3) Consumer が結果を Result Collector へと送る（Push）
4) Result Collector で取得した結果（Pull）を集計する

本提案でのクライアントは，Producer と Result Collector の 2 つの役割を持つ実装とする．これによりリクエストの発行からキューイング，結果の取得と集計を 1 プロセスで行うことができる．**図 4** にクライアントのソースコードを示す．

図 5 で示すようにクライアントは，処理ホストへ投入する処理リクエストをキューイングし，各ホストで動作する Worker が処理を Pull し実行した後に結果をクライアントへと送信し，クライアントが結果の集約を行う．クライアントは TCP の 5557 番ポートを用い Worker からの接続を待ち受け，処理リクエストをキューに Push する．処理結果は，TCP の 5558 番ポートで受け付け集計を行う．

次に，Worker のソースコードを**図 6** に示す．Worker はクライアントへの接続をブロックし，クライアントが動作

© 2019 Information Processing Society of Japan

したタイミングでクライアントがオープンした TCP 5557 番へ処理リクエストを Pull するために接続する．その後 Pull した処理リクエストを取得し，リクエストに含まれる

```
1  import sys
2  import zmq
3
4  context = zmq.Context()
5  sender = context.socket(zmq.PUSH)
6  sender.bind("tcp://*:5557")
7  receiver = context.socket(zmq.PULL)
8  receiver.bind("tcp://*:5558")
9
10 cmd = 'parallel target-data "SQLite3 Query
        Strings"'
11
12 sender.send(cmd.encode('utf-8'))
13 message = receiver.recv()
14 print(message.decode('utf-8'))
```

図 4　クライアントソースコード
Fig. 4　Example of client code.

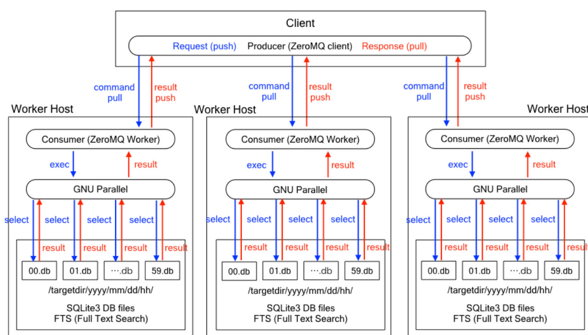


図 5　分散 Hayabusa で用いる ZeroMQ の Push/Pull パターン
Fig. 5　Push/Pull pattern using ZeroMQ on Distributed Hayabusa.

```
1  import zmq
2  import subprocess
3
4  context = zmq.Context()
5  receiver = context.socket(zmq.PULL)
6  receiver.connect("tcp://client:5557")
7  sender = context.socket(zmq.PUSH)
8  sender.connect("tcp://client:5558")
9
10 while True:
11     recv = receiver.recv()
12     cmd = recv.decode('utf-8')
13     res = subprocess.check_output(cmd)
14     sender.send(res)
```

図 6　Worker ソースコード
Fig. 6　Example of Worker code.

コマンドを実行した後に結果をクライアントの TCP 5558 番ポートへと Push する．

5.　評価

本研究では，スケールアウト試験を行うために Amazon Web Service [1] で提供される EC2 上の仮想サーバ群を用いた．スケールアウト試験では，仮想サーバを 1 台から 10 台へと増やし検索速度の評価実験を行う．処理ホストのほかに，分散クエリのリクエストを行うクライアントホストを 1 台用意する．実験ホストのスペックは**表 1** である．

5.1　分散検索

2017 年の ShowNet の syslog 受信サイズは，実データを解析したところ会期期間（6 月 7 日から 9 日の 3 日間）に入ってから 1 分あたり平均約 5 万件の受信量であった．将来的にはさらなる syslog 受信量の増加が見込まれるため，10 万件の syslog を用いて分散環境でのスケールアウト検索の検証を行った．

