D-02

Hadoop の性能ボトルネックを特定するための実行トレース可視化ツール

A trace visualizing tool for finding performance bottlenecks in Hadoop

古谷 達朗† 置田 真生† 萩原 兼一†
Tatsuro Furuya Masao Okita Kenichi Hagihara

ABSTRACT

Hadoop は MapReduce プログラミングモデルを実装した並列分散処理のフレームワークである．Hadoop には並列化に必要な処理があらかじめフレームワークに実装されているため，ユーザは計算部分のみをユーザ定義関数として実装するだけでよい．

しかし，Hadoop プログラムの性能改善において，ユーザはユーザ定義関数のみでなくフレームワークの動作を理解する必要がある．この際，Hadoop のログは記録される情報が膨大な上，フレームワークの動作理解に必要な情報が不足しているため性能改善に不十分である．

本研究は Hadoop プログラムの性能改善の支援を目的とする．ユーザ定義関数およびフレームワークの実行時情報を取得し，プログラムの実行状況を可視化するツールを作成する．適用実験の結果，本ツールはフレームワークの動作であるデータ入出力および負荷分散における性能ボトルネックの特定を支援できることを示した．

1． はじめに

近年，データの大規模化に伴い，多数の計算機を用いた並列分散処理の需要が高まっている．大規模な並列分散処理の実装を容易にする技術として，Google 社が提唱する MapReduce プログラミングモデル[1] がある．その代表的なオープンソース実装 Hadoop[2]は広く普及している．

Hadoop は，MapReduce に特化したマスタ・ワーカ型のフレームワーク（以降，FW）を提供する．MapReduce では，1 つのジョブは複数の Map タスク（以降，M）と Reduce タスク（以降，R）から構成される．各タスクをクラスタ上の計算機に割り当て，同時に実行することで並列処理を実現する．

Hadoop におけるプログラム開発は単純である．ユーザは，M および R のそれぞれ主要な計算部分である Map 関数（以降，f_m）および Reduce 関数（以降，f_r）の 2 つだけを実装すれば良い．並列分散処理の実現に必要な機能（入力データの分割，計算機へのタスクの割り当て，タスク間のデータ通信）は全て FW が提供するため，ユーザが FW の詳細な動作を知る必要はない．

しかし，Hadoop プログラムの性能改善は煩雑である．性能改善において，ユーザはまずボトルネックを特定し，次にその原因を解消する．ボトルネック特定における問題は次の 2 つである．

- FW の動作原理を把握する必要がある
- Hadoop のログの有用性が低い

まず，ユーザは開発段階では不要であった FW の動作原理を理解する必要がある．例えば，タスクの実行スケジュールによっては，データ転送の待ち時間が増大してボトルネックとなる場合がある．このとき，一見して R の処理時間が増大するため，FW に関する十分な知識を持たないユーザは f_r ばかりに注目してしまい，ボトルネックを発見できない恐れがある．したがって，性能改善にあたっては FW の動作の理解が不可欠であるが，このために開発段階と比較してユーザの負担が増大する．

次に，ボトルネックの特定を目的とした場合，Hadoop が提供するログ（以降，単にログ）は不十分である．ログはタスク単位での実行時情報を記録したテキストデータである．大規模なプログラムの実行時には記録される情報が膨大なため，その中からボトルネックを特定することは非常に煩雑である．また，タスク単位の記録であるため，より詳細な原因の特定は難しい．例えば，処理時間の長大な R があった場合に，データ転送の待ち時間と f_r の処理時間のどちらが原因かを判別できない．

一方，ボトルネックの原因の解消にも FW の動作原理の理解が必要である．ボトルネックの原因が FW にあった場合，ユーザは FW のソースコードを変更できないため，FW に与えるパラメータを変更することで FW の動作を間接的に制御する．FW に関する十分な知識を持たないユーザにとって，各パラメータを変更した場合の動作の予測は難しい．

そこで我々は Hadoop プログラムの性能改善を半自動化により支援する研究に取り組んでいる．この研究の全体構想は，FW に関する十分な知識を持たないユーザを対象に，(1)性能ボトルネックの特定に要する負担を軽減し，(2)性能改善のための適切なパラメータを推薦することである．

本稿では，(1)の目的のために開発した，Hadoop の実行状況を可視化するツールについて報告する．本ツールは，ログよりも詳細な実行時情報（以降，トレース）をプログラムの実行時に自動的に取得する．このトレースに基づいてガントチャートを表示することで，FW の動作の直観的な理解を支援する．また，ユーザ定義関数（f_m および f_r）と FW が提供する機能の実行時情報を区別して提供することで，ボトルネックの詳細な原因の特定を支援する．

本稿の構成は以下の通りである．まず，2 章で Hadoop について説明する． 次に，3 章で Hadoop のふるまいをモデル化する．4 章で性能解析における問題について述べ，5 章で可視化ツールを設計する．続く 6 章でツールを実装し，7 章で適用実験の結果について述べる．その後，8 章で関連研究を紹介する．最後に，9 章で本稿をまとめる．

†大阪大学　大学院情報科学研究科
Graduated school of Information Science and Technology, Osaka University

表 1　タスクを構成する関数

	Map タスク	Reduce タスク
ユーザ定義	Map 関数	Reduce 関数
FW 提供	Sort 関数 Output 関数 Merge 関数	Copy 関数 Sort 関数

