インデータベース分析を巡る動向

齋藤　宗香，矢島　安敏

数テラバイトを超えるいわゆるBig Dataに対する新たな分析の枠組みとして，インデータベース分析について解説する．大規模データを扱う上では，分散処理を効率的に実現することが不可欠であるが，MapReduceと呼ばれる分散フレームワークを分散データベースとともに活用することで，新たな分析パラダイムが構築可能であることを紹介する．

キーワード：データマイニング，データベース，並列処理，MapReduce

1. はじめに

今日，社会や企業には膨大な量のデータが蓄積され続けており，いわゆるBig Dataの時代が到来した．インターネットにおけるユーザ行動ログは大規模データの典型例である．さらに今後は，例えばスマートフォンに代表されるモバイルツールが普及することで，ユーザ位置情報といったリアルな行動データなど，より大量なデータが生みだされるのは確実である．Big Dataを適切に分析することが可能となれば，将来さまざまな分野で新たな価値が生みだされるはずである[5]．

すでに，データベース技術の進歩とHDDなどのストレージ価格の低下により，このような大規模データの蓄積を実現するハードウェア環境は整備されつつある．一方で，大規模データを価値あるものとするためには，新たな分析技術や環境の整備が急務となっている．

数万から数百万レコードのデータを対象としていた従来のデータ分析では，データベースにデータをロードし，SQLなどによりデータ抽出・集計を行い，データマートとして分析用に形を整えた後，Rやデータマイニングソフトウェアを用いてクライアントPC上で分析を行うといったスタイルが主流であった．

しかし，分析対象のデータが数億レコードを超える規模となると，HDDやネットワーク転送速度がボトルネックとなり，データベースへのデータのロードやサーバからクライアントPCへのデータ転送さえままならなくなる．また，クライアントPCに搭載しているメモリでは，このようなデータすべてをロードするには不十分である．さらにまた分析ソフトウェアも，これだけの規模のデータを効率的に処理する分散実行機能を備えているわけではない．このようにデータの大規模化により，さまざまなソフトウェア・ハードウェアの制約が顕在化し，従来の分析スタイルを適用することは不可能となってきている．

1.1 インデータベース分析

このような制約をさけるべく，データベースを単なるストレージとしてではなく，高速なデータ集計，加えてはより高度な統計的分析・データマイニングを行う環境として用い，リアルタイムにデータから知見を得ようというのがインデータベース分析（In-Database Analytics）のモチベーションである．

増え続けるデータを処理すべく，BI向けデータベースソフトウェアは機能を増し発展してきた．まず，カラム型DB，テーブル圧縮など，データの保存方法の変更による1ノードでの処理性能の向上が行われた．その後，スケーラブルな分散ストレージによりノードを加えれば加えただけ格納可能なデータ量は増加し，大規模データの保存が可能になった．それだけでなく，分散ストレージに格納されたデータに対しての演算・集計処理を，各ノードで分散して実行する仕組みが考案され，膨大なデータを高速に処理する基盤ができつつある．例えば，データベース操作で頻繁に使われる制限，射影，結合といった関係代数の処理や要約処理は，分散ストレージ/分散処理との相性がよく，容易に分散型のものに置き換えることが可能である．

BI向け分散データベースであるGreenplumは，データを複数のサーバノードに分散して格納する．Greenplumにおいて，SQLは適切な処理単位であるスライス（Slice）に分割される．スライスは，分散処理に適した処理単位であり，スライスの処理をノードごとに実行し，その結果を集約（gather）すること

さいとう　むねよし，やじま　やすとし
㈱ブレインパッド
〒141-0022 品川区東五反田5-2-5

で1つのスライスの処理を終わらせる．その結果は再度分散格納されて，次のスライス処理の対象となる．このような処理を繰り返して，SQLで指定した処理を高速に実行している．

インデータベース分析では，大規模データに対して，ストレージから分析ソフトウエアにデータを移動させることなく，分散ストレージ・分散処理基盤を使い，データベースの内部で複雑な統計分析やデータマイニングを行うことを目的としている．Big Data時代の到来とともに，インデータベース分析技術には大きな期待がかけられているのである．