5.1.1　処理ホストのスケールアウト

スケールアウト性能を調べるため，処理ホストが増加した場合に処理時間が短縮するか試験を行った．処理ホストは 1 台から 10 台の範囲で増加し，クライアントは 1 日分のデータに対して繰り返し 100 回リクエストを実行する．100 回分のリクエスト対象のレコードサイズは，144 億レコードとなる．処理結果を**図 7** に示す．

ホスト 1 台時の検索処理時間は約 249 秒だが，ホストの台数を増やすに従い処理時間は減少し，10 台のホストの処理時間は約 39 秒になる．ホストを増加させた場合に検索処理がスケールアウトした結果となった．ここでの各処理

表 1　実験環境
Table 1　Experiment environment.

EC2 インスタンス	c4.4xlarge
vCPU	Intel Xeon CPU E5-2660 (2.9 GHz/16 core)
メモリ	30 GB
ディスク (EBS)	SSD 8 GB (OS) + SSD 50 GB (Data)
OS	Ubuntu 16.04.4 LTS (Xenial Xerus)

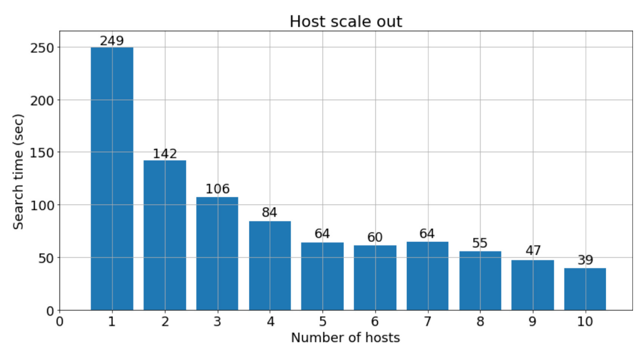


図 7　検索ホストのスケールアウト性能
Fig. 7　Host scale-out performance test.

© 2019 Information Processing Society of Japan

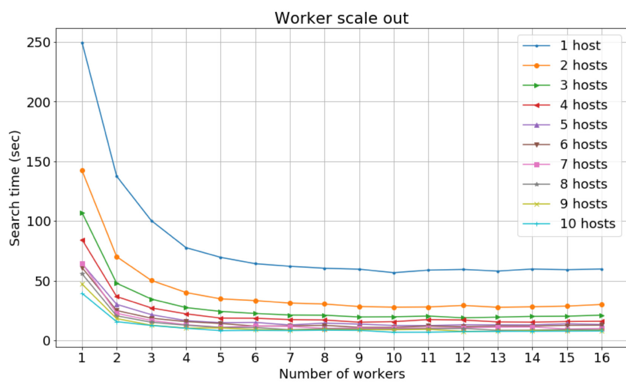


図 8 Worker プロセスのスケールアウト性能

Fig. 8 Worker scale-out performance test.

時間は 10 回試行した平均値である．

5.1.2 Worker プロセスのスケールアウト

次に 10 台のホストで処理を実行し，Worker の数を 1 から 16 の間で増加させた．16 という数字の根拠は仮想マシンの vCPU コア数であり，vCPU コアの数だけ Worker プロセスを増加させた場合にどの程度性能が伸びるか試験を行った．処理結果を図 8 に示す．

各処理ホストを 1 台から 10 台に増加させ，さらに Worker プロセス数を 1 から 16 まで増加させた図となる．ホスト数が 1 台から 10 台まで変化するなか，Worker 数が 10 あたりで最高値となる．1 ホスト 1 Worker 時に約 249 秒かかっていた処理が，10 ホスト 10 Worker 時に約 6.8 秒まで処理時間が短縮され，Worker の数を増やすことによる処理のスケールアウトも確認できる．こちらの実験も，各処理時間は 10 回試行した平均値である．