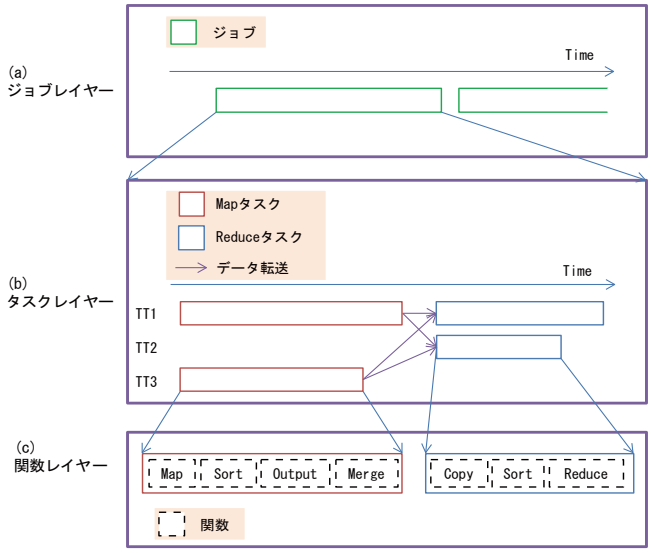

図 1　Hadoop のふるまいの模式図

2．Hadoop

プログラムを実行する際，ユーザは以下の 3 つの要素を定義する．

（a）f_m と f_r の組として実装した MapReduce プログラム
（b）FW の動作を制御するパラメータ
（c）（a）に対する入力データ

以下に，FW の動作およびそれにおけるユーザ定義の要素の役割を説明する．

Hadoop が（a）を開始すると，マスタであるジョブトラッカ（以降，JT）は（c）を（b）で定めたサイズに等分し，ワーカであるタスクトラッカ（以降，TT）へ均等に分配する．これらの入力データ片をブロックと呼ぶ． JT はブロックごとに 1 つの M を TT に割り当てる．M は f_m を実行し，（b）で定めた R の数と同数の出力ファイルを生成する．いずれか 1 つの M が完了した際，JT は R を均等に TT へ割り当てる．完了した M は出力ファイルを全ての R に 1 つずつ送信する．全ての M が出力ファイルを送信した後，R は f_r を実行する．全ての R が完了した際，Hadoop は（a）を終了する．

3．Hadoop のふるまいのモデル化

本研究では FW の動作を理解するため，Hadoop のふるまいをモデル化する．

我々は Hadoop のふるまいを 3 階層のレイヤーで解釈する．まず，最上層はジョブに着目したレイヤーである．Hadoop のふるまいをジョブの集合と定義し，各ジョブを開始時刻と終了時刻で表す（図 1（a））．次は，タスクに着目したレイヤーである．1 つのジョブを M および R の集合と定義し，各タスクを開始時刻，終了時刻および実行 TT で表す（図 1（b））．最後は，関数に着目したレイヤーである．1 つのタスクを表 1 に示す関数の集合と定義し，各関数を開始時刻および終了時刻で表す（図 1（c））．

4．Hadoop プログラムの性能解析における問題

4.1　FW の動作原理の理解が必要なパターン

Hadoop では，FW が暗黙的に実行する処理がボトルネックとなる場合がある．そのうち，本稿では次の 3 つのパターンを想定する．

1 つ目は，M の出力に伴う処理である（**C1**）．Hadoop は f_m の出力をメモリにバッファし，一定のサイズを超えるごとにファイルへ出力する．その後，Merge 関数はこれらのファイルを R と同数になるまでマージする．したがって，f_m の出力データサイズが大きい場合，出力ファイルの数が増加するため Merge 関数の処理時間が増加する．その結果，M の実行時間が増加する．

2 つ目は，R の入力に伴う処理である（**C2**）． R は M の出力ファイルを集める際，ファイル数と同じ回数だけ Copy 関数を実行する．この際，全ての Copy 関数は同時に開始せず，M が完了するごとに開始する．したがって，時間を要する M が存在する場合，Copy 関数の開始を待機するため R の実行時間が増加する．

3 つ目は，JT のタスクスケジューリングである（**C3**）．Hadoop では全ての M が完了した後，R が一斉に開始する．ゆえに，割り当てられた M を完了する時刻が全ての TT において同じであることが理想である．しかし，TT ごとに割り当てられるタスク数や入力データが異なるため，負荷が不均等になり他の TT よりタスクの完了に時間を要する TT が発生する．その結果，ジョブの実行時間が増加する．

これらのボトルネックを特定するため，（C1）および（C2）の場合，ユーザはユーザ定義の関数のみでなく FW が提供する関数の仕様を理解する必要がある．また，（C3）の場合，ユーザは個々のタスクのみでなく全てのタスクの実行手順を把握する必要がある．

4.2　ログを用いた性能解析における問題

ユーザがログを用いてプログラムの性能を解析する際，問題が 2 つある．

1 つ目は，Hadoop がログに記録する情報の粒度が粗い問題（**P1**）である．（C1）の場合，Hadoop は M ごとに開始時刻，終了時刻およびタスクの進捗をログファイルに記録する．しかし，Hadoop は f_m および Merge 関数の開始時刻と終了時刻をログファイルに記録しないため，ユーザは f_m と Merge 関数のいずれが M の実行時間を決定するかをログから判断できない．