1.2 企業動向

Oracle，SQL Serverなど既存のRDBMSは大規模データに対応するべく，キャッシュ処理，インメモリ処理などを駆使し高速化を続けている．このような既存のRDBMSとは異なり，ひたすらに処理速度を追い求めたDWH向けDBと，それを基盤とした高度なアナリティクス技術を持った企業が現れてきている．

GreenplumはPostgreSQLをベースとしたシェアードナッシング型[1]の大規模並列処理（MPP：Massively Paralell Processing）により，大規模データを高速に処理するDBを開発した．それに加え同社は，DB内のテーブルに対してMapReduceと呼ばれる分散処理を実行する環境も提供している．また，Aster Data社は，nClusterという分散DBとSQL内のユーザ定義関数（UDF）をMapReduceで実装できるSQL-MapReduceという分散処理フレームワークを提供している．同社とも，分散ストレージ/分散処理フレームワークを備え，データ容量・分析処理が共にスケールアウトするRDBMSにより，大規模データ分析を実現するソフトウェアを提供している．

2010年よりこれらの企業の買収が立て続けに起きている．2010年7月にEMCがGreenplumの買収で合意したのを皮切りに，IBMがNetezzaを買収，2011年に入ってHPがVerticaの，TeradataがAster Dataの買収をそれぞれ発表している．買収を行ったのはいずれもハードウェア/アプライアンス製品と営業力/販売網に強みを持つ企業である．これらの企業は，ハードウェアのパフォーマンスを必要とするソフトウェア・ソリューションを持つ企業を買収することにより，既存ビジネスの拡大を図っているものと考えられる．

このような動きに先立ち，IBMは2009年にデータマイニングソフトウェアを開発しているSPSSを買収している．一方，SASは依然として独立企業としての地位を保ったままであるが，データ分析の領域に手を伸ばしてきているDBアプライアンスと既存のデータマイニングソフトウェアがどのように連携するか，各社の今後の動向に注目したい．

2. インデータベース分析を支える分散処理

この節では，MapReduceと呼ばれる分散処理のフレームワークを紹介し，データー分析手法がこのフレームワークの中でどのように分散処理されるのかについて解説する．

2.1 MapReduce

インデータベース分析を実現するためには，大規模データをデータ並列性の仕組みで処理することが有効なアプローチのひとつになってきている．

複数のプロセッサを使う並列処理には，大きく分けて，タスク並列とデータ並列の考え方がある．タスク並列による処理では，依存関係がなく同時に処理可能なタスクへの分解に注目した設計がなされるのに対して，データ並列では処理するデータを分割することで独立に処理が実行されるような設計が行われる．GoogleのMapReduce[3]は後者の代表的なものであると考えることができる．

MapReduceはGoogleによって提案された分散処理のフレームワークのひとつである．実行させたい処理をMapとReduceという2つの処理に分割することで分散化を実現するものである．一般的には，Map処理は，あるデータを入力として処理を行い，結果としてキーと値の組を中間データとして出力するものとして設計される．一方，Reduce処理ではMap処理で生成された同一キーの複数の中間データを入力とし，値に対して集計などの計算処理を実行した上で，やはりキーと値の組として出力するよう設計される．

このとき，Map処理はデータが分散配置された各計算ノードごとに，自身が持っているデータに対して実行され，他のノードでのMap処理とは独立に並列実行可能なように設計されていなくてはならない．その結果，計算ノード数を増やすことで，ノード数に比例した並列化が可能となる．同様に，Reduce処理もキーごとに各計算ノードに処理が分散され並列に

[1] 複数のサーバが1つのディスクを共有してデータを統一的に扱う「共有ディスク方式」に対して，各サーバ後のディスクに分散させる方式．

(高々キーの種類数までしか並列化できないが) 処理される.