5.1.3 AWS EMR との比較

次に，AWS 上でサービス提供される，Amazon EMR（Elastic MapReduce）サービスとの比較を行った．EMR は指定した台数で Apache Hadoop/Hive/Spark などの Hadoop エコシステムが構築できるサービスである．かつ，Amazon S3 サービスにデータを置くことで，HDFS 環境を準備することなく EMR から直に S3 のデータを参照することが可能となる．

本実験では，分散 Hayabusa の評価と同等の性能の EC2 インスタンス（c4.4xlarge）を用いた．EMR では，構成として 1 マスターノードが必須となり，それ以外の環境として実際にデータの処理を行うコアノードが必要となる．ホストを増加させたときのスケールアウト性能を確認するために，コアノードの数を 2 台から 10 台まで増加させ評価を行った．使用した EMR のリリース番号は emr-5.12.0 であり，アプリケーションとして，Apache Spark がメインのパッケージ（Spark: Spark 2.2.1 on Hadoop 2.8.3 YARN with Ganglia 2.7.2 and Zeppelin 0.7.3）を選択した．

S3 に，1 ファイル 10 万行の syslog ファイル 1 日分にあたる 1,440 ファイル用意し，クライアントから 1 日分のデータに対して繰り返し 100 回リクエストを実行して，対象の syslog 行サイズが 144 億行という分散 Hayabusa と同等の情報量へアクセスする状況で実験を行った．

```
1  import time
2  from pyspark.sql import SQLContext
3
4  sqlContext = SQLContext(sc)
5  lines = sc.textFile("s3://abe-work/ssd2/
       benchmark-log/files/100k/100k-*.log")
6  lines.cache()
7
8  for i in range(5):
9      start = time.time()
10     [lines.filter(lambda s: ’noc’ in s).
           count() for i in range(100)]
11     elapsed_time = time.time() - start
12     print elapsed_time
```

図 9 PySpark ソースコード

Fig. 9 PySpark code.

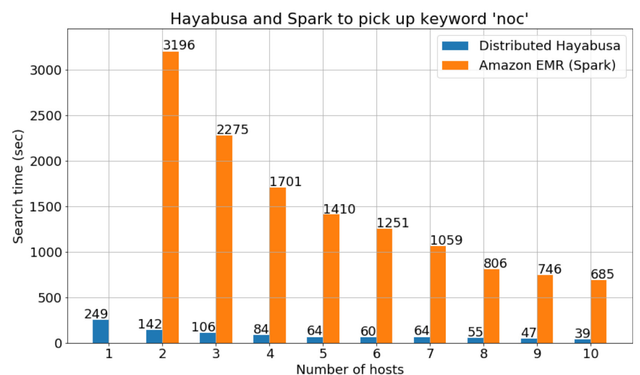


図 10 Hayabusa と Spark の性能比較

Fig. 10 Hayabusa and Spark time comparison.

クライアントは，図 9 で示す PySpark のコードをマスタノードで実行し，各コアノードで検索処理を行う．5 行目で，S3 から対象ログデータを読み込み，6 行目で Spark の機能である RDD（Resilient Distributed Dataset）[19] へとデータをロードする．RDD は複数コアノード間でシェアされる分散共有メモリであり，複数ノード間で高速にデータを検索できる．

処理結果を図 10 に示す．EMR ではホスト 1 台での環境は構築できないため，ホスト 2 台からとなる．こちらの各処理時間は 5 回試行した平均値である．EMR 上の Spark でもホストを増加させた場合に，全文検索の処理性能がスケールアウトしていく．本実験では，同じ syslog データに対する全文検索結果として，EMR Spark が 10 台構成の場合と比較して分散 Hayabusa の方が 17 倍高速に動作することを確認した．

