Hanoi: 複数レイヤーのトレースログを用いたHadoopのパフォーマンス解析

清水裕亮[†] 櫻井孝平 山根智

ネットワーク上で相互に接続された複数の計算機を用いて, テラバイト・ペタバイト級のデータセットに対して分散並列処理を行う大規模分散処理システムがある. このような大規模分散処理システムを構築するためのフレームワークとして, Apache Hadoop があげられる. Apache Hadoop を用いることでユーザは, 大規模分散システムを構築する際に解決すべき非機能要件 (ネットワーク通信の非決定性の隠蔽や, 計算機の故障の管理等のフォールトトレラント性) を意識する必要なく, 容易に分散プログラムを書くことができる. しかし, このような大規模分散処理システムにおいては, ネットワーク上で相互に接続された複数の計算機上のプロセスは非決定的であり, システム動作の再現は困難である. また, 障害の種類も, アプリケーション自身のバグ以外に, ハードウェア, ネットワーク, クライアントの行動による障害など多層にわたっており, 種類が多いこともデバッグを困難とする原因といえる. 分散システムの動作の把握・デバッグのためには, クラスタを構築する各ノードの動作を複数レイヤーに渡ってリアルタイムに監視を行い, 取得したログをリアルタイムに収集し解析することが有効と考えられる. 本研究では, Hadoop を用いて構築されたクラスタを対象として, 複数レイヤーのログを収集・解析することで, システムのパフォーマンスを解析する. 計算機リソースの使用率、HDFS/MapReduce クライアントとの通信ログ、そして Java のメソッドトレースを収集し解析に利用しており, 既存研究と比べてログの粒度が細かく, より詳細な解析を可能とする.

Hanoi: Analysis of Performance of Hadoop using the Trace Log of Multi Layer

Yusuke Shimizu,[†] Kohei Sakurai and Satoshi Yamane

Large-scale distributed system processes a job using a huge number of computer that are connected to each other on the network, and manage a terabytes or petabytes of data sets. For building those massively distributed system, Apache Hadoop a framework designed for build a distirbuted system do good. Apache Hadoop enabled us to write and running a distributed program easily, thanks to it transparently fulfill non-functional requirements(non-deterministic behavior on network communication, fault-tolerance treat any fault of cluster node, etc..) that we must be solved. However, distributed systems works on network which is non-deterministic, so it is diffficult to reproduce behaviors of distributed system. In addition, distributed system involved various kind of failures (MapReduce application itself, hardware, network, and client failure). Debugging or understanding the distributed system behaviors is a difficult problem. For debugging and understanding the distributed system operations, it is valid to monitoring system operations at real-time across multi layers of each node, and analyze those collected logs. In this study, targeting a cluster built using Hadoop, we demonstrate detection of the peformance problem, using monitoring of system behavior and log analysis. Compared with existing research, our method differ in monitoring the method-trace of Java program, using fine-grained logs about system behaviors enabled us to analyze more advanced system states.

1. はじめに

今日クラウドコンピューティングが注目され, 広く利用されているが, その背景には大規模分散システムの構築のためのフレームワーク (Google MapReduce[1] など) の存在がある. Google MapReduce とは, テラバイト・ペタバイト級の大規模なデータを対象として, 高スループットな処理を行うための MapReduce プログラミングモデルの実装, フレームワークである. Google MapReduce のオープンソース実装である Apache Hadoop[2] は, Yahoo!, Facebook, Toreasure Data など多くの企業で開発・利用されている[3][4].

Apache Hadoop を用いることで Hadoop ユーザは, 大規模分散システムを構築する際に解決すべき非機能要件 (ネットワーク通信の非決定性や, 計算機の故障の管理等のフォールトトレラント性) を意識することなく, 実際のデータセットに対して行いたい処理の実装

† 金沢大学 自然科学研究科
Kanazawa University Natural Science & Technology

に集中することができる. また, その処理性能は計算機の台数を増やすことでスケールアウトする.

しかし, このような大規模分散処理システムのテスト・デバッグは困難である. 理由として, ネットワーク上で相互に接続された複数の計算機上で協調動作するプロセスは非決定的であり, システム動作の再現が困難であることがあげられる. 例えば, Apache Hadoop について、同じ MapReduce プログラムでも、タスクのスケジューリングや割り当て、フォールトトレランスや処理性能のスケールアウトの実現のための, タスクやブロックの転送や複製など Hadoop クラスタの内部での振る舞いは, 入力データのサイズや, クラスタの構成、ハードウェアリソースの状態, ネットワーク環境などによって大きく異なりうる.