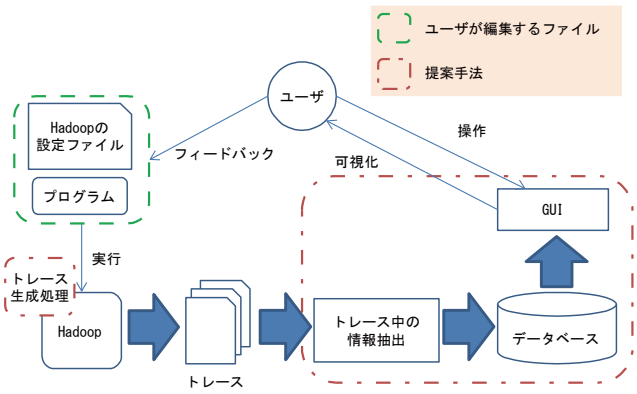

図 2　本ツールを用いた性能解析の手順

同様に，（C2）の場合，Hadoop は *M* の出力ファイルの転送において，全体の進捗を（転送済みファイル数）/（転送予定のファイル数）という形式でログファイルに記録する．しかし， Copy 関数ごとの開始時刻および終了時刻をログファイルに記録しないため，（C1）と同様にユーザは転送時間に対する転送待ち時間の割合や Copy 関数ごとの実行時間をログから把握できない．

2 つ目は，Hadoop がログに記録する情報の量が多い問題（**P2**）である．（C3）の場合，ユーザはログからタスクごとの開始時刻，終了時刻および実行 TT を取得する．ただし，11GB のファイルを入力としてサンプルプログラムの WordCount[2]を実行した場合，Hadoop は総計して約 85000 行，12MB のログを出力する．ゆえに，ユーザが目的の情報を特定し，FW の動作を再現するための労力が大きい．

5．可視化ツールの設計

本研究は，3 章のモデルに基づいて Hadoop のふるまいを可視化するツールを開発した．

本章では，以下の 3 点について設計方針を述べる．

- 取得する情報
- 情報の可視化方法
- 情報の取得方法

5.1　取得する情報

（P1）を解決するため，4.1 節の全パターンのボトルネック特定に必要な情報を取得する．

まず，（C1）および（C2）において表 1 の関数の開始時刻および終了時刻が必要である．さらに，関数の実行に時間を要する原因を特定するための詳細な情報として，以下の情報を取得する．

- 入力レコード数
- 入力キー数
- 出力レコード数
- 転送データサイズ
- データ転送元の TT

次に，（C3）において各タスクの開始時刻，終了時刻および実行 TT が必要である．

5.2　情報の可視化方法

（P2）を解決するため，トレースから得た情報を視覚的にユーザに提供する．この際，3 章のモデルに基づいて 3 階層のレイヤーを表示する．

まず，ジョブレイヤーとして表形式で実行したジョブの一覧をユーザに示す．次に，タスクレイヤーとして縦軸をタスク ID，横軸を時刻とするガントチャートを示す．最後に，関数のレイヤーとして縦軸をスレッド，横軸を時刻とするガントチャート形式でタスクの実行時間の内訳を示す．

ただし，大規模なプログラムではタスク数および関数の実行回数が増加するため，目的の情報を特定する際労力を要する．そこで，提案手法ではソート機能および絞り込み機能を提供する．ソート機能はタスクレイヤーおよび関数レイヤーにおいて，任意の項目についてソートする．絞り込み機能はタスクレイヤーにおいて，ガントチャートで提示するタスクの数を削減するため，FW が実行を中断したタスクや *M* または *R* の一方を除外する．

5.3　情報の取得方法

まず，5.1 節に挙げた情報を取得するため，提案手法では Hadoop ソースコード中にトレース生成処理を追加する．

次に，取得した情報の管理方法を説明する．ボトルネックの解析において，ユーザは実行時間のみでなく実行時間を決定する要因を調べる．例えば，Copy 関数の実行時間は転送データサイズにより決まる．したがって，関係する複数の情報をまとめて管理する方法が必要である． そこで，提案手法では関係データベースを用いる．(1)ジョブの情報を管理する表，(2)タスクの情報を管理する表および(3)タスク内の関数の情報を管理する表を作成する．

最後に，トレースの解析と可視化のタイミングを説明する．ボトルネックを解析する際，ツールの動作によりプログラムの実行性能が変化することは望ましくない．リアルタイムに解析する場合，Hadoop とツールが同時に動作する．ツールはサーバとクライアント間の通信に資源を要求するため，解析しない場合と比べ Hadoop の実行性能が低下する．一方，プログラムの実行後に解析する場合，実行性能の低下を抑制できる．そこで，提案手法はプログラムの実行時においてファイルへのトレースの記録のみを行う．ツールはプログラムの実行後にトレースから情報を抽出し，データベースに保存する．その後，情報を可視化してユーザに提示する．

5.4　本ツールを用いた性能解析の手順

本ツールを用いた性能解析の手順を図 2 に示す．まず，ユーザはトレース生成処理を追加した Hadoop を用いてジョブを実行する．その結果，Hadoop はトレースをファイルに記録する．次に，本ツールはトレースからボトルネック解析に必要な情報のみを抽出し，データベースに保管する．最後に，本ツールは GUI を用いてデータベースの情報をユーザに提示する． ユーザは提示された情報に基づいてプログラムおよび設定ファイルのパラメータを改善する．

