Vol.2010-DBS-150 No.9
Vol.2010-IFAT-99 No.9
2010/8/4

MapReduceを用いた木構造データのための並列分析処理フレームワーク

柳井孝介[†1]　小林義行[†1]　森本康嗣[†1]

本稿では，木構造データを分析処理するためのフレームワークを提案する．提案するフレームワークでは，垂直分割データ格納方式により木構造データを管理する．即ち，木構造データを属性ごとに分割し，別々のファイルにデータを保存する．これにより属性の数が多い木構造データを扱うことが可能となる．また大規模のデータに対し分析処理を実行可能とするため，MapReduce アーキテクチャに基づく並列処理を行う．評価実験により，属性の追加や属性値の集計のような典型的な分析処理に対して，提案するフレームワークが有用であることを示す．

Parallel Data Analysis Framework for Tree-structured Data using MapReduce

Kohsuke Yanai,[†1] Yoshiyuki Kobayashi[†1]
and Morimoto Yasutsugu[†1]

We propose a parallel data analysis framework for tree-structured data. Our framework implements vertical partitioning, in which tree-structured data is stored in separated files corresponding to each attribute. Handling a large number of attributes becomes feasible by means of vertical partitioning. Additionally, our framework adopts MapReduce architecture for parallel computing in order to process large-scale data. We show our framework is efficient to process typical data analysis, such as appending new attributes and calculating statistics of attributes.

†1 日立製作所 中央研究所
Hitachi, Ltd., Central Research Laboratory

1. はじめに

収集した大量のデータから有益な知識を抽出し，それを業務で活用する技術に注目が集まっている．この背景として，IT 技術の普及により大量データの収集が容易になってきたことと，IT に高付加価値を求める傾向が出てきたことが挙げられる．実際に大量のデータを分析することで，Web 広告の最適化や書籍の推薦を行い高収益をあげて企業がある．これらを成功事例と見て，同様のプロセスを試みる企業も増えてきている．

大規模データ分析には，以下の 2 つの主要技術がある．

(1)　大規模な計算機リソースの利用を容易にするための大規模計算プラットフォーム
(2)　データから知識を抽出するためのデータ分析技術

前者の大規模計算プラットフォームに関しては，仮想化，分散ファイルシステム，MapReduce[3),4)]，キーバリューストア[2)] などの技術に期待が高まっている．仮想化は Xen[7)]，分散ファイルシステムは HDFS（Hadoop Distributed File System）[6)]，MapReduce は Hadoop MapReduce[6)]，キーバリューストアは HBase[6)] と，それぞれオープンソースソフトウェアが開発されており，Web サービス企業などで，これらのオープンソースソフトウェアが実際に業務に活用されている．特に分散処理フレームワークである Hadoop に関しては，Web サービス企業以外も高い関心を示しており，業務への適用を検討している[8)]．一方，データ分析技術に関しては，相関やクロス集計などの統計的な方法に加え，最近では機械学習関連の技術がよく利用されている．

このような背景の中，日立製作所では，大量の実業データと IT リソースを活用し，知識をサービスとして提供する KaaS (Knowledge as a Service)[9)] を提案しており，KaaS 実現のために，大量データから知識を抽出する技術の研究を進めている．本稿では，この KaaS 研究の一環として研究を行った木構造データ向けのデータ分析フレームワークについて報告する．提案するフレームワークでは，大規模計算プラットフォームとして MapReduce を用いる．また分析技術としては機械学習を想定し，機械学習用のデータを生成するための前処理に相当する処理を実行する．

本報告の構成は以下の通りである．まず 2 章で対象とするデータ分析処理について述べる．次に 3 章にて提案するデータ分析フレームワークを説明し，4 章で実験結果を報告する．5 章で考察を述べ，6 章で本報告を締めくくる．