同じ MapReduce プログラムでも, 内部・外部環境の影響によって, その処理性能が大きく異なってくる. 分散システムのテスト・デバッグについて, 従来のソフトウェア (シングルプロセス, 外部環境の影響を受けにくい, ネットワーク通信を行わないプログラム) のように, 静的で網羅的なテスト・検証を適応するのは, 分散システムの内部動作が, 内部・外部環境の影響によって非決定的であることから困難と言える. そのため Hadoop などの大規模分散システムのチューニングやデバッグは非常に困難であり、課題であると言える。

実際に, Apache Hadoop コミュニティに投稿されたバグレポートの統計を図 1 に示す. Hadoop のクラッシュ障害に関するバグレポートが全体の 31%に対して, パフォーマンス障害の割合は 20%と少ないが, パフォーマンス障害はシステムの停止などのように表に出にくいという特徴があり, さらに日常的に使用しているなかでは, ユーザは小さな性能の低下に気付かない, もしくは意識しないと考えられるため, 実際にはパフォーマンス障害は, レポートの集計結果よりも多数発生していると考える.

Hadoop コミュニティの場でも, Hadoop の内部動作の把握が困難な課題であると取り上げられている[5]. Hadoop のチューニングが困難であるために, Hadoop ユーザはトライアンドエラーなチューニングに頼っているという報告もある[6]. そこで, 大規模分散システムの内部動作の可視化やパフォーマンス・チューニング, デバッグのために, 今日さまざまなシステム監視とそのデータの解析の研究が行われている (4 節 “関連研究”). 大規模分散システムの内部動作の監視, デバッグは困難かつ、重要な取り組むべき課題として考えられる.

本研究では, 大規模分散処理システムを対象とした、複数階層のトレースログを用いたデバッグのための手法を提案する。 システム実行時の動作を監視しそのログのリアルタイムな収集・解析を行うことで、システムの動作や、パフォーマンスを評価し、デバッグに有効であることを示す。大規模な分散システムの動作に影響を与える要因が複数レイヤーにまたがっていることより, クラスタを構築する各ノードの動作を複数レイヤーに渡ってリアルタイムに監視を行い, 取得したログを用いて解析することがパフォーマンス障害の検知と, その原因特定のために有効と考えられる. 具体的には, Hadoop を用いて構築されたクラスタを対象に, 計算機リソースの使用率、DataNode とクライアントとの通信ログ、TaskTracker とクライアントの通信ログ、そして map/reduce タスクの java メソッドトレースを収集・解析する。Hadoop の入力データとして, Key の分布に偏りがある KeyValue データを与えた場合の, 計算機リソースの使用率, パフォーマンスの変化について解析・評価を行う. 本研究では Java のメソッドトレースを収集し解析に利用しており, 既存研究と比べてログの粒度が細かく, より多くのレイヤーからログを取得することで, より詳細なシステム動作の解析を可能とする.

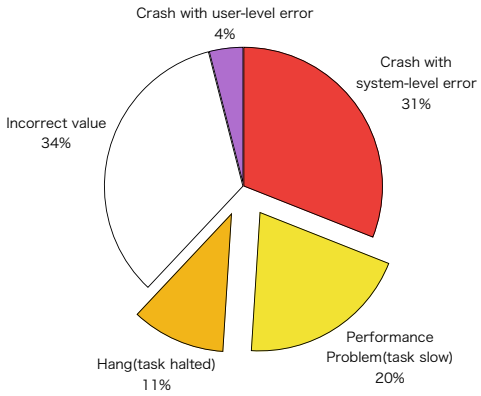

図 1: Apache Hadoop コミュニティへのバグレポートの種類とその割合

1.1 Hadoop

Hadoop は大規模分散処理システムを構築するためのフレームワークであり, Google が開発した MapReduce[1] のオープンソースクローンである. Hadoop Distributed File System(HDFS) と, Hadoop MapReduce から構成され, HDFS はテラバイト, ペタバイト級のデータを扱うために最適化されたファイルシステムであり,

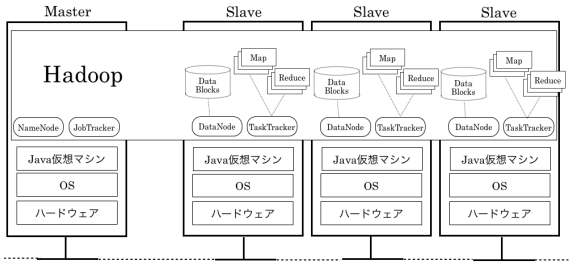

図 2: Hadoop クラスタの構成の概要図

入力データをブロック ☆ 単位に分割・管理し, スケールアウトと耐故障性のために複製などを行う. MapReduce は, HDFS 上のデータを効率よく処理するための MapReduce アルゴリズムを実行するフレームワークである. Hadoop はマスタ・スレーブ型のアーキテクチャであり, HDFS は NameNode と DataNode, Hadoop MapReduce は JobTracker と TaskTracker によって構成される (図 2). Hadoop は Java で記述されており、各デーモンは JavaVM 上で動作する。

1.1.1 Hadoop Distributed File System(HDFS)

HDFS は KeyValue データベースであり, 各データは Key とその値 Value の組からなる. HDFS では, Linux のファイルシステムと同様にひとつのファイルをブロックに分割し, このブロックの読み書きを行う際の最小の単位として管理を行う. ただし, Linux のファイルシステムにおけるブロックサイズが標準では 512 バイトであるのに対して, HDFS が扱うブロックサイズは標準で 64MB と大きいという特徴がある. 分散ファイルシステムにおいて, ブロックを抽象化することで, ネットワークで接続されたクラスタ内の計算機のディスクを使用し, 数テラ, ペタバイト級のデータの格納を実現する. HDFS 上のブロックの名前空間は NameNode が管理し, 実際のデータであるブロックはスレーブノードの DataNode に格納される. HDFS では, 耐故障性とスケールアウト性のために, ブロックの複製を行う. HDFS Client は, ユーザの HDFS へのアクセス要求にしたがって, NameNode と DataNode と通信を行いファイルシステムにアクセスを行う.