2.2 単語数のカウント

頻繁に挙げられるMapRduce例として, 文書内に出現する単語数のカウント処理がある. Map処理では, 計算ノードに保存されている文章を入力とし, 単語をキー, 値を定数1とした

(単語, 1)

となる組を中間データとして生成する. Map処理では集計的な処理は行わず, 単語を1つ見つけるたびに(単語, 1)の組を1つ生成する. Reduce処理では, キーが同一, すなわち単語が同一な中間データを全て集めたものを入力として, それらの値の和を計算する. 結果, キーとなっている単語の出現回数が計算され

(単語, 出現回数)

の組として出力される. このようにMapとReduceという2つの処理を適切に設計することで, 分散配置された大規模なデータに対してデータ並列な処理が実行可能となる. 同様な考え方で, データベースにおける射影, 結合あるいは要約などの関係代数の演算も, MapとReduce関数の組合せで実現可能である.

このような単語のカウントといった単純な処理だけでなく, より複雑なデータ分析の処理も, MapとReduce処理のフレームワークで実行することが可能である.

2.3 行列演算

まず, 多くのデータ分析で基本となる行列演算を取り上げ, 具体的にどのように分散処理が実行可能か解説する. ここでは, 行列は非ゼロ要素のみが行番号, 列番号, 値の3列を1レコードとしたテーブルに記録されているものとする. さらに, このテーブルは行で分割され複数の計算ノードに分散して格納されていると考える.

まず, $A=\{a_{ij}\}$ を n 行 m 列の行列, $v=\{v_j\}$, $x=\{x_i\}$ をそれぞれ n 次と m 次のベクトルとする. このとき,

$$x=Av$$

となるベクトル x を求めるためのMap処理とRduce処理を構成する.

Map処理は, 行列 A の各要素 (a_{ij}) に対応したレコード (i, j, a_{ij}) とベクトル v を入力とし, キーと値の組 $(i, a_{ij}v_j)$ を中間データとして出力する. ここで, キーは行の添え字 i で, 値が $a_{ij}v_j$ である. Reduce処理では, キーが等しい中間データ全てを入力として, その値の和 $\sum_j a_{ij}v_j$ を算出する. これはベクトル x の i 番目の要素 x_i に他ならない. なお, 中間データには非ゼロ要素のみが現れているだけなので, 結果的にReduce処理も非ゼロ要素のみの和を計算することになる. 最後に全てのReduce処理の結果を集めれば, ベクトル x が構成できることになる. 言うまでもなく, ここでのMap処理は行列 A の要素ごとに独立して実行可能であり, Reduce処理も x の要素ごとに独立して計算可能である.

次に, $B=\{b_{jk}\}$, $C=\{c_{ik}\}$ をそれぞれ m 行 p 列, n 行 p 列の行列とし, 行列と行列の積 $C=AB$ を計算する場合のMapとReduceについて考える. Map処理では, 行列 A の各要素 a_{ij} ごとに, p 個(すなわち行列 B の列数)の中間データ

$$((i, k), (A, j, a_{ij})),\ k=1, 2, ..., p$$

と行列 B の各要素 b_{jk} ごとに n 個(すなわち行列 A の行数)の中間データ

$$((i, k), (B, j, b_{jk})),\ i=1, 2, ..., n$$

を生成する. これらの中間データでは, キーが2つの添え字の組 (i, k), 値が (A, j, a_{ij}) あるいは (B, j, b_{jk}) である. 一見すると, だいぶ冗長に中間データが生成されているが, このようにすることでその後のReduce処理が互いに独立して実行ができる.

Reduce処理では, キー (i, k) が等しい中間データが全て集められ入力となる. これらの値は添え字 j ごとに A の要素と B の要素とを

$$(A, j, a_{ij}), (B, j, b_{jk})$$

と組にして和を

$$c_{ik}=\sum_j a_{ij}b_{jk}$$

と計算すれば, これは行列 C の i 行 k 列要素 c_{ik} となる. もちろん, この場合もMap処理では行列 A, B の非零要素のみが生成されるので, 全ての j に関してこのような組がReduce処理に渡されるわけはなく, 非ゼロ要素が組になったものだけに対して処理がなされる. 最終的なReduce処理の出力を添え字の組 (i, k) をキー, c_{ij} を値としたものとすれば, これは計算結果の行列 C の保存形式にそのまま対応する.