Vol.2010-DBS-150 No.9
Vol.2010-IFAT-99 No.9
2010/8/4

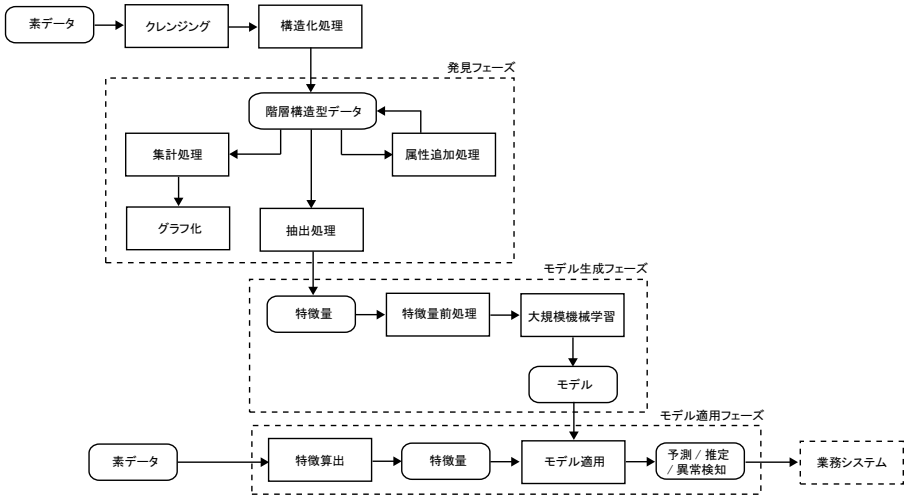

図 1 本研究で想定するデータ分析/活用プロセス

2. 対象とするデータ分析処理

図 1 に想定するデータ分析/活用プロセスの全体像を示す．本報告では，このうち，発見フェーズの分析処理を実行するフレームワークを提案する．データ分析/活用プロセスは素データのクレンジングから始まる．クレンジングとは，データを分析処理に適した形式に変換したり，不正なデータを除去したりする処理のことをいう．次にクレンジングされたデータに対し，構造化処理を実行する．構造化処理では，表形式の素データを XML のような木構造型のデータに変換する．これは分析対象となるデータが本質的に木構造をなすことが多いためである．続いて，発見フェーズ，モデル生成フェーズを経てデータがモデルに変換され，最終的にモデル適用フェーズにて業務に組み込まれる形式で活用される．

前述の通り，本報告では発見フェーズを対象とする．発見フェーズでは，集計分析を試行錯誤で繰り返すことにより，データをどのように活用すれば業務に適用できるかを見出す．ここで集計分析とは，平均，分散，頻度分布，クロス集計などのデータの全体的な傾向を把握するための分析のことを指す．試行錯誤の過程では，既存の属性値を組み合わせて新しい属性値を算出し，元のデータ構造に属性を追加する処理を繰り返す．ここで属性とは RDB（Relational Database）のカラムに相当する．実際，これまでに著者らが行ったデータ分析

```
[ (tag001
   [ (sid001 ave-cpu:84.0%
      [ (2010/02/05;10:10:00 cpu:92% mem:532MB rd:4.03blk/s)
        (2010/02/05;10:20:00 cpu:76% mem:235MB rd:3.32blk/s) ])
     (sid002 ave-cpu:12.6%
      [ (2010/02/05;15:30:00 cpu:13% mem:121MB rd:1.03blk/s)
        (2010/02/05;15:40:00 cpu:15% mem:142MB rd:1.22blk/s)
        (2010/02/05;15:50:00 cpu:10% mem:140MB rd:2.21blk/s) ]) ])
  (tag021
   [ (sid001 ave-cpu:50.0%
      [ (2010/02/05;11:40:00 cpu:88% mem:889MB rd:2.22blk/s)
        (2010/02/05;11:50:00 cpu:12% mem:254MB rd:2.36blk/s) ]) ]) ]
```

図 2 サーバ稼働データの例．tag は設置場所を表すタグ，sid はサーバの ID，ave-cpu はサーバごとの平均 CPU 使用率，t は時刻，cpu は CPU 使用率，mem はメモリ使用量，rd は HDD の読み込み量，wr は HDD の書き込み量を表す．

案件で，30 以上の新しい属性値を追加しており，これらを繰り返し利用して集計分析を実行している．最終的にデータの活用方針が定まると，機械学習で利用される特徴量データを出力する．発見フェーズでは 10GB〜1TB 程度のデータ量を繰り返し処理することを想定している．そのため，発見フェーズの処理は IO に負荷がかかる．