© 2019 Information Processing Society of Japan

6. 考察

6.1 検索性能/処理のスケールアウト

5.1.1 項と 5.1.2 項で示した結果から，1 台の処理ホストとの検索時間と比較したときに，ホスト数のスケールアウト実験で約 6.3 倍処理時間が高速化した．またホスト数と Worker 数の両方のスケールアウトを組み合わせた結果，1 台の処理ホストで約 249 秒かかっていた検索時間が約 6.8 秒まで短縮され，約 36 倍高速化した．対象となるレコード数は 144 億レコードであり，144 億レコードを 6 秒台でフルスキャンできたということは，数千台以上のサーバを用いている Google の BigQuery に匹敵するデータのフルスキャン速度が実現できたことを意味する．10 台の処理ホストでこれだけの高速なスキャンを実現できるということは，コスト的に考えてもリーズナブルで高性能な分散処理システムが実現できたといえる．

6.2 ログデータ蓄積の並列化

本研究では検索性能を向上させるために分散クエリに対応するため，各処理ホストに同一の syslog データを複製する手法を用いた．これは，本質的には重複するデータを大量に複製する行為であり，データの量が増加すればするほどネットワーク帯域と保持するデータに無駄が発生することを意味する．Hadoop の HDFS のようにレプリケーション数を設定し，複数ホストでデータを分散させ保持することはもちろん可能であるが，その場合にはデータの管理をメタデータ管理機構で行い，データアクセスはメタデータ管理機構経由となり，処理性能を低下させる恐れがある．

本研究では，データを複製することによる帯域問題や容量問題が存在するが，機器の故障時には他の分散ファイルシステムのようにデータの再配置を行う必要もなく，シンプルに機器を管理対象から外すことで対応できる．機器故障以前に，各処理ホストへとデータが正しく配送されずに，処理ホスト間で保持するデータの一貫性が保証されないという問題が生じる場合があり，一貫性がないデータへとアクセスを行った場合に異なる値が返される可能性がある．また障害によりログの欠損が発生した場合には，同様に一貫性のないデータが結果として得られる可能性がある．処理ホスト間のデータ一貫性に関しては，ファイルのハッシュ値を用いたチェック機構を実装することで監視を行うことができ，監視によって一貫性の崩れが見つかった場合に，該当ホストを処理対象ホスト群から外すといった運用の実現が考えられる．障害によるログ欠損への対処は上記チェック機構を利用し，正しいログデータを該当ホストに複製する運用を行うことによりデータに一貫性のある状態へと復元可能となる．

今回採用した syslog の複製という手法は，本質的にスタンドアロンで処理可能なストレージデータ量しか処理していないことになる．問題を根本的に解決するには，ストレージの分散化が不可欠である．たとえば月ごとに時間軸単位にホスト増加させ処理を分散させる方法や，Amazon EMR のようにクラウドストレージにデータを保存し，各処理ホストから分散アクセスを行いデータを読み取る手法などが考えられるが，検索性能とストレージ性能の予備実験が必要となり，今後の課題とする．

6.3 シンプルな設計による運用の簡略化

本提案では分散検索を実現するために，データの複製機構の実現と Producer/Cosumer モデルによる検索の分散化を行った．設計と実装はともにシンプルであり，管理しなければならないプロセスの数も少ない．Hadoop のような分散システムは，多くの複雑なソフトウェアコンポーネントから実現されており，システムトラブルが発生した場合には，トラブル原因を把握するコストが高まる．きわめて少ないコンポーネントで作られる Hayabusa の分散処理機構は，問題が発生した場合にも原因の把握を高速に行うことができ，システム運用管理の負荷を軽減させる．

6.4 他の分散処理アーキテクチャの違い

本研究では，Amazon EMR との比較実験を行い，分散 Hayabusa の方が EMR より約 17 倍高速に全文検索を行うという結果を得た．これほど処理に差が出るにはいくつかの点があげられる．EMR の場合にはボトルネックとなりうる箇所が数カ所あげられる．

6.4.1 クラスタのリソース管理

EMR で使われる Spark は Hadoop のリソース管理機構である YARN（Yet Another Resource Negotiator）[18] 上で動作する．YARN は，Hadoop エコシステムにおけるリソース管理機構を司っており，クラスタ内のリソース割当てや実行されるジョブの監視とトラッキングを行い，さらにクラスタが保持する共通のデータセットに対するアクセスを管理する．これによりクラスタ全体の健全性と，マルチテナント化した場合に，複数ジョブの制御を行うことが可能となる．