以上のように, 基本的な行列演算をMapReduceの枠組みで設計すれば, CG法やLanczos法により線形方程式系の解や特異値分解といった複雑な行列処理も実行可能となる.

2.4 MapReduceのマイニングアルゴリズムでの活用

以下では，幾つかの代表的なデータマイニングのアルゴリズムに対して，分散的な環境下でどのように処理が可能かを，MapReduceを中心に解説する．

以降では，学習データはm個のn次ベクトル$x_1, x_2, ..., x_m \in \mathrm{R}^n$で与えられ，各データにはラベル$y_1, y_2, ..., y_m$が付与されていることを仮定する．また，データの個数mは属性数nと比べ非常に大きく，チャンクに分割され複数の計算ノードに分散して保存されている状況を想定する．各計算ノードに配置された部分的な学習データに対して他のデータとは独立にMap処理を行い，その結果をReduce処理で統合するというMapReduceの枠組みで，多くのデータマイニングアルゴリズムを設計することが可能であることを紹介する．

クラスタリング　まず，代表的な教師なし学習であるk-means法によるクラスタリングアルゴリズムを考える．クラスタリングの目的は，m個のデータを適当な距離を使い互いに近いデータ同士を選び出し幾つかのクラスタに分割することである．

k-means法では，まず分割数kを定めた後，ランダムにk個のデータを選び仮のクラスタ重心を定める．その後，この重心を繰り返し更新する操作が行われる．各繰り返しでは，

- m個のデータそれぞれを，最も距離の近いクラスタ重心に対応させることでk個に分割する操作
- 各クラスタごとにそこに分類されたデータの重心を求めクラスタ重心を更新する操作

が交互に行われる．

この場合，Map処理では，データベクトルx_iとk個のクラスタ重心を入力とし，キーとしてx_iに最も近いクラスタ重心のインデックス，値としてデータのベクトルx_iを出力する．Reduce処理では，キーの等しい，すなわち同一クラスタに所属する全ての中間データを入力として，値となっているベクトルの重心を計算する．出力としてクラスタ重心のインデックスと更新されたクラスタ重心を生成する．Map処理もReduce処理も互いに独立して実行可能である．

ナイーブベイズ　ナイーブベイズ法は，教師ありの学習のひとつで，クラス分類を行うための手法である．分類器を求めるためには，データの属性間に独立性の仮定をおいたうえで確率$P(y)$およびj番目の属性$x(j)$に対する条件付き確率$P(x(j)|y)$を推定する．

$P(y)$の推定には，各クラスごとのデータ数のカウントが必要である．そのためには，2.2節で説明した単語のカウントと同様なMap処理とReduce処理を行えばよい．すなわち，Map処理では，i番目のデータのラベルy_iを入力として，キーとしてy_i，値として定数1となる組

$(y_i, 1)$

を中間データとして生成する．Reduce処理では，キーが等しい中間データ全てを入力として，それらの値の和を算出すればよい．また，確率$P(x(j)|y)$も，属性$x(j)$が離散的であれば，$(x(j), y)$をキーとした同様なMapとReduce処理を設計しデータ数をカウントすることで推定が可能である．

サポート・ベクター・マシン　次に2クラスの教師つき学習を行う代表的な手法であるサポート・ベクター・マシン（SVM）を考える．なお，ここでは，クラスラベルy_iは± 1の2値であると仮定する．

SVMでは，各学習データに対応したm個の変数α_iと適当なカーネル関数$K(\,)$および正の定数Cで定まる2次計画問題：

$$\left|\begin{array}{ll} \text{最小化} & \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}K(x_i, x_j)\alpha_i\alpha_j + \sum_{i=1}^{m}\alpha_i \\ \text{制約} & \sum_{i=1}^{m}\alpha_i y_i = 1, \\ & 0 \le \alpha_i \le C,\ i=1, 2, ..., m \end{array}\right. \tag{1}$$