本報告では，発見フェーズの処理を実行時間を短縮し，試行錯誤の回数を増やすことができるようにするため，属性追加処理，集計処理，抽出処理を高速に実行できるフレームワークを提案する．

3. 木構造データ分析フレームワーク

提案するフレームワークでは，各分析処理で共通の処理はフレームワーク側で実装されている．そのためユーザは各分析処理に固有のロジックのみを分析プログラムとして開発だけでよい．またデータ格納方法や分散処理をフレームワーク側で隠蔽するため，ユーザはこれらを意識せずに分析プログラムを開発できる．

3.1 木構造データに対する垂直分割データ格納

大規模データを分析する場合，よく利用されるグループ化（SQL の group by）やデータ結合（SQL の join）は実行速度が極めて遅い．よって本研究では，データを正規化して複数の表で管理することはせず，図 2 に示すように木構造のまま管理するアプローチをとる．これにより，グループ化やデータ結合の処理が発生することなく，分析処理を実行できる．

また属性追加処理の繰り返しにより属性の数が増えた場合でも，分析処理の実行速度が低

Vol.2010-DBS-150 No.9
Vol.2010-IFAT-99 No.9
2010/8/4

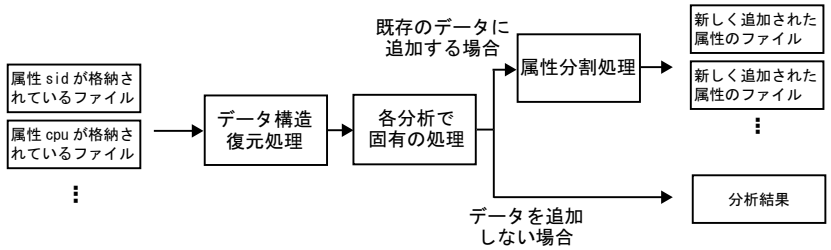

図 3 データ分析処理時のフロー.

下しないようにするため，木構造データを属性ごとに分割してファイルに保存する．これにより，分析処理に必要な属性のみ入出力することができるため，属性数が増えても IO の量は変化しない．テーブル形式のデータに対して，属性ごとに分割して管理する方式が有効であることはよく知られている[5] ⋆1．本研究では，グループ化されたままのデータ，すわなち木階層構造型データに対して，属性ごとに分割する方式をとる．

提案するフレームワークでは，以下で再帰的に定義されるデータを処理の対象とする：

属性値: スカラ，ベクトル，行列，文字列などの値．例："532MB"

タプル: 複数の属性値またはリストの組み合わせ．例：("sid002" [13 15 10])

リスト: 同じ型を持つタプルの繰り返し．例：[("sid001" 84.0) ("sid002" 12.6)]

タプルは小括弧 ()，リストは大括弧 [] で表し，要素は空白区切りで表記する．リストの要素は再帰的に同じ型を持たなければならない．上記データモデルは，一般的な木構造データに比べ型の制限を厳しくすることにより，属性ごとに分割する処理が可能となる．

図 3 にデータ分析処理時のフローを示す．まずデータ構造復元処理により，分析処理に必要な属性を各ファイルから読み込み，木構造を復元する．次に，各分析処理で固有の処理を実行する．続いて，属性追加処理など，既存のデータに新しいデータを追加する場合には，再度，木構造のデータを属性ごとに分割して，データをファイルに保存する．データ構造復元および属性分割のアルゴリズムに関しては，アルゴリズムの詳細の説明が煩雑になり，かつ本報告の論点からはずれるため省略する．アルゴリズムの実装には，補助関数がいくつか必要であるが，ロジックのコアの部分の実装は，両方とも Lisp で 7 行程度である．

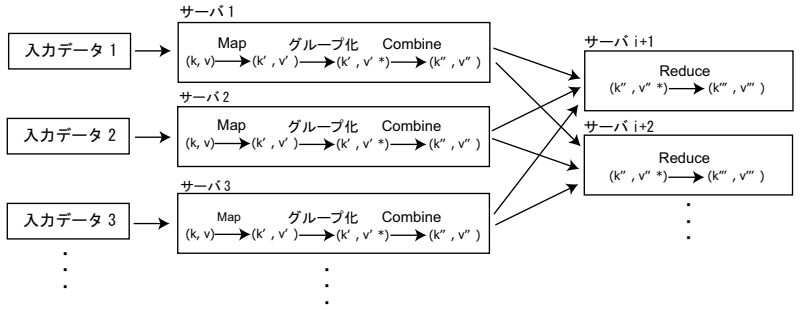

図 4 MapReduce 処理の流れ.