表 2　トレース生成メソッドの呼び出しを追加するソースファイル

	ファイル名
1	JobTracker.java
2	JobInProgress.java
3	MapTask.java
4	ReduceTask.java

表 3　開始時刻と終了時刻を取得するメソッド

	クラス名	メソッド名	関数との対応
1	MapTask	runOldMapper	Map 関数
2	MapTask	runNewMapper	
3	MapOutputBuffer	sortAndSpill	Sort 関数（Map）
4	MapOutputBuffer	mergeParts	Merge 関数
5	MapOutputCopier	shuffleInMemory	Copy 関数
6	MapOutputCopier	shuffleToDisk	
7	ReduceTask	merge	Sort 関数（Reduce）
8	ReduceTask	runOldReducer	Reduce 関数
9	ReduceTask	runNewReducer	

6．可視化ツールの実装

本章では，特に断りがない限りオリジナルとは提案手法適用前の Hadoop を指すものとする．オリジナルのバージョンは 1.0.0 である．

6.1　トレースの取得

6.1.1　トレース生成処理の実装

オリジナルは Log4J[3]のログ生成メソッドを用いてログを生成する．ログ生成メソッドは呼び出された時刻および引数として与えられた文字列を含むログをファイルに出力する．

Log4J オブジェクトを生成する際，引数として Java のクラス名を与える．ログ生成メソッドが呼び出されると，Log4J オブジェクトは log4j.properties におけるクラスまたはパッケージごとの設定を参照し，ログの出力先を決定する．したがって，オリジナルに存在しないクラスをオブジェクトの引数に与えることで，オリジナルとは出力先が異なるログ生成メソッドを用いることができる．

そこで，提案手法ではオリジナルに存在しないクラス CustomLog を作成する．CustomLog は宣言のみで処理が存在しないクラスである．その後， CustomLog を引数に与えた Log4J オブジェクトを生成する．

以降では，トレース生成メソッドとは提案手法で生成した Log4J オブジェクトのログ生成メソッドを指すものとする．

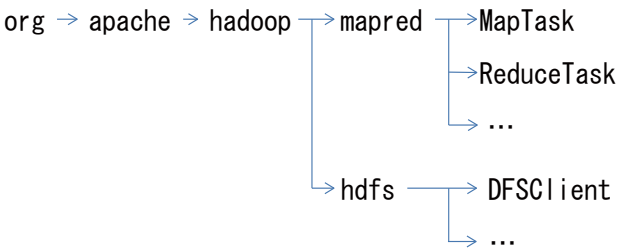

図 3　Hadoop におけるパッケージの階層

```
CREATE TABLE jobs (ind decimal,
job_id text,              //ジョブID
job_start decimal,        //ジョブの開始時刻
job_finish decimal,       //同上の終了時刻
job_input decimal,        //入力データサイズ
map_num decimal,          //Mapタスク数
reduce_num decimal,       //Reduceタスク数
status text)              //ジョブの実行状態
```

図 4　データベースの表 jobs の構造

```
CREATE TABLE tasks (ind decimal,
task_id text,                //タスクID
node text,                   //実行TT
start_time decimal,          //タスクの開始時刻
finish_time decimal,         //同上の終了時刻
map_start decimal,           //Map関数の開始時刻
map_finish decimal,          //同上の終了時刻
reduce_start decimal,        //Reduce関数の開始時刻
reduce_finish decimal,       //同上の終了時刻
reduce_sort_start decimal,   //ReduceタスクのSort関数の開始時刻
reduce_sort_finish decimal,  //同上の終了時刻
map_merge_start decimal,     //Merge関数の開始時刻
map_merge_finish decimal,    //同上の終了時刻
task_type text,              //MapまたはReduce
job_id text,                 //ジョブID
status text,                 //タスクの実行状態
input_record decimal,        //入力レコード数
input_group decimal,         //入力キー数
output_record decimal)       //出力レコード数
```

図 5　データベースの表 tasks の構造

```
CREATE TABLE details (ind decimal,
task_id text,                 //タスクID
map_sort_start decimal,       //MapタスクのSort関数の開始時刻
map_sort_finish decimal,      //同上の終了時刻
map_output_start decimal,     //Output関数の開始時刻
map_output_finish decimal,    //同上の終了時刻
reduce_copy_start decimal,    //Copy関数の開始時刻
reduce_copy_finish decimal,   //同上の終了時刻
reduce_copy_from text,        //同上の転送元のTT
reduce_copy_size decimal)     //同上の転送データサイズ
```

図 6　データベースの表 details の構造

6.1.2　トレース生成メソッドの呼び出しの追加

提案手法において，トレース生成メソッドの呼び出しを追加するソースファイルを表 2 に示す． 各ソースファイルにおいて，ジョブ，タスクおよび表 3 に示すメソッドの開始直前および終了直後にトレース生成メソッドの呼び出しを追加する．

job ID	run time[ms]	map	reduce	input size[byte]
201207041425_0001	74,229	175	108	11,692,307,690
201207041425_0002	70,061	175	108	11,692,307,690
201207041425_0003	70,170	175	108	11,692,307,690
201207041425_0004	70,290	175	108	11,692,307,690
201207041425_0005	72,213	175	108	11,692,307,690