1.1.2 Hadoop MapReduce

Hadoop MapReduce とは, Google が考案した MapReduce プログラミングモデル[1] を用いて大規模データを処理するためのフレームワークである. Hadoop MapReduce(以降単に MapReduce と呼ぶ) では, 処理を map フェーズと reduce フェーズの 2 つに分割して行う. ユーザは, データの map 関数と reduce 関数の 2 つを記述することで, HDFS 上の数テラ, 数ペタバイトの膨大なデータを対象とした, 分散処理プログラムを書くことができる. Hadoop MapReduce は, マスタスレーブ型の構造をしており, マスターノードでは JobTracker が, スレーブノード群では TaskTracker が動作する. ジョブ ☆☆ が投入されると, JobTracker はジョブをタスク ☆☆☆ に分割し, TaskTracker にタスクの処理を依頼する. JobTracker は TaskTracker に依頼したタスクの処理状況を把握し, タスク全体のスケジューリングを行う. タスクは, その処理内容である map/reduce 処理と, その対象のデータのセットから構成されるが, ここで JobTracker はスケジューリングを行う際に, データ局所性の最適化のために, 処理対象のデータをもっているノードにタスクの割り当てを試みる. また, JobTracker のタスク管理の中には, タスク失敗時のフェイルオーバーの処理も含まれている.

2. 提 案 手 法

本研究では検証対象の大規模分散システムとして, Hadoop を取り上げる. Hadoop システムのトレースログを収集し、リアルタイムに可視化することで、Hadoop システムのデバッグに有効な解析を行うことを目的とする. 提案システムの概要図を図 3 に示す。

提案システムは主に次の 3 つの機構からなる。Hadoop クラスタを監視しロギングする機構 “Monitor”、分散配置されたログの収集を行う機構 “Log Collector”、ログの解析・可視化を行う機構 “Performance Visualizer” から構成される。

Monitor は、システム動作の監視によって与える負荷が少なく、またロギングの機能のために、Hadoop オリジナルのコードに変更を加える必要がない. Monitor は次のログを取得する。

- 計算機リソースの使用率
- map/reduce タスクの Java メソッドトレース
- DataNode/TaskTracker のクライアント間通信ログ
- jvm のメトリクス

Monitor である HanoiPickerDaemon は、各ノードの map/reduce タスク毎に起動し、そのタスクの動作を監視する。大規模な Hadoop クラスタになると、日常的にノードの不具合が発生しうる。そのため、LogCol-

☆ Hadoop 分散ファイルシステムが管理する, データを分割したもの. DataNode が扱う一単位.