の最適化を実行することで学習が行われる．最適解α_i^*の中で非零のものに対応した学習データx_iはサポートベクターと呼ばれている．最適化問題(1)からも明らかなように，判別器の学習にはサポートベクターだけで十分であることが知られている．

最適化問題(1)は，学習データの増加とともに大規模な2次計画問題となってしまうため，SVMは大規模なデータの学習には必ずしも適していないと考えられている．そこで，Cascade SVM[4]と呼ばれるアルゴリズムでは，学習データをいくつかに分割した上でそれぞれ独立に小さなサイズの2次計画問題を解き，最適解を集約する処理を繰り返すことで学習を行う．

このCascade SVMはMapReduceのフレームワークで処理が可能である．すなわち，まず，各ノードにおいて，自身が保持している学習データのみを対象に最適化を実行する．これは，Map処理に対応し，各ノードごとに並列に処理が実行され，最適化の結果と

してそれぞれのノードでサポートベクターが算出される．

次に，幾つかのノードで得られたサポートベクター同士を集め，新たに何組かの学習データ集合を分散環境上に配置する．再びそれぞれのノードごとに独立して学習（最適化）を実行することで，新たなサポートベクターを生成する．このプロセスを学習データが1組になるまで次々に繰り返せば，最後の最適化問題の解は，全学習データによる最適解に非常に近いものとなることが，Graf等によって報告されている[4]．さらに，彼らは最後に得られたサポートベクターを，最初に分割した学習データ集合にそれぞれにマージした上で，再び同じステップの実行を繰り返すことで，理論的にも最適解に収束することを示している．

回帰分析　ここでは，各ラベル y_i は実数値と考え，線形関数

$$y=\beta^{\mathrm{T}}x$$

の n 次のパラメータベクトル $\beta\in\mathrm{R}^n$ を分散的な環境で効率よく求めることを考える．

X をデータベクトル x_i を第 i 行に持った m 行 n 列行列とすれば，β は線形方程式系

$$(X^{\mathrm{T}}X)\beta=X^{\mathrm{T}}y \tag{2}$$

の解として記述される．データ数 m が属性数 n に比べて非常に大きい場合，最も計算が必要となる部分は行列の積 $X^{\mathrm{T}}X$ である．

そこで，この行列の積を

$$X^{\mathrm{T}}X=\sum_{i=1}^{m}x_ix_i^{\mathrm{T}} \tag{3}$$

と書き換えれば，Map処理として，それぞれのノードで保持しているデータを使い部分和 $\sum x_ix_i^{\mathrm{T}}$ の計算を並列に実行し，その後1つのReduce処理で各Map処理結果の総和を算出すればよい．$X^{\mathrm{T}}y$ も同様で

$$X^{\mathrm{T}}y=\sum_{i=1}^{m}x_iy_i \tag{4}$$

と和の形に書き換えれば，ノードごとにMap処理として部分和 $\sum x_iy_i$ を計算し，Reduce処理で総和を求めればよい．

なお，多くの機械学習のアルゴリズムは，式(3)のように学習データ集合上で和の形（summation form[1]と呼ばれている）で表現可能であることが知られており，MapReduceのフレームワークで実装可能であることが知られている．

ここで取り上げたアルゴリズム以外にも，決定木[8]，頻出集合の抽出[7]，あるいはブースティング[6]などさまざまのデータマイニングアルゴリズムをMapReduce処理を設計することで並列化する試みがなされている．

3. インデータ分析の実際

3.1 Greenplumにおけるインデータ分析

2010年にGreenplumを買収し企業の一部門としたEMCは，Greenplumを利用したインデータベース分析向けオープンソースライブラリMADLibの開発を推進している．MADLibは，Greenplumが提供する高速な分散SQL処理を用い，より高度な分析へと足を踏み入れている．分析処理をSQLとして記述することで，スケーラビリティを確保しつつ高度な処理を実現しようとしている．