図 7　ジョブレイヤーの表示

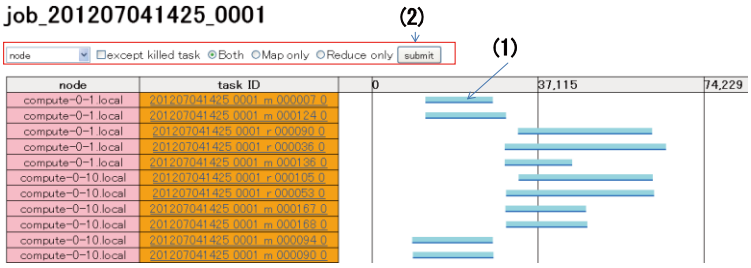

図 8　タスクレイヤーの表示

6.1.3　トレース生成メソッドの出力先の設定

Hadoop はタスクごとに異なるログファイルを生成するため，log4j.properties において環境変数を用いて出力先を指定し，タスクの開始時に環境変数へ出力先を設定する．log4j.properties において静的に出力先を指定した場合，Log4J は全てのタスクのトレースを 1 つのファイルに記録するため，特定のタスクの情報を抽出する際時間を要する．

そこで，提案手法では log4j.properties においてトレース生成メソッドの出力先としてオリジナルに存在しない環境変数を指定する． hadoop および hadoop-daemon.sh は，表 2 における 1 と 2 のログの出力先を設定する．TaskLog クラスは，表 2 における 3 と 4 のログの出力先を設定する．これらのファイルに環境変数の設定を記述することで，ログの出力先を動的に決定する．

6.1.4　トレース生成メソッドの二重出力の抑制

log4j.properties では下位のパッケージが上位の設定を継承する．図 3 に示す Hadoop のパッケージの階層において，hadoop で A，mapred で B，MapTask で C という出力ファイルを指定した場合，MapTask のログは A，B および C に出力される．さらに，Log4J には全てのパッケージの上位として rootlogger が存在する．rootlogger においてオリジナルの出力先が指定されているため，追加したトレース生成メソッドはトレースを新しいファイルおよびオリジナルのログファイルに重複して出力してしまう．

そこで，提案手法では rootlogger における設定を削除し，CustomLog を除く下位のパッケージ全てに元の rootlogger の内容をコピーする．その結果，トレース生成メソッドは新しいファイルにのみトレースを記録する．

6.2　トレースから得られる情報の管理

関係データベースを用いて，ファイルに記録されたトレースから取得した情報を管理する．データベースを操作する SQL として，SQLite3.5.4 を用いる．

6.2.1　データベースの表の作成

提案手法では 3 つのデータベースの表 jobs，tasks および details を作成する．まず，jobs は図 4 に示すようにジョブレイヤーで提示する情報を保持する．次に，tasks は図 5 に示すようにタスクレイヤーで提示する情報，および関数レイヤーで提示する情報のうちタスクごとに一度のみ実行される関数の情報を保持する．最後に，details は図 6 に示すように関数レイヤーで提示する情報のうち，タスクごとに複数回実行される関数の情報を保持する．

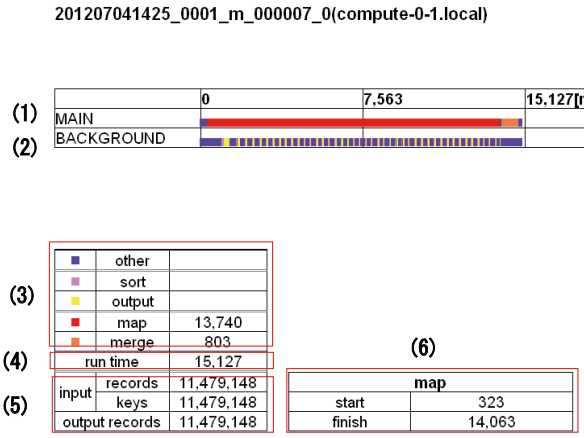

図 9　関数レイヤーの表示

6.2.2　ファイルからの情報の抽出

提案手法ではパターンマッチングを用いて，ファイルに記録されたトレースから性能解析に必要な値を抽出する．

その後，抽出した値を表に挿入する． SQLite では.import コマンドを実行することにより，CSV 形式のファイルの各行を 1 つのデータとして表に挿入できる．.import コマンドは，データ 1 つごとに insert コマンドを実行するよりも高速である．ゆえに，提案手法では抽出した値を所属する表ごとに分類し CSV 形式で出力する．

6.3　トレース中の情報の可視化

Ruby on Rails を用いて，データベースの情報を可視化するウェブ UI を作成する．バージョンは Ruby1.8.6 および Rails2.2.2 である．