☆☆ ユーザが Hadoop MapReduce に依頼する作業単位

☆☆☆ MapReduce プログラムを分割したものであり, Map/Reduce タスクなどの一単位

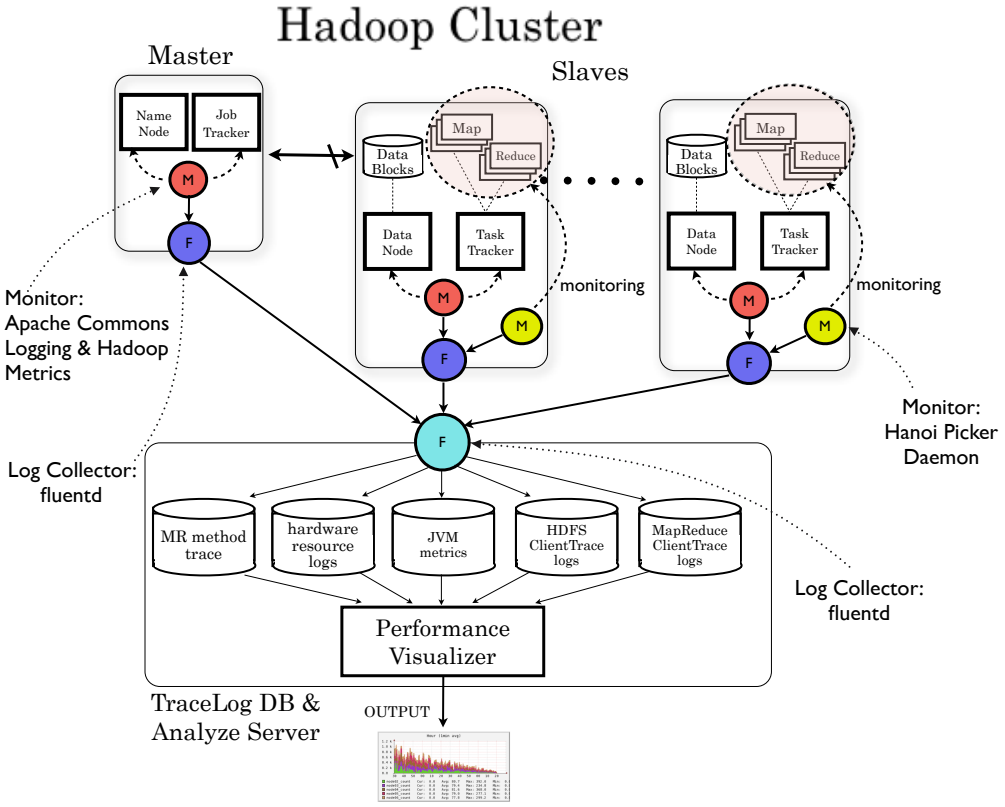

図 3: 提案手法概要図

lector はノードの不具合に対して耐故障性を備えている。Performance Visualizer について. Hadoop システムを構成する各ノードから収集する時系列ログデータは膨大となりうるが、継続的な稼働を可能とするため、解析サーバでは固定サイズのデータベースを採用する。

2.1 モニタリング

2.1.0.1 計算機リソースの使用率

計算機のリソース使用率として, CPU, メモリー, ディスク IO, ネットワーク IO の情報を dstat☆ コマンドを用いて取得する.

2.1.0.2 MapReduce メソッドトレース

MapReduce ジョブ実行時の, map/reduce タスクの Java メソッドトレースを取得する. HanoiPicker は、各ノードの map/reduce タスク毎に起動しその振るまいを監視し、ロギングを行う。さらに、取得したメソッドトレースの引数をローカル内でカウントアップし、ホットスポットの検出に役立てる。ここでローカル内で key のカウントアップをすることで、解析サーバに転送する際のネットワーク IO のボトルネックとなることを軽減できる。

ここで、map/reduce タスクの Java メソッドトレースのロギングの機能は, Hadoop オリジナルのソースコードとは別にモジュール化して管理すべきだが, これにはアスペクト指向プログラミングが適している. アスペクト指向とは, オブジェクト指向ではモジュール化の困難な横断的関心事をモジュール化するためのプログラミングパラダイムの一つである. アスペクト指向を用いることで, Hadoop のオリジナルのソースコードに直接修正を加えることなく, メソッドトレースのロギングの機能を実装することができる. Hadoop は Java 言語で記述されたフレームワークであるため, AspectJ を用いることができる. AspectJ[7] とは, アスペクト指向プログラミングの Java 実装である.

2.1.1 HDFS クライアントトレース

1.1 節で説明したように, NameNode は HDFS の名前空間を管理し, 実際のデータはブロックに分割され, DataNode にて保管される. そして, ユーザのディス

☆ http://dag.wieers.com/home-made/dstat/

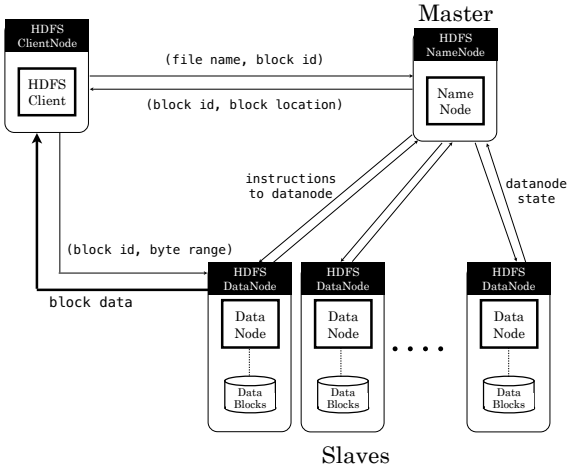

図 4: NameNode, DataNode, そして HDFS Client が行う通信の概要図

表 1: HDFS クライアント トレース

要素名	説明	例
date	日付	2013-11-10 17:27:51
src	ローカル HDFS Client の IP アドレスと port 番号	192.168.1.17:50010
dest	操作対象の HDFS DataNode の IP アドレスと port 番号	192.168.1.21:35024
bytes	送信バイトサイズ [byte]	66048
op	オペレーションネーム	HDFS_READ
cliID	クライアント ID	DFSClient_*****
offset	アクセスするブロックのオフセット	0
srvID	Jetty サーブレットの ID	DS_***_****
blockid	ブロック ID	blk_*****_**
duration	実行時間 [ms]	82494440

クアクセス要求に従い, HDFS のファイルを扱うために HDFS Client が存在する. HDFS 上での NameNode と DataNode, そして HDFS Client のメッセージ通信とデータ転送の概要図を図 4 に示す. 図 4 が示すように, NameNode と DataNode 間では, ブロックの受け渡しは行わず, HDFS Client を経由していることがわかる. つまり, HDFS Client と DataNode 間のログを監視することで、HDFS のファイル入出力の情報が取得できる。HDFS Client のログは, Hadoop が利用する log4j の設定から実現できる。取得した HDFS Client のトレースログからは, DataNode へ行った read/write の各アクセス処理について, read/write 操作を行ったバイトサイズ, ブロック ID, 処理の実行時間など各種メトリクス (表 1) が取得できる.