分散 Hayabusa には現状リソース管理機構はなく，クライアントから届いたリクエストは速やかに Worker で実行するモデルになっている．これは分散 Hayabusa が高速に動作する仕組みの 1 つではあるが，ジョブの管理やトラッキングを行っていない以上，トラブルが発生した場合にエラーハンドリングやリトライ処理ができないことを意味する．

本提案でのシステム構成では，トラブルが発生したことを利用者が把握することはできず，処理結果からトラブルを推測することしかできない．そこで現在，クライアントと Worker の間にリクエスト管理機構を備える再設計を行っている．リクエスト管理機構を用いることにより，ど

© 2019 Information Processing Society of Japan

の Worker へ割り振ったリクエストがどのようなステータス（処理中・処理完了・エラー終了・タイムアウト）であるか判定可能となり，利用者がトラブルを検知することが可能となる．

6.4.2　スケジューラの構造

次にプロセス実行スケジューラについて議論する．Spark は，タスクをスケジューリングする前にタスク実行の有向非巡回グラフ（DAG: Directed Acyclic Graph）を作成する．また Spark は，RDD を用いて分散共有メモリ内にデータを保存し，DAG 間でデータを共有する．このように DAG 間でデータを共有（ディスクに中間結果を書き込まない）して最適化することで，高速にジョブを完了することができる．

Hayabusa が実現するスケジューリング機構は，ZeroMQ クライアントが行うロードバランシングと GNU Parallel の実行スケジューリングに依存する．シンプルでオーバヘッドが少ないので高速に動作するが，Spark のように最適な実行計画を計算し，実行計画に沿って分散共有メモリを使いながら処理を実現するようなものではない．

6.4.3　ストレージに対するアクセス性能

本実験で EMR は S3 からデータを読み取ったが，本来であれば Hadoop エコシステムは HDFS 上にデータが分散されて設置される．HDFS を使った場合にはデータがある一定以上のサイズになった場合にはブロック単位で各ホストに分散して配置されることになる．その場合には，データへのアクセスは HDFS のメタデータ機構を経由することになり，ストレージアクセスが低速になる．

Hayabusa はデータを 1 分単位の SQLite3 データベースとして保持し，個々のファイルは全文検索に特化した FTS フォーマットでインデクシングされた高速な検索機構を有する．さらに，時間レンジで絞り込みをかけたい場合に，Hayabusa の手法では時間を SQL のクエリ条件に指定する必要がなく，本来クエリ条件としては遅くなる可能性の高い時間レンジ指定を排除することで高速化が可能となる．また，メタデータ機構を経由しないため，高速なデータアクセスが可能となる．

6.4.4　マルチテナント化に向けた課題

本提案での分散 Hayabusa は，高速化を目指すためにリソース管理機構や明確なスケジューリング機構などいくつかの機能を実装していない．そのためエラーや例外が発生した場合に，リトライできないようなアーキテクチャとなっている．また，インフラのマルチテナント化を考えると分散 Hayabusa 環境を占有できない場合に，現状のクライアントから直にクラスタへとリクエストが渡る設計では，正常な利用はできない可能性がある．マルチテナント化を考えるためには，リクエスト投入部分でのリクエストの識別とキューイング，優先順位づけが必要となるが今後の課題とする．