6.3.1　可視化のレイヤー

提案手法のウェブ UI は，3 章で示した 3 つの可視化のレイヤーを持つ．Hadoop プログラムの性能解析において，ユーザは最初にジョブレイヤーを参照する．

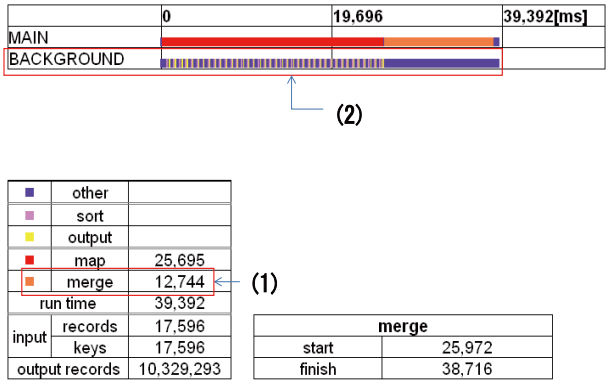

図 10　Dijkstra における Map タスクの実行時間の内訳

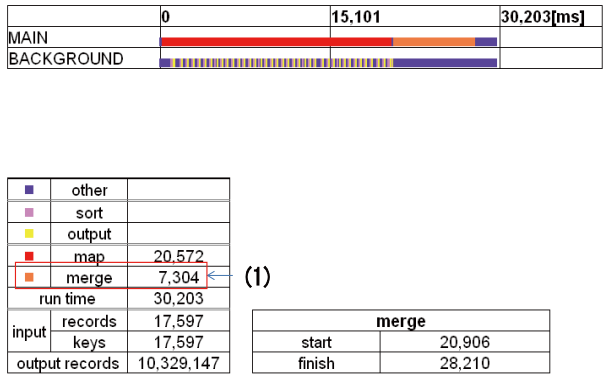

図 11　パラメータ変更後の Dijkstra における Map タスクの実行時間の内訳

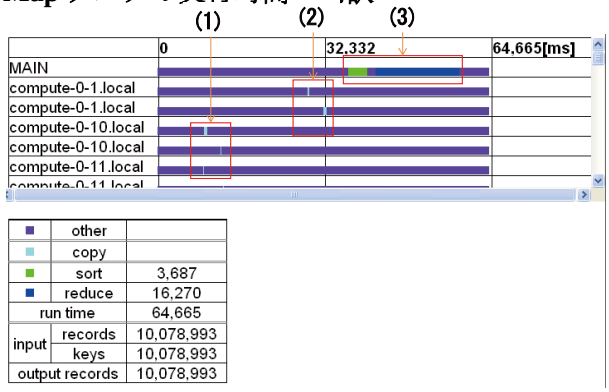

図 12　TeraSort における Reduce タスクの実行時間の内訳

まず，図 7 に jobs の情報に基づくジョブレイヤーの表示を示す．ウェブ UI は表の形式でジョブ ID，ジョブの実行時間，*M* の数，*R* の数，および入力データサイズをユーザに提示する． ユーザは解析するジョブのジョブ ID をクリックすることにより，2 つ目のレイヤーへ移動する．

次に，図 8 に tasks の情報に基づくタスクレイヤーの表示を示す．ウェブ UI は縦軸がタスク ID，横軸が時刻のガントチャート形式でタスクスケジューリングの様子をユーザに提示する． (1) はタスクの実行時間を示す．ユーザは (2) のプルダウンボックス，チェックボックスおよびラジオボタンを用いて条件を指定し，submit ボタンをクリックしてレーンのソートおよび絞り込みを行う．ユーザは解析するタスクのタスク ID または (1) をクリックすることにより，3 つ目のレイヤーへ移動する．

表 4　適用実験で用いるクラスタ環境

ノード数	54
CPU	Intel Xeon 3.2GHz×8
RAM	80GB
ネットワーク	Gigabit Ethernet
R の数	10

最後に，図 9 に tasks および details の情報に基づく *M* の関数レイヤーの表示を示す．*M* は f_m と Merge 関数を実行するメインのスレッドと， Sort 関数と Output 関数を実行するバックグラウンドのスレッドを持つ．*M* は複数の関数を並列に実行するため，(1) と (2) のように横軸が時刻であるレーンをスレッドごとに与え，重なり合わないようにする． (3) は凡例と関数ごとの実行時間を示し， (4) はタスクの実行時間を示す． (5) は入出力のレコード数およびキー数を示す．各レーンにおいてユーザが色付きの部分をクリックすると，ウェブ UI は (6) のようにその関数の開始時刻および終了時刻を示す．

6.3.2　ソート機能と絞り込み機能

タスクレイヤーおよび *R* の関数レイヤーにおいて，ウェブ UI はソート機能および絞り込み機能を提供する．

ユーザがプルダウンボックスを用いてソートする値を指定すると，ウェブ UI は問い合わせに ORDER BY 句を追加する．その結果，データは指定された値について昇順にソートされる．また，ユーザがチェックボックスまたはラジオボタンを用いて条件を指定すると，ウェブ UI は問い合わせに WHERE 句を追加する．その結果，指定した条件に合致するデータのみが表示される．

7．適用実験

本ツールを用いて，3.1 節で示したパターン（C1），（C2）および（C3）のそれぞれについて，実際にプログラムのボトルネックを解析した例を示す．

7.1　実験環境

本実験で用いるクラスタ環境を表 4 に示す．

解析対象のプログラムとして Dijkstra と TeraSort を用いる．Dijkstra はグラフの最短経路を探索するアルゴリズムの 1 つである Dijkstra 法を実装したプログラムである．Dijkstra の入力として，頂点数が 200000，エッジ数が完全グラフの 0.3%であるループが無い有向グラフを用いる．そのファイルサイズは 0.76GB である．