2.1.2 MapReduce クライアントトレース

HDFS クライアントとレースと同様に、MapReduce についても TaskTracker クライアントのトレースログ

表 2: TaskTracker クライアント トレース

要素名	説明	例
date	日付	2013-10-11 17:27:51
src	ローカル TaskTracker Client の IP アドレスと port 番号	192.168.1.17:50060
dest	shuffle 処理によるデータ転送先 IP アドレスと port 番号	192.168.1.19:40563
bytes	送信バイト数	11896
op	オペレーションネーム	MAPRED_SHUFFLE
cliID	クライアント ID	attempt_201311101546_***
duration	実行時間 [ms]	352858

を取得する。TaskTracker クライアントのログから、shuffle 処理でのデータの転送先と、バイトサイズ、実行時間などが取得できる 2。hadoop 設定ファイルの log4j を設定することでロギングできる。

2.2 ログデータの収集と管理

2.1 節で述べた各種トレースログは, Hadoop クラスタ上の各ノードのローカルファイルシステム上に作成されるため、それら分散配置されたデータの収集, 管理について述べる.

クラスタを構築するノードに分散配置しているトレースデータをほぼリアルタイムに収集することができ, また Hadoop システムのように日常的に障害が発生することが想定される環境でも動作する可用性と信頼性が求められる. 本研究では, このデータ収集における課題の解決のために, fluentd[8)☆] を利用する. fluentd とは, 複数サーバからデータをリアルタイムに収集するための, オープンソースのソフトウェアである. ログをネットワーク経由で収集する際に通信エラーが起きた場合, fluentd が提供するバッファ機能によって, 転送されるログデータはメモリ内に一時的に保持され, 後に再送を試みる.

データ収集・解析サーバでは、ログを保存するデータベースとして MongoDB を採用する。本提案手法では, 各ノードから収集されるログデータは時系列データとしてたえず生成され, 解析を行うサーバには膨大なディスクサイズが必要といえるが, MongoDB は capped コレクションという固定サイズのデータベースの扱いをサポートしている. これによって, 格納した順にデータは管理され, データベースの最大サイズを超えた場合, 古いデータから削除されるため, 本提案手法におけるトレースログを扱うのに向いていると考える.

☆ http://docs.fluentd.org. TreasureData Inc. (https://www.facebook.com/TreasureData) が開発した Ruby 製のソフトウェア. オープンソースとして公開されている.

3. 実験と考察

提案手法のシステムを構築し、Hadoop フレームワーク上でいくつかの MapReduce ジョブを実行し、トレースログや可視化結果からデバッグを試みる.

3.1 実験環境

実験環境を、表 4 に記してある性能のホストマシン上に仮想マシンを立ち上げて構築する. 各ホストマシンでは, 仮想マシンを一つ立ち上げる. 今回の実験では, 6 台のホストマシンを用いて, 6 台構成のマスタスレーブ型の Hadoop クラスタを構築した ☆. 実験環境の仮想マシンの性能を表 6 に示す.

検証対象の Hadoop クラスタを構築する計算機とは別に、解析サーバの性能を図 5 に記す.

Hadoop クラスタの構成を表 3 に示す. node01 がマスターであり, node02〜node06 がスレーブとなる.

表 3: Hadoop クラスタの構成

ホスト名	デーモン
node01	NameNode, JobTracker
node02〜node06	DataNode, TaskTracker

次に, 実験に関係する Hadoop の各種デーモンの起動パラメータを示す. HDFS のデーモンのパラメータを表 8, MapReduce のデーモンのパラメータを表 7 に示す.

3.2 実　　験

3.1 節で記した実験環境で実際に MapReduce プログラムを実行し、Hadoop クラスタの動作の解析を行う. 実験対象の MapReduce プログラムとして、ファイル中の単語の種類ごとの出現回数を数えるワードカウントプログラムを用いる. ワードカウントプログラムの map メソッドをソースコード 1 に、reduce メソッドをソースコード 2 に記す.

map メソッドでは, 入力文字列を単語で分割し, 各文字列にカウントアップのための重み “1” をつけた Key/-Value データとして出力する. 入力文字列とはジョブ実行時に指定したファイルをブロックに分割したデータのスプリット (ブロックの一行のデータ) のことを指す. reduce メソッドでは, 同一 key で value の値を加算する処理を行う.