3.2 MapReduce による分散処理

MapReduce は，Map，Combine，Reduce の 3 つの処理フェーズにより分散計算を行う分散計算アーキテクチャである[3),4)]．MapReduce では，キーとバリューのペアの集合を処理対象のデータとする．以下では Hadoop の計算モデルを簡単に説明する．詳細は文献 3), 4) を参照されたい．

キーとバリューのペアを (k_i, v_i) と書く．Map フェーズでは以下の処理を行う．

$$(k_1, v_1), (k_2, v_2), \cdots \longrightarrow f_M(k_1, v_1) \cup f_M(k_2, v_2) \cup \cdots \quad (1)$$

ここで $f_M : (k, v) \mapsto (k', v')*$ は Map フェーズで行う計算を表す副作用のない関数であり，キーとバリューのリストを返す．ここで $(k', v')*$ はキーとバリューのリストを表す（$*$ がリストを意味する）．また $\cup$ はリストの連結を表す．よって Map フェーズの後もキーとバリューのペアの集合が出力される．

Combine フェーズでは，データは同じキーを持つもの同士集められて処理が行われ，以下の処理が行われる．

$$(k_1, \{v_{11}, v_{12}, \cdots\}), (k_2, \{v_{21}, v_{22}, \cdots\}), \cdots \longrightarrow f_C(k_1, \{v_{11}, v_{12}, \cdots\}) \cup f_C(k_2, \{v_{21}, v_{22}, \cdots\}) \cup \cdots \quad (2)$$

ここで $(k_i, \{v_{i1}, v_{i2}, \cdots\})$ は，同じキー k_i を持つバリュー $v_{i1}, v_{i2}, \cdots$ を集めたデータである．$f_C : (k, v*) \mapsto (k', v')*$ は Combine フェーズで行う計算を表す副作用のない関数であり，1 つのキーと，バリューのリストを受け取り，キーとバリューのペアのリストを返す．

⋆1 RDB の列方向格納に関する論文が VLDB2009 の 10-Year Best Paper Award を受賞している．

Vol.2010-DBS-150 No.9
Vol.2010-IFAT-99 No.9
2010/8/4

Reduce フェーズも Combine フェーズと同様であり，

$$(k_1, \{v_{11}, v_{12}, \cdots\}), (k_2, \{v_{21}, v_{22}, \cdots\}), \cdots \longrightarrow f_R(k_1, \{v_{11}, v_{12}, \cdots\}) \cup f_R(k_2, \{v_{21}, v_{22}, \cdots\}) \cup \cdots \quad (3)$$

の処理が行われる．ここで．$f_R : (k, v*) \mapsto (k', v')*$ は Reduce フェーズで行う計算を表す副作用のない関数である．f_M，f_C，f_R はデータごとに独立に計算することができるため，各サーバにタスクを独立に振り分けることにより分散計算を行う．

MapReduce 処理の流れを図 4 に示す．入力データとなるキーとバリューのペアの集合は分割され，各サーバで独立に Map フェーズの処理が実行される．次に Map と同じサーバで Combine フェーズの処理が実行される．続いてデータが Reduce フェーズの処理が行われるサーバに送られ，Reduce フェーズの処理が実行される．Combine では，同じキーのデータを集めて処理が実行されるが，異なるサーバには同じキーを持つ他のデータが存在している．同じキーを持つこれらのデータは Reduce フェーズで集められる．

提案するフレームワークでは，大規模データに対して分析処理を現実的な実行可能にするため，MapReduce により IO を並列化し，処理を高速化する．本研究では MapReduce の実装とし Hadoop 0.20.0 の Hadoop Streaming[6] を用いた．属性追加処理，抽出処理では Map フェーズのみを用いる．一方，集計処理では，Map フェーズで集計する属性を抽出し，Combine フェーズと Reduce フェーズで抽出した属性値を集計する．