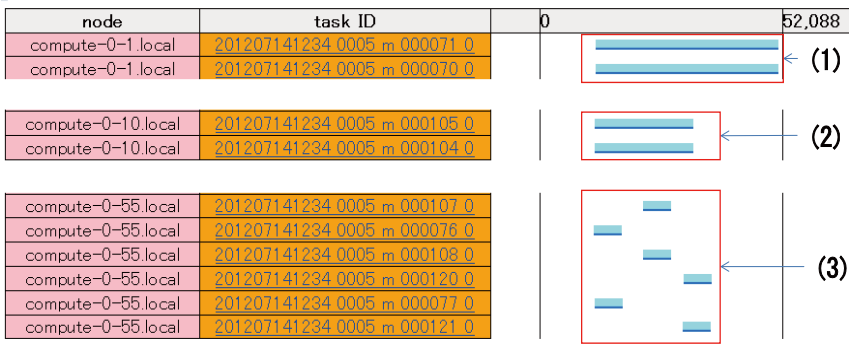

図 13　TeraSort におけるタスクスケジューリングの様子の一部

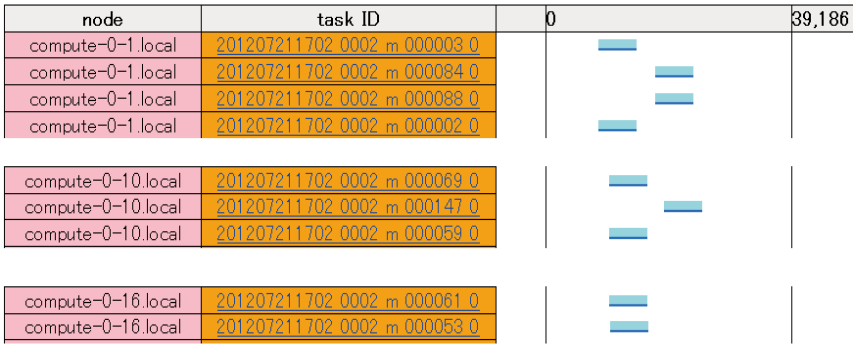

図 14　パラメータ変更後の TeraSort におけるタスクスケジューリングの様子の一部

TeraSort は文字列をソートする Hadoop のサンプルプログラム[2]である．TeraSort の入力として，同じく Hadoop のサンプルプログラムである TeraGen を用いて生成した 10GB のファイルを用いる．

7.2　Map タスクのボトルネック解析

本ツールを用いて，（C1）のパターンにおけるボトルネックを特定する．Dijkstra の M では， Merge 関数がタスク実行の約 3 分の 1 の時間を要する（図 10（1））．Merge 関数はパラメータで指定した数のファイルを 1 つにマージする処理を反復実行し，Output 関数の出力ファイルを Reduce タスクと同数になるまでマージする．したがって，Dijkstra では Output 関数を頻繁に実行する（図 10（2））ため，マージ処理の反復回数が増加し Merge 関数の実行に時間を要すると予測できる．

ボトルネックの原因を特定したため，パラメータの変更による改善を試みた．Merge 関数におけるマージ処理の反復回数を減少させるため， Merge 関数が一度にマージするファイル数（設定ファイル mapred-site.xml の io.sort.factor）を 10 から 500 に増加させる．その結果，図 11（1）に示すように Merge 関数の実行時間が約 5 秒減少した． さらに，ジョブの実行時間は約 4 秒減少した．

7.3　Reduce タスクのボトルネック解析

本ツールを用いて，（C2）のパターンにおけるボトルネックを解析する． TeraSort における R の実行時間の内訳を図 12 に示す．R は Copy 関数の実行ごとにスレッドを生成するため， MAIN がメインのスレッド，他のレーンが転送元の TT ごとの Copy 関数のスレッドを示す．図 12 には，開始時刻が早い Copy 関数（1）と開始時刻が遅い Copy 関数（2）が存在する．最も早い関数はタスク開始から約 5 秒後に開始する．一方，最も遅い関数はタスク開始から約 34 秒後に開始する．（3）のように Sort 関数および f_r は全ての Copy 関数が完了した後に開始するため，最後に完了する Copy 関数が R の実行時間を決定する．Copy 関数は M が完了するたびに実行されるため，TeraSort では M の完了時刻に差があると予測できる．

次節において M の完了時刻に差が生じた原因を調査する．

7.4　タスクスケジューリングのボトルネック解析

本ツールを用いて，（C3）のパターンにおけるボトルネックを解析する．TeraSort のタスクレイヤーにおける M の実行の様子を図 13 に示す．（2）および（3）の M と比較して，（1）の M が実行に時間を要する．

図 13（1）の実行に時間を要する原因を考察する． JT は M を割り当てる際，未だに M を割り当てられていないブロックが存在する TT に割り当てる．しかし，他のタスクを実行しているためその TT に割り当てられない場合，JT は他の TT に M を割り当てる．その後，割り当てられた M はネットワーク経由でブロックを読み込むため，データ転送により実行時間が増加する．

ボトルネックの原因を特定したため，パラメータの変更による改善を試みた．*M* が必ずブロックを持つ TT に割り当てられるようにするため，各ブロックの複製の数（設定ファイル hdfs-site.xml の dfs.replication）を 1 から 3 に増加させる．同一の内容のブロックはそれぞれ異なる TT に配置される．その結果，図 14 に示すように *M* の実行時間が均等になり，各 TT におけるタスクの完了時刻の差が減少した．さらに，ジョブの実行時間は約 26 秒減少した．