☆ 仮想環境上で Hadoop クラスタを構築したのは、バージョンや実験環境の異なる複数の Hadoop クラスタを利用するためであるが、本実験に関して言えば特に意味はない.

```
1 public void map(Object key, Text value, Context context
2                 ) throws IOException, InterruptedException {
3   StringTokenizer itr = new StringTokenizer(value.toString());
4   while (itr.hasMoreTokens()) {
5     word.set(itr.nextToken());
6     context.write(word, one);
7   }
8 }
```

ソースコード 1 : 実験用プログラム map メソッド

```
1 public void reduce(Text key, Iterable<IntWritable> values,
2                    Context context
3                    ) throws IOException, InterruptedException {
4   int sum = 0;
5   for (IntWritable val : values) {
6     sum += val.get();
7   }
8   result.set(sum);
9   context.write(key, result);
10 }
```

ソースコード 2 : 実験用プログラム reduce メソッド

実験では, 上記のワードカウントプログラムを繰り返し実行する. その際に, 提案する Monitor がシステム動作を監視し各種ログを生成する. それらは解析サーバに収集・格納・可視化されるためそのデータを用いて, システム動作の把握を試みる.

まず、node04 での CPU 使用率を図 5 に示す。図 5 のグラフ全体にわたってグラフ上部に空白がある。これは IO wait が発生していることを意味する。また同グラフの中央のあたりでは、CPU 使用率が急激に減少している。この原因を調べるため、node04 のメモリ使用率のグラフ (図 6) を見る。

node04 のメモリ使用率が同じタイミングで急激に減少していることがわかる。

jvmmetrics から取得できる node04 のガーベジコレクションの発生回数のグラフを図 7 に示す。これより、図 5 で CPU 使用率が急激に減少したタイミングでガーベジコレクションが行われていたことがわかる。

ここで、DataNode クライアントの通信ログから作成した、HDFS からの読み込みバイト数のグラフを図

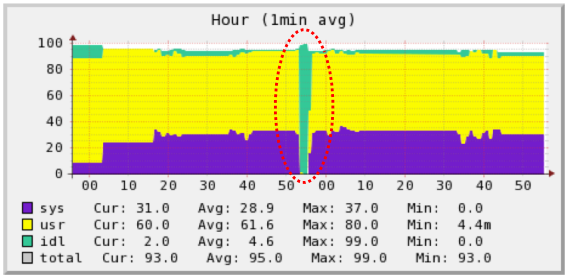

図 5: node04 の CPU 使用率

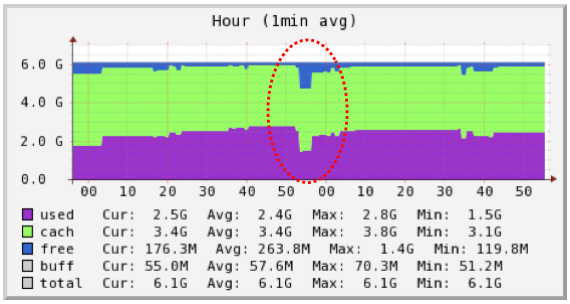

図 6: node04 のメモリ使用率

8 に示す. これによると、node04 では図 5 で CPU 使用率が急激に減少した時に、HDFS から約 50M バイトのサイズのデータを読み込んでいたことがわかる。

次に、TaskTracker の Java メソッドトレースログから作成した reduce メソッドの実行回数のグラフを表 9 に示す。これより、先のタイミングの直後に約 2200 回の shuffle 処理が実行されたことがわかる。つまり、HDFS からのデータ読み込みが発生していたのは shuffle 処理のためのコピーフェーズであったと考えられる。また、そのタイミングで、map フェーズで使用したメモリのガーベジコレクションを行ったと分かる。

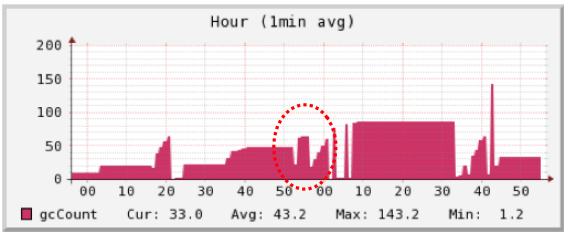

図 7: node04 でのガーベジコレクションの発生回数

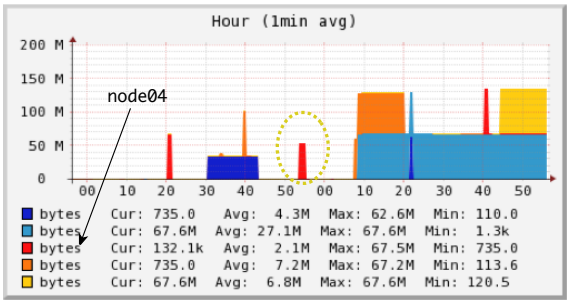

図 8: HDFS ファイルシステムからの読み込みバイト数

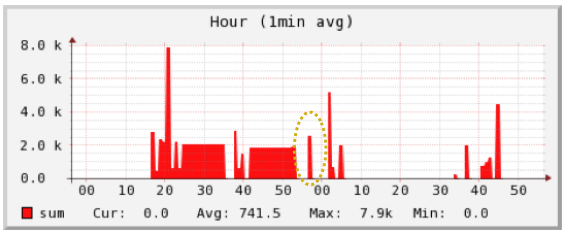

図 9: node04 での reduce 処理での実行回数

4. 関連研究

HiTune[9] は, Hadoop のような大規模分散システムに対して, 内部動作の把握が困難であると指摘し, それを解決するためにデータ駆動型パフォーマンス解析ツールを開発した. これによって, Hadoop システムの高次のデータフローから, 低次元のパフォーマンスボトルネックとなっているであろう挙動を紐付けることを実現した.