3.3 コマンド/API とライブラリ

提案するフレームワークでは，分析処理を実行するためのコマンド/API と，分析でよく利用される関数のライブラリを提供している．

コマンドには，分析スクリプトをコンパイルするコマンド，小規模のデータに対してテスト実行するコマンド，全データに対して分析処理を実行するコマンドがある．また Web アプリケーションと連携できるように，これらの Web API も提供している．

以下ではサーバごとの平均 CPU 使用率の属性を追加する処理を例に，簡単にフレームワークの利用例を示す．固有の分析処理に関しては，将来的には，Java での記述が容易にできるようモジュールを補充する予定であるが，現在の実装では，Lisp の 1 つである Scheme 言語を用いることで，最も高速に動作し，かつフレームワークの機能をフルに利用できる．そのため以下では Scheme で分析スクリプトを記述した場合の例を示す．

まず図 5 に示すデータスキーマ schema.scm と，図 6 に示すサーバごとの平均 CPU 使用率の属性を追加する処理のプログラム ave-cpu.scm を記述する．schema.scm では，[l...] でリストを表し，(*階層名* ...) でタプルを表す．図 5 の例では，tag の階層の下に server の階層があり，さらにその下に record の階層がある．tag の階層には属性名が tag のデータがあり，server の階層には属性名が server-id のデータがある．record の階層には，utc や cpu などの属性がある．ave-cpu.scm には，分析処理の結果新しく生成されるデータのスキーマを new-index という変数名で定義し，また分析処理を mapper という関数名で定義する．ave-cpu.scm では，server の階層に新しく ave-cpu という名前の属性値を追加するので，new-index には server のタプルに ave-cpu とだけ記述されている．mapper 関数では，ref-servers や ref-cpu など，フレームワークが提供するアクセサを使って，CPU 使用率の平均を算出する処理を記述している．l-map 関数は，map 関数の第一引数と第二引数を入れ替えた関数であり，分析処理プログラムとデータスキーマとの対応を取りやすくするために用いる．

```
(*tag*
  tag
  [l (*server*
       server-id
       [l (*record*
            utc
            cpu
            mem
            rd
            wr
            net
            num-user
            log-text
            query-text)])])
```

図 5 データスキーマ schema.scm

分析処理を実行する際には，まずこれらのスクリプトを以下のコマンドでコンパイルする．

```
append_client.py $USERNAME compile --attrs="(cpu)" \
    ave-cpu.scm schema.scm
```

これによりフレームワーク側で実装されているライブラリと結合され，カレントディレクトリに実行可能なプログラムが生成される．attr オプションで分析処理に必要な属性名を指定する．USERNAME はユーザ名を表す環境変数である．

次に以下のコマンドで，テスト実行する．

```
append_client.py $USERNAME test --attrs="(cpu)" \
    ave-cpu.scm $DATA_ID
```

これにより，大規模のデータに対して分析プログラムを実行する前に，小規模のデータに対

Vol.2010-DBS-150 No.9
Vol.2010-IFAT-99 No.9
2010/8/4

```
(define new-index
  '(*tag*                                          ; tag のタプル
     [l                                            ; server のリスト
        (*server*                                  ; server のタプル
          ave-cpu)]))                              ; 新しく追加される属性

(define (mapper key tag)
  (mapreduce-write key
    (*tuple*                                       ; tag のタプルに対応
      [l-map (ref-servers tag) (lambda (server)    ; server のリストに対応
        (*tuple*                                   ; server のタプルに対応
          (mean (map ref-cpu (ref-records server))) ; ave-cpu の属性
        )])))
```

図 6 サーバごとの平均 CPU 使用率の属性を追加する処理のプログラム ave-cpu.scm

して，分析プログラムを実行し，プログラムにバグがないか確認する．DATA_ID は分析対象となるデータの ID を表す環境変数である．本フレームワークでは，データは ID で管理されている．詳細は本稿の主題とは関連が薄いため割愛する．