なおMADとは，インデータベース分析が，あるいはそれを実現するデータベースが備えるべき3つの特徴を表すキーワード「Magnetic」「Agile」「Deep」に由来している．「Magnetic」とは，企業内に散在しているありとあらゆるデータを，たとえデータの質が異なっているものであっても，データを集中させることが可能なこと，「Agile」は，データベースやそこでの分析手法がデータの変化に柔軟に対応できること，また，「Deep」は，単に集計にとどまらない高度で複雑な分析が実行可能であることを意味している[2]．

一つの例として，SQLを使って線形回帰を実行する例を紹介する．ここでは，データ行列 X は，前節とは異なり行ベクトルを1レコードとしたスキーマ

```
M(row_number integer, vector numeric[])
```

として行番号（row_number）とベクトル（vector）の組で表現されている．その上で，行番号のハッシュ値などを使いデータベース内に分散的に保持されている．

図1は，回帰係数 β と R^2 値を計算するSQL文の例である．この中の，

```
sum(transpose(m.vector) * m.vector)
sum(m.vector * y)
```

がそれぞれ前節の式(3)と(4)に対応した部分となる．

なお，図1のpseudo_inverse関数は逆行列処理のための関数であるが，その処理はSQLでの実装は困難である．そのため，他の言語で実装しSQLから呼び出せる機能がサポートされている．SQLで記述できる箇所については，Greenplumが適切に分散処理

```
CREATE VIEW ols AS
SELECT pseudo_inverse(A) * b as beta,
(transpose(b) * (pseudo_inverse (A) * b)-sum_y2/
count)
  /(sum_yy-sumy2/n)
     as r_squaredp
FROM(
  SELECT sum(transpose(m.vector) * m.vector) as A,
  sum(m. vector * y) as b,
  sum(y)^2 as sum_y2, sum(y^2) as sum_yy,
  count(*) as n
  FROM M m
)
ols_aggs;
```

図1 回帰分析を実行するSQL文の例

を行ってくれるが，ユーザ自身が実装したユーザ定義関数（UDF : User Defined Function）は自動的にスケールするわけではない．ちなみに，MADLibの実装[2]では，式(2)の線形方程式の処理をlapackのdgesdd_を呼び出し特異値分解を使い実現している．

前節の回帰の部分で述べたように，X^TXの計算の手間が支配的な状況であれば上記のような戦略で有効と思われるが，X^TXが大きな行列になるにつれ，式(2)の処理を並列化する必要がある．

3.2 AsterDataにおけるSQL-MapReduce

Aster Data社はnClusterと呼ばれる高速なデータベースを主力製品としている．同製品はMySpace，Akamaiなどに導入され，その性能を生かして実績を挙げている．同社はnCluster上でのインデータベース分析環境として，SQL-MapReduceを提供している．SQL-MapReduceは，UDF内の処理をMapReduceで記述するインターフェースを提供している．これを用いて実装された関数はnCluster上での分散実行が可能になる．SQL-MapReduceを用いることで，複雑なSQLを駆使して行っていた集計処理を，よりシンプルに実装できるという利点もある．

SQL-MapReduceでは，UDF内の処理を，行関数（Row-Function），パーティション関数（Partition-Function）として実装する．SQL-MapReduceとして実装したUDFはテーブルと同形式のデータを出力

```
SELECT ... FROM functionname(
       on table-or-query
       [PARTITION BY expr...]
       [ORDER BY expr...]
       [clausename(arg...)]...
)
```

図2 SQLMapReduceの例

```
SELECT ts, userid, session
FROM sessionize(
    on clicks
    PARTITION BY userid
    ORDER BY ts
    TIMECOLUMN('ts")TIMEOUT(60)
)
```

図3 SQL-MapReduceで実装した関数の呼び出し例

し，SQLからはテーブルと同等に扱うことができる．SQLから図2のように呼び出すことができる．

行関数（Row-Funciton）は各レコードに対するデータ変換処理を行う．行関数によって変換されたレコードがPARTITION BY，ORDER BYにより分割，ソートされて，パーティション関数（Partition関数）でそれらのレコードに対する集計処理を行う．