Kahuna[10] では, Hadoop を対象としたシステムにおいて, パフォーマンスボトルネックの発生の検知を対象としている. Hadoop クラスタ内の各ノードの計算機リソース使用量と Hadoop のログを用いて解析を行い, その 2 つのメトリクス値についてノード間で比較を行い, パフォーマンスボトルネックの検知を行う. パフォーマンス障害の検知を目的とする点は等しいが, 本研究ではさらにパフォーマンス障害の原因の切り分けを行うことまでを目的としていることに違いがある. そのため, 本手法ではより詳細なトレース情報が求められ, MapReduce の Java メソッドトレースを取得している点に大きな違いがある.

Starfish[11] では, Hadoop を対象として, BTrace[12] を用いてメソッドトレースの取得とガーベジコレクションの時間などの情報を取得し解析を行うことで, 数千におよぶ Hadoop のパラメータ項目の最適な設定を提案することを目的とする. 本研究と比較し, 目的の違いはあるが, 注目するログがリソース使用状況と, Java メソッドトレースである点は等しい. しかし, Starfish では, そのメソッドトレースの実行時間に注目するためメソッドがコールされ, その実行結果がリターンされるまでの時間に注目しているのに対して, 本手法では, Java メソッドトレース内に現れる map タスクのパラメータとしての key の値に注目している. Java のメソッドトレースの取得の手法も異なり, Starfish が BTrace を用いているのに対して, 本研究では AspectJ を用いた.

Chukwa[13] Chukwa とは, 大規模な分散システムを管

理するためのオープンソースデータ収集システムであり, Hadoop DistributedFileSystem(HDFS) と Hadoop MapReduce フレームワークを土台に構築されている. Hadoop 自身では提供していない, 自身のログデータ収集のためのパイプラインを提供し, その解析のためのツールが備えられている. HDFS を土台としたシステムのため, 長期間の膨大なログから, トレンドを発見することに秀でている. Chukwa が, 長期間のログを保管しそれらを用いて, システム動作の可視化やトレンドの発見を目的とするのに対して, 本研究では目的がパフォーマンス障害の検知であるという違いがある. パフォーマンス障害の検知はリアルタイムに行えることが望ましく, そのため, 本研究では, ログのリアルタイム性を重視したログ収集と, 解析機構を構築した. また, Chukwa が各種タスクの処理時間やディスク使用率などのメトリクスを収集しているのに対して, 本手法では目的の違いより, HDFS へのアクセスログや, MapReduce プログラムの Java メソッドトレースを取得している.

分散システムの内部動作をブラックボックスとして扱う手法:
分散システムの内部動作はブラックボックスとして扱ったまま, ネットワーク通信のログをトレースすることで, 実際に実行されたパスの中から, パフォーマンスボトルネックとなっているパスを導出する[14]. PinPoint[15]: クライアントと分散システム間のエンドツーエンドのリクエストのメッセージ通信をトレースし, システムの障害の箇所を検出する. ただし, クライアントリクエストのパス自体は, 正確には一台の計算機内のものしかトレースすることはできない. そこで, 高次元での障害を低次元の障害に関連付けるデータマイニング手法を用いた. 本研究でも, システムの内部動作をブラックボックスとして扱い, その入出力の結果に注目し, 分散システムの障害やパフォーマンスボトルネックの検知を目的とする点で本研究と関連している. 既存研究の手法では, メッセージ通信のトレースログに注目しており, それは本提案手法である Java のメソッドトレースや, 計算機リソースの使用率を取得するのに比べ, より高次でのブラックボックス化を行なっていると言える.

5. ま と め

本論文では, 大規模分散システムのデバッグが困難であることを指摘し、有効な手法として, システム実行時のロギングとそのログデータのリアルタイムな解析手法を提案した。実際に, Hadoop を用いて構築された分散システムを対象に, 提案システムを構築し実験を行い、それらのログ・可視化した情報がシステム動作の把握に有効であると示した.

今後の展望として, 収集したログデータを用いて, Hadoop システムのパフォーマンスの定量的評価手法とその自動化が考えられる.