続いて以下のコマンドで，全データに対して分析処理を実行する．

```
append_client.py $USERNAME run --attrs="(cpu)" \
    ave-cpu.scm $DATA_ID $NEW_DATA_ID
```

NEW_DATA_ID は新しく生成されるデータの ID を表す環境変数である．

以上のように提案するフレームワークを利用することで，ユーザは分析処理のロジック部分にのみ集中してプログラムを記述できる．フレームワーク側でアクセサやデータ変換機能を提供するため，ユーザはデータ格納方式に関して意識する必要がない．また分散処理もフレームワーク側で自動的に実行される．これにより，分析処理のロジックと，分析処理を高速化するためのデータ変換や並列処理に関する実装を分離することができ，分析プログラムを開発が容易になる．

4. 評価実験

表 1 に示すデータを対象に評価実験を行ったデータ構造はタグの階層の下にサーバの階層があり，サーバの階層の下にレコードの階層がある．各階層ごとのデータ量を表 2 に示す．

典型的な属性追加処理と集計処理を評価対象のタスクとした．属性追加処理のタスクとして，レコードの階層でサーバごとに CPU 使用率の平均を算出し，1 つ上のサーバの階層に算出した平均 CPU 使用率を追加する処理を評価した．また集計処理のタスクとして，まず

表 1 評価用データのスキーマ

階層	属性数	備考
タグ	1	設置場所を表す tag 属性のみ
サーバ	1	サーバの ID を表す server-id 属性のみ
レコード	9	CPU 使用率を表す cpu 属性，メモリ使用量を表す mem 属性など

表 2 データ量

項目	データ量
タグ数	10,000
1 タグあたりのサーバ数	25
1 タグあたりのレコード数	4,000
合計レコード数	1,000,000,000
合計データサイズ	90,153,992,458 bytes

表 3 評価結果．スループットは分析対象となる全データ量を実行時間で割った値．

タスク	実行時間 (sec)	スループット (GB/sec)	1 サーバでのスループット (MB/sec)
属性追加	69.2 ± 13.3	1.33 ± 0.204	83.5 ± 0.64
集計	69.6 ± 1.5	1.29 ± 0.028	68.6 ± 0.94

レコードの階層でサーバーごとに CPU 使用率の標準偏差を計算し，次にサーバの階層でその標準偏差値の平均を算出する処理を評価した．実験環境として，コア数 2 のサーバ 9 台とコア数 8 のサーバ 10 台の合計 19 台からなるクラスタを用いた．フレームワークのコアの部分は Scheme で実装し，コマンドや Web API 等のインターフェース部分は Python で実装した．分析処理に固有の分析プログラムは Scheme で実装した．

評価実験の結果を表 3 に示す．表 3 には，5 回の試行に対する実行時間の平均と標準偏差を示している．分散ファイルシステムおよび Hadoop を用いず，ローカルファイルシステムと 1 サーバのみを用いて評価した結果を右欄に付記している．提案手法は，90GB のデータに対し，90 秒以内に分析処理を終えている．垂直分割データ格納と並列 IO により，平均で 1.2 GB/sec 以上のスループットを実現していることが分かる．ただしここでのスループット値は分析対象となる全データ量を実行時間で割った値であり，疑似的なスループット値であることに注意されたい．垂直分割データ格納方式により，実際に読み込んでいるデータ量は全データ量に比して小さい．また MapReduce による並列 IO により，1 台あたりのサーバが処理しているデータ量は，全サーバで処理するデータ量の 1/40〜1/13 程度である．

1 サーバで実行した場合に比べてスループットが約 16〜18 倍に向上している．Hadoop を利用した場合には，サーバ間の通信コストや分散処理の開始するための準備の処理のコ

Vol.2010-DBS-150 No.9
Vol.2010-IFAT-99 No.9
2010/8/4

ストがかかる．これらのオーバヘッドがあるにも関わらず，分散処理をすることで，スレーブ 19 台で約 16〜18 倍の性能向上を達成できている．従って分散処理の効果が十分に出ていると考えられる．