図3で示したSQL-MapReduce処理の呼び出し例では，クリックログに対して，同一ユーザによるクリック間隔が60秒以内のクリックの同一のセッションIDを付与する．

この呼び出しにより，

```
ts userid
10:00:00  238909
00:58:24  7656
10:00:24  238909
02:30:33  7656
10:01:23  238909
10:02:40  238909
```

というテーブルに対して，次のようなsession列が付与される．

```
ts userid session
10:00:00  238909  0
10:00:24  238909  0
10:01:23  238909  0
```

[2] MADLib 0.1.0 alpha版

```
10:02:40  238909  1
00:58:24  7656    0
02:30:33  7656    1
```

行関数（Row functions）は行ごとの変換処理であり，パーティション関数（Paritition function）は分割，ソートされたデータ群に対する集計処理を実装する．それぞれMapReduceにおけるMapとReduceに対応する処理である．

4. おわりに

本稿では，いわゆるBig Dataと呼ばれている超大規模なデータに対して，MapReduceなどの分散処理を使ったデータ分析を巡る企業動向と技術について解説をした．

従来のRDBでは，四則演算，データ変換など各レコードに対する加工手続き（Function）と，最大・最小値，平均値の算出など，レコードの1回走査で処理が完結する集計（Aggregation）を主な処理対象にしていた．通常は，データベースの構築設計の段階で格納するデータの型や集計の方法が定められており，OLAPキューブを構成したり，あるいはインデックスを構成することなどにより，できるだけデ全データに対する走査処理が行われない仕組みを作ることが，データベースの設計の上で重要な要素になっている側面が強い．

しかし，データの変化が激しく，適切な分析の方法論が確立されていない領域では，データ型の変更や新たな属性追加，それに伴い全データに対する走査処理は避けられない．また，単なる集計にとどまらず，回帰，クラスタリングや判別分析といったより高度で複雑なデータマイニングアルゴリズムがデータベース内で実現できることは，大規模分析を行う上で必須な条件である．3.1節で紹介した「MAD」という考え方は，まさにBig Data時代のデータ分析の特徴を表している．インデータベース分析では，これらの処理を通常のSQL処理と組合せ，スケーラブルかつシームレスに実現することを目指さなくてならないと考えている．

参考文献

[1] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng and K. Olukotun, Map-Reduce for Machine Learning on Multi-Core, NIPS, 281-288, MIT Press, 2006.

[2] J. Cohen, B. Dolan, M. Dunlap, J.M. Hellerstein and C. Welton, MAD Skills : New Analysis Practices for Big Data, Proc. VLDB Endow, 2, 1481-1492, 2009.

[3] J. Dean and S. Ghemawat, MapReduce : Simplified Data Processing on Large Clusters, Commun. ACM, 51, 107-113, 2008.

[4] H.P. Graf, E. Cosatto, L. Bottou, I. Durdanovic and V. Vapnik, Parallel Support Vector Machines : The Cascade SVM, In Advances in Neural Information Processing Systems, 17, 521-528, 2005.

[5] McKinsey Global Institute Report, Big data : The Next Frontier for Innovation, Competition, and Productivity, 2011.

[6] I. Palit and C. Reddy, Parallelized Boosting with Map-Reduce, Proceedings of the 2010 IEEE International Conference on Data Mining Workshops, 1346-1353, 2010.

[7] H. Li, Y. Wang, D. Zhang, M. Zhang and E. Chang, PFP : Parallel FP-growth for Query Recommendation, In Proceedings of the 2008 ACM Conference on Recommender Systems, 107-114, 2008.

[8] B. Panda, J. Herbach, S. Basu and R.J. Bayardo, PLANET : Massively Parallel Learning of Tree Ensembles with MapReduce, Proc. VLDB Endow., 2, 1426-1437, 2009.