6.5　評価手法定義の課題

本論文では，分散 Hayabusa と Amazon EMR との全文検索にかかる処理時間を比較対象とした．分散 Hayabusa はストレージと密結合に近い形で処理を行うが，EMR の場合は S3 を透過的に扱う部分で，ストレージアクセスに対して遅延が生じてしまう．また，Elasticsearch との比較を行う場合には全文検索部分に対する処理時間は計測可能だが，リクエストを投入する REST の処理にかかる時間とクライアントの距離によって処理時間が変化するという問題やシャードの数によるレスポンス時間の変化など考慮するポイントが存在する．予備実験では，Elasticsearch の全文検索はクラスタの性能に依存するが，100 万件の syslog データに対して数ミリ秒で応答を返すものが多かった．しかしながら，REST API の応答にはその 100 倍ほどの時間がかかり，結果トータルの応答時間は 0.1 秒以上になる場合が多く，リクエスト回数を増やした場合に Elasticsearch のエンジンは高速に動作するが，REST API の処理時間を含むトータル処理時間は Hayabusa に劣る結果になる場合がある．エンジンの処理時間，クライアントとの応答時間，ストレージアクセス，それらをふまえた応答合計時間を評価軸として，公正な評価指標を定める必要があるが今後の課題とする．

また，本実験では蓄積されたデータに対して検索を行ったが，蓄積と検索を同時に実施した場合の性能劣化に関して，各比較対象のシステムでどのような結果が出るかも評価手法定義の 1 つの課題となる．

7.　まとめと今後の課題

7.1　まとめ

本論文では Hayabusa の分散処理の設計と実装を行った．計測した結果，1 台の処理ホストでは約 249 秒かかった検索処理が最大約 6 秒まで短縮した．144 億レコードの syslog データを 6 秒でフルスキャンし，全文検索可能な分散 Hayabusa はログ検索エンジンとして高い性能を発揮する．これはマルチベンダ機器を管理するネットワーク管理者が大量のログを用いて，トラブルシューティグやインシデントレスポンスを行ううえで使い勝手の良いツールとなり，対応時間を短縮できる可能性がある．また，システムをシンプルに設計しているため，システム管理コストが著しく低くなり，ネットワーク管理者は本来注力したい業務へと時間を割くことができる．

7.2　今後の課題

本論文では Hayabusa の分散処理の実現と測定を行い高いパフォーマンスを実現した．しかしながら，他の処理系との比較がまだ十分に行えておらず，正しく評価可能な比較指標を定めたうえでさらなる比較実験を行い Hayabusa の優位性を示す必要がある．

© 2019 Information Processing Society of Japan

またHayabusaは検索基盤ソフトウェアとして動作するため，その上で動作する具体的なアプリケーションソフトウェアを組み合わせることでさらなる有益なシステムとなりうる．syslogを高速に検索できることから，先行研究であるインベントネットワークにおけるsyslogを用いた異常検知[21]と組み合わせることで高速に動作する異常検知アプリケーションを作成することが可能となったり，Chainer[17]，TensorFlow[9]，Pandas[13]などと組み合わせることで，機械学習フレームワークやデータ処理基盤との融合が可能となる．

謝辞 本研究の一部は，国立研究開発法人科学技術振興機構（JST）の研究成果展開事業「戦略的創造研究推進事業（CREST）JPMJCR1783」の支援によって行われた．

参考文献

[1] Amazon Web Service, available from ⟨https://aws.amazon.com/⟩.

[2] Apache Hadoop, available from ⟨http://hadoop.apache.org/⟩.

[3] Elasticsearch, available from ⟨https://www.elastic.co/products/elasticsearch⟩.

[4] Splunk, available from ⟨https://www.splunk.com/⟩.

[5] SQLite, available from ⟨https://www.sqlite.org/⟩.

[6] UDP Samplicator, available from ⟨https://github.com/sleinen/samplicator⟩.

[7] VMware vRealize Log Insight, available from ⟨https://www.vmware.com/products/vrealize-log-insight.html⟩.

[8] An inside look at google bigquery (2013).

[9] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y. and Zheng, X.: Tensorflow: A system for large-scale machine learning, *12th USENIX Symposium on Operating Systems Design and Implementation* (*OSDI 16*), pp.265–283 (2016).