5. 考　　察

提案手法は，データ構造復元処理および属性分割処理の計算コストがかかり，さらに階層をたどるための計算コストがかかる．しかしながら提案手法では，1.2 GB/sec 以上の高スループットを実現できており，これらの計算コスト以上に，木構造データを属性ごとに分割して管理する方式による IO 負荷低減効果が大きいことが分かる．

提案するフレームワークでは，Hadoop により分散処理を行っている．大規模データ分析においては，IO を並列化するため大規模な計算機環境を必要とするが，このとき，ヘテロ環境での性能，スケーラビリティ，耐故障性が重視される．Abouzed らは，この 3 つの点において，Hadoop はパラレル構成の RDB よりも優れていると報告しており，Abouzed らが提案する HadoopDB システムにおいても分散処理アーキテクチャとして Hadoop を採用している[1)]．従って，大規模データ分析においては，パラレル構成の RDB よりも，Hadoop による分散処理の方が適切と思われる．一方で，RDB においても今後，MapReduce と同様の設計コンセプトを取り入れる可能性があり，RDB と MapReduce は融合していくことも予想される．

著者らの研究チームは 2008 年 11 月より Hadoop をデータ分析の用途で運用しており，2009 年 1 月より木構造よるデータ管理を開始した．2009 年 8 月に木構造データ分析フレームワークのプロトが完成し，その後，本フレームワーク上に分析用の Web アプリケーションを構築して，データ分析に実際に利用している．これまでのデータ分析において，本フレームワークを利用することにより，大規模のデータに対し短期間でデータの特性や特異性を見つけることが出来ている．フレームワークを利用することで，試行錯誤を含む発見フェーズにおいて，フレームワークを利用しない場合に比べて数十倍以上の分析効率を実現できており，本フレームワークの有用性を経験的に確認している．

総じて，(1) 木構造データを属性ごとに分割してファイルに格納する方法，(2) MapReduce モデルで分散処理する方法はいずれも大規模データ分析において有効であり，提案するフレームワークは大規模データ分析における属性追加処理や集計処理に適していると結論付けられる．

6. おわりに

本報告では，階層構造を持つ大量のデータを高速にデータ分析処理するフレームワークを提案した．本フレームワークでは，(1) 階層構造を持つデータを属性ごとに分割してファイルに保存し，(2) 分散計算フレームワークである MapReduce 上で分散処理する．評価実験の結果，19 台のスレーブからなる分散計算環境において，平均 1.2GB/sec 以上のスループットを達成できることが分かった．

今後は，提案手法の効果を定量的に評価できるようにするため，RDB との比較を行う予定である．また Scheme から Java への実装に切り替え，さらにボトルネックになっている部分は C で実装する．これにより，5〜6 倍程度の高速化を実現する．またフレームワークにおける分析プログラムの開発効率向上のため，GUI および DSL の開発を行う予定である．

参 考 文 献

1) Abouzeid, A., Pawlikowski, K.B., Abadi, D.J., Rasin, A. and Silberschatz, A.: HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads, *PVLDB*, Vol.2, No.1, pp.922–933 (2009).
2) Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D.A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R.E.: Bigtable: a distributed storage system for structured data, *OSDI '06: Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, Berkeley, CA, USA, USENIX Association, pp.15–15 (2006).
3) Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Proceedings of Sixth Symposium on Operating System Design and Implementation (OSD2004)*, pp.137–150 (2004).
4) Dean, J. and Ghemawat, S.: MapReduce: simplified data processing on large clusters, *Communications of the ACM*, Vol.51, No.1, pp.107–113 (2008).
5) Manegold, S., Boncz, P.A. and Kersten, M.L.: Optimizing database architecture for the new bottleneck: memory access, *The VLDB Journal*, Vol.9, No.3, pp.231–246 (2000).
6) White, T.: *Hadoop: The Definitive Guide*, Oreilly & Associates Inc (2009).
7) Xen: http://www.xen.org/.
8) 柳下幹生：Hadoop World NY2009 レポート，日経コンピュータ， Vol.11.25, pp.106–109 (2009).
9) 植田良一，佐藤嘉則，森　正勝，中村浩三，佐川暢俊：社会インフラの革新に貢献する知識化サービス基盤 KaaS，日立評論， Vol.92, No.5, pp.362–325 (2010).