参 考 文 献

1) Dean, J. and Ghemawat, S.: MapReduce: simplified data processing on large clusters, OSDI '04: PROCEEDINGS OF THE 6TH CONFERENCE ON SYMPOSIUM ON OPERATING SYSTEMS DESIGN AND IMPLEMENTATION, USENIX Association (2004).
2) : Apache Software Foundation(2007), "Hadoop", `http://hadoop.apache.org/core`.
3) : Hadoop Wiki, `http://wiki.apache.org/hadoop/PoweredBy` (2012).
4) inc., T.: *2012 Big Data Survey Results* (2012).
5) Todd Lipcon: Hadoop and HBase at RIPE NCC, http://blog.cloudera.com/blog/2010/11/hadoop-and-hbase-at-ripe-ncc/.
6) Pavlo, A., Paulson, E., Rasin, A., Abadi, D.J., Madden, S., Csail, M. I. T., Stonebraker, M. and Dewitt, D. J.: A Comparison of Approaches to Large-Scale Data Analysis (2009).
7) Meredith, P. O.N., Jin, D., Griffith, D., Chen, F. and Ros, G.: An Overview of the MOP Runtime Verification Framework (2011).
8) : fluentd: open-source log collector,treats logs as JSON, `http://fluentd.org`.
9) Dai, J., Huang, J., Huang, S., Huang, B. and Liu, Y.: HiTune : Dataflow-Based Performance Analysis for Big Data Cloud.
10) Marinelli, E., Kavulya, S., Gandhi, R. and Narasimhan, P.: Kahuna: Problem diagnosis for Mapreduce-based cloud computing environments, *2010 IEEE Network Operations and Management Symposium - NOMS 2010*, Ieee, pp. 112–119 (2010).
11) Herodotou, H., Lim, H., Luo, G., Borisov, N. and Dong, L.: Starfish : A Self-tuning System for Big Data Analytics, *In Proc. of the 5th Conference on Innovative Data Systems Research(CIDR'11)* (2011).
12) : BTrace: A Dynamic Instrumentation Tool for Java., `http://kenai.com/projects/btrace`.
13) Boulon, J., Rabkin, A., Berkeley, U.C., Konwinski, A. and Yang, M.: Chukwa : A large-scale monitoring system, *In Cloud Computing and its Applications (CCA '08)*, pp. 1–5 (2008).
14) Aguilera, M. K., Mogul, J. C., Wiener, J. L., Reynolds, P. and Muthitacharoen, A.: Performance debugging for distributed systems of black boxes, *Proceedings of the nineteenth ACM symposium on*

Operating systems principles - SOSP '03, p. 74 (2003).

15) Chen, M. Y., Kcman, E., Fratkin, E., Fox, A. and Brewer, E.: Pinpoint : Problem Determination in Large , Dynamic Internet Services, *In Proc. 2002 Intl. Conf. on Dependable Systems and Networks*, pp. 595—-604 (2002).

16) Repantis, T. and Kalogeraki, V.: Hot-spot prediction and alleviation in distributed stream processing applications, *2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC (DSN)*, pp. 346–355 (2008).

17) : The /proc File System (2007).

18) : MongoDB: an open-source document database, and the leading NoSQL database, `http://www.mongodb.org`.

付　　録

表 4: 実験環境 ホストマシン性能

プロセッサ	クロック数	OS	カーネル	bit 数
Intel(R) Core(TM) i5-3470 CPU	3.20GHz	CentOS release 6.4	2.6.32-358.18.1.el6.x86_64	64

コア数	メモリ	ATA	HDD	ethnet
4	8GB	WDC WD10EALX-759	1000GB	100baseT

表 5: 実験環境 解析サーバ性能

プロセッサ	クロック数	OS	カーネル	bit 数
Intel(R) Core(TM) i7-3770 CPU	3.40GHz	CentOS release 6.4	2.6.32-358.18.1.el6.x86_64	64

コア数	メモリ	ATA	HDD	ethnet
8	16GB	ST500DM002-1BD14	500GB	100baseT

表 6: 実験環境 仮想マシン性能

仮想マシン	OS	カーネル	bit 数	コア数	メモリ	HDD
VirtualBox-4.2.12r84980	CentOS release 6.4	2.6.32-358.18.1.el6.x86_64	64	3	6GB	500GB

表 7: mapred-site.xml

パラメータ	値	説明
mapreduce.task.io.sort.mb	100	map の出力をソートする際に使用するバッファメモリの上限
mapreduce.task.io.sor.factor	10	reduce の結果をソートする際の, 同時にマージするストリームの数
mapred.map.child.java.opts	-Xmx256m -Xmx1024m	map タスクの Java オプション
mapred.reduce.child.java.opts	-Xms512m -Xmx1526m	reduce タスクの Java オプション
mapreduce.tasktracker.map.tasks.maximum	3	TaskTraceker が同時に起動できる map タスク数の上限
mapreduce.tasktracker.reduce.tasks.maximum	3	TaskTracker が同時に起動できう r reduce タスク数の上限

表 8: hdfs-site.xml

パラメータ	値	説明
dfs.replication	3	データの複製数
dfs.blocksize	64m	HDFS ブロックのサイズ