[10] Abe, H., Shima, K., Sekiya, Y., Miyamoto, D., Ishihara, T. and Okada, K.: Hayabusa: Simple and fast full-text search engine for massive system log data, *Proc. 12th International Conference on Future Internet Technologies*, *CFI'17*, pp.2:1–2:7, ACM (2017).

[11] Dean, J. and Ghemawat, S.: Mapreduce: Simplified data processing on large clusters, *Comm. ACM*, Vol.51, No.1, pp.107–113 (2008).

[12] Hintjens, P.: 0mq – the guide (2011).

[13] McKinney, W.: Pandas: A foundational python library for data analysis and statistics (2011).

[14] Melnik, S., Gubarev, A., Long, J.J., Romer, G., Shivakumar, S., Tolton, M. and Vassilakis, T.: Dremel: Interactive analysis of web-scale datasets, *Proc. 36th Int'l Conf. Very Large Data Bases*, pp.330–339 (2010).

[15] Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The hadoop distributed file system, *Proc. 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies* (*MSST*), *MSST '10*, pp.1–10, IEEE Computer Society (2010).

[16] Tange, O.: Gnu parallel – the command-line power tool, *;login: The USENIX Magazine*, Vol.36, No.1, pp.42–47 (Feb. 2011).

[17] Tokui, S., Oono, K., Hido, S. and Clayton, J.: Chainer: a next-generation open source framework for deep learning, *Proc. Workshop on Machine Learning Systems* (*LearningSys*) *in the 29th Annual Conference on Neural Information Processing Systems* (*NIPS*) (2015).

[18] Vavilapalli, V.K., Murthy, A.C., Douglas, C., Agarwal, S., Konar, M., Evans, R., Graves, T., Lowe, J., Shah, H., Seth, S., Saha, B., Curino, C., O'Malley, O., Radia, S., Reed, B. and Baldeschwieler, E.: Apache hadoop yarn: Yet another resource negotiator, *Proc. 4th Annual Symposium on Cloud Computing*, *SOCC '13*, pp.5:1–5:16, ACM (2013).

[19] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S. and Stoica, I.: Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, *Proc. 9th USENIX Conference on Networked Systems Design and Implementation, NSDI'12*, p.2, USENIX Association (2012).

[20] Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S. and Stoica, I.: Spark: Cluster computing with working sets, *Proc. 2nd USENIX Conference on Hot Topics in Cloud Computing, HotCloud'10*, p.10, USENIX Association (2010).

[21] 阿部 博，敷田幹文：イベントネットワークにおけるsyslogを用いた異常検知手法の提案と実データを用いた評価，インターネットと運用技術シンポジウム2016論文集，Vol.2016, pp.57–64 (Dec. 2016).

[22] 阿部 博，篠田陽一：スケールアウト可能なログ検索エンジンの実現と評価，インターネットと運用技術シンポジウム2017論文集，Vol.2017, pp.73–80 (Nov. 2017).

阿部 博 （正会員）

株式会社レピダム研究員．ココン株式会社社長補佐/技術研究室研究員．北陸先端科学技術大学院大学篠田研究室博士後期課程所属．情報通信研究機構協力研究員．ACM会員．

島 慶一

株式会社IIJイノベーションインスティテュート技術研究所副所長．

宮本 大輔

奈良先端科学技術大学院大学特任准教授．

© 2019 Information Processing Society of Japan

関谷 勇司

東京大学情報基盤センターネットワーク研究部門准教授.

石原 知洋

東京大学大学院総合文化研究科助教.

岡田 和也

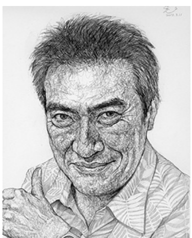
東京大学情報基盤センター情報メディア教育研究部門助教.

中村 遼

東京大学情報基盤センターネットワーク研究部門助教.

松浦 知史

東京工業大学学術国際情報センター准教授.



篠田 陽一

北陸先端科学技術大学院大学教授.

© 2019 Information Processing Society of Japan