8． 関連研究

まず，Hadoop プログラムの実行状況をリアルタイムに可視化するツールとして MR-Scope[4] がある．このツールは Hadoop のワーカがマスタへ送るハートビートを検知した際，ハートビートと共に送られるタスクの進捗，実行 TT およびデータ転送の情報をサーバに収集する．その後，サーバは GUI を用いて収集した情報をユーザに提示する．MR-Scope は各 TT におけるタスクの進捗，および HDFS の情報を示すことを目的とする．しかし，MR-Scope は表 1 の関数の情報を提示しない．

次に，Konwinski[5]らはネットワークシステム向けのリアルタイム解析ツールである X-Trace[6] を，Hadoop プログラムの性能解析に用いた．Konwinski らはタスクや関数の開始および終了をイベントと定義し，イベントの発生位置に X-Trace のトレース生成メソッドを追加する．クライアントはトレースに加え，タスク ID のようなメタデータをサーバに送る．サーバは集めたメタデータとトレースをデータベースに保存し，ウェブ UI を用いて情報をユーザに提示する．Konwinski らはタスク全ての統計的な情報および個々のタスクにおける表 1 の関数の情報を示す．しかし，タスクスケジューリングの様子を提示しない．

最後に，ログを用いて Hadoop プログラムの性能を解析するツールとして Mochi[7]がある．このツールは既存の性能解析ツールである SALSA[8]を拡張したもので，タスクの実行時間，実行 TT および入出力データサイズを相互に関連付けて Hadoop プログラムの実行状況を可視化する．しかし，ユーザプログラムおよび Hadoop のソースコードに解析用のコードを追加しないため， Mochi は表 1 の関数の情報を提示しない．

これらの既存研究と比較して，本研究の特徴はタスクスケジューリングの様子および表 1 の関数の情報の両方を取得し，かつそれらを関連付けて提示する点である．

9． まとめ

本研究は Hadoop プログラムの性能改善を支援するため，ユーザ定義の処理および FW の動作の情報を自動的に取得し，プログラムの実行状況を可視化するツールを開発した．

Hadoop では FW の動作がボトルネックとなる場合があるため，ユーザは FW の動作を理解する必要がある．ただし，Hadoop のログは記録される情報が膨大な上，FW の動作の情報が不足しているため性能改善に不十分である．

提案手法では， まず Hadoop のふるまいをモデル化し，性能解析に必要な情報を定義した．次に， Hadoop のソースコードにトレース生成処理を追加し，性能解析に必要な情報を取得した．最後に，Hadoop のふるまいを 3 階層のレイヤーで可視化してユーザに提示した．

適用実験の結果，本ツールは Map 関数の出力処理，Reduce 関数の入力処理およびタスクスケジューリングにおけるボトルネックの解析および改善に役立つことを示した．

今後の課題は，性能改善のための適切なパラメータの推薦である．まず，我々は各パラメータの値と FW の動作の関係を実験的に調査する．次に，ボトルネックを自動的に特定する実装を目指す．その後，ボトルネックの原因となる FW の動作を制御するための適切なパラメータの値をユーザに提示する手法を検討する．

10． 謝辞

本研究の一部は科学研究費補助金（基盤研究（B）23300007 および 若手（B）23700036）の支援による．

日頃ご議論頂く，大阪大学大学院情報科学研究科萩原研究室の伊野准教授，院生の井上佑希氏，黒松信行氏，はじめ研究室の諸氏に感謝する．

11． 参考文献

[1] Jeffrey Dean and Sanjay Ghemawat.: MapReduce: Simplified Data Processing on Large Clusters, Communications of the ACM - 50th anniversary issue: 1958 - 2008, Vol.51, No.1, pp.107-113, 2008.

[2] The Apache Software Foundation.: Apache Hadoop, http://hadoop.apache.org/.

[3] Apache Logging Services Project.: Apache log4j, http://logging.apache.org/log4j/.

[4] Dachuan Huang, Xuanhua Shi, Shadi Ibrahim, Lu Lu, Hongzhang Liu, Song Wu and Hai Jin.: MR-scope: a real-time tracing tool for MapReduce, In Proceedings of High-Performance Parallel and Distributed Computing (HPDC'10), pp.849-855, ACM, Chicago, IL, USA, 2010.

[5] R. A. D. S. Laboratory.: Projects/Monitoring Hadoop through Tracing, http://radlab.cs.berkeley.edu/wiki/Projects/X-Trace_on_Hadoop.

[6] Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker and Ion Stoica.: X-Trace: A Pervasive Network Tracing Framework, In Proceedings of the 4th USENIX conference on Networked systems design & implementation (NSDI'07), pp.20-20, USENIX Association, Cambridge, MA, USA, 2007.

[7] Jiaqi Tan, Xinghao Pan, Soila Kavulya, Rajeev Gandhi and Priya Narasimhan.: Mochi: Visual Log-Analysis Based Tools for Debugging, In Proceedings of USENIX Workshop on Hot Topics in Cloud Computing (HotCloud'09), Article No.18, USENIX Association, San Diego, CA, USA, 2009.

[8] Jiaqi Tan, Xinghao Pan, Soila Kavulya, Rajeev Gandhi and Priya Narasimhan.: SALSA: Analyzing Logs as StAte Machines, In Proceedings of the First USENIX conference on Analysis of system logs (WASL'08), pp.6-6, USENIX Association, San Diego, CA, USA, 2008.