大規模データ解析を支えるOSS技術

基般

海野裕也（(株) Preferred Infrastructure）
熊崎宏樹（日本電信電話（株））

大規模データ処理の潮流とOSS

大規模データ処理の潮流

インターネット技術の進歩，利用人口の増加，サービスの高度化などに伴って，処理すべきデータ量は爆発的に増大を続けている．また，量だけではなく種類も増えている．従来のデータベースに適した構造化データだけでなく，テキスト，動画，音声など，多様な非構造化データが扱われている．

企業に蓄えられるこれらのデータは新しい価値を生む原石であり，それらを活用したいというニーズは，この数年の間に急速に強まってきている．

しかし，これらのデータを活用するためには3つの問題がある．1つ目は単一のストレージやデータベースでは対応できない量となったデータの処理方法，2つ目は大規模データ処理手段の操作性，3つ目はデータの種類の多様性である．コンピュータの黎明期から，上記課題を解決する大規模データ処理基盤のコア技術として，大規模並列分散処理は数多く研究されてきたが，たとえば大規模分散物理シミュレーションでのチェックポイントを用いた耐障害性の仕組みのように，個々の問題設定と障害耐性やスケールアウト性が密に結合しており，どのシステムにも適用できる汎用的な分散処理技術はなかった．そのため個別に開発を行うしかなく，その膨大な開発コストが分散システム導入の大きな障害となっていた．

汎用的な分散処理技術を目指していろいろなソフトウェアが開発されてきているが，注目を集めている物の多くはオープンソースソフトウェア（OSS）である．OSSに注目が集まった最大の理由は，業界の多くの企業が情報爆発時代に対応できる共通の処理基盤が早急に必要であると判断した結果にほかならない．特にWeb企業の大半は，解析基盤を持つこと自体が競争力の源泉ではなく，データの解析によっていかにサービスを改善できるかにあることにいち早く気が付いて利用していた．現在では，企業や研究機関もOSS開発に参加して，開発コストを抑えつつそのOSSを利用して独自の価値を生み出す方法が主流になってきている．

Hadoopの出現

大規模データ処理を支えるOSSの中でも，最も名が知られ，最も成功したソフトウェアの1つがApache Hadoopであろう．Hadoop開発の元となったのが，2004年にGoogleの発表したMap Reduceという計算モデルである[1)]．MapReduceは，さまざまなバッチ処理をMapとReduceの2つの単純な処理に分解して実行する．Mapはデータ単位ごとの数え上げや集計の処理を行い，Reduceは変換結果の集約や統合を行う（**図-1**）．処理対象のデータは，あらかじめGoogle File System（GFS）という，分散ファイルシステムに保存する．MapReduceを実行する際には，GFS中のデータをメモリに十分収まる程度に小さな処理単位ごとに分割して分散処理を行う．Map処理は，データごとに独立して行うため分散環境と相性がよい．Map処理を行ったすべてのデータを横断的に集計するReduce処理は，計算にすべてのデータが必要なことからも分かる通り，Map処理ほどのスループットは出ない．しかし，計算処理の大部分はMap処理によって並列化されるため，全体の計算効率は

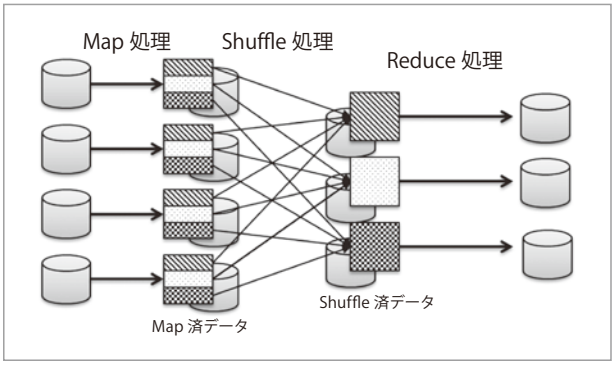

図 -1　MapReduce の模式図．図中の四角形はデータを表している．Map 処理では，分散されたディスクに保存されたデータに対してそれぞれ独立にデータ単位ごとの数え上げや集計のような処理を行う．そして Shuffle 処理で，指定したキーごとにデータをまとめ，データ間にまたがる処理を最後の Reduce 処理で行い，結果を再びディスクに格納する．

きわめて高い．実際に Google では，この仕組みを使うことで，数千台のサーバにまたがる大規模な MapReduce 処理を毎日千回以上行うことができるようになった．

GFS は，常にすべてのデータを三多重で複数のサーバに保存しているため，一部のサーバに電源断やハードウェア故障などの障害が発生した場合であってもデータを喪失する危険性は低い．加えて，Map 処理の結果は常に個々のサーバのディスクに書きだされ，Reduce 処理の結果は常に GFS に書き出されるため，データの喪失の危険はさらに低くなる．もし，Map 処理の途中でサーバが故障した場合でも，別の Map 用のサーバ上で GFS から対象データのみを読み出して再実行する．Reduce 処理の途中でサーバが故障した場合は，Map 処理を行ったサーバのディスクに書き出された内容を，別の Reduce 処理用のサーバが読み出して再実行する．このように重厚な再実行の仕組みとこまめなディスク書き出し処理によって障害発生時の影響を小さく抑えている．

サーバ台数を増加させて分散処理を行うことで，より高速にデータを処理できる特性のことをスケールアウトと言う．MapReduce 論文の最大の貢献は，処理を Map と Reduce に分割して，さまざまなワークロードにおいてスケールアウト性と耐障害性を実現するフレームワークの一例を示したことである．

MapReduce 以前にもエンタープライズシステムではスケールアウトを実現していたが，対障害性と両立するフレームワークは考えられていなかった．

この Google の論文のアイディアに触発されて分散処理基盤を OSS という形で再実装したものが Hadoop である．Hadoop では，前述した MapReduce の仕組みを実装し，かつ GFS の代わりに Hadoop File System（HDFS）を提供している．Hadoop は，その高いスケールアウト性と信頼性が認められ，さまざまな企業で使われるようになり，現在では日々誕生するさまざまな大規模データ処理基盤の性能・対障害性のベースラインの役割を担っている．

Hadoop で解決できない課題

Hadoop はすべての問題を解決できる「銀の弾丸」ではなく，解決できない課題もある．それらを以下に示す．

（1）レスポンスが遅い

- すべての処理内容をディスクに一度書き出し，その後データ全数を対象として処理を行っているため，素早いレスポンスが要求されるタスクには向かない．
- データ総数が増え続ける環境下でリアルタイムなレスポンスが求められるアプリケーションには適用できない．
- 機械学習や統計解析のように繰り返しデータを使う処理に対しては十分なパフォーマンスを発揮できない．

（2）汎用的な機械学習や統計解析の仕組みが提供されていない

- Hadoop は分散処理の仕組み以外は提供しない．たとえば機械学習や統計解析処理は提供されていない．そのため，各利用者が個別に解析処理を記述したり用意したりする必要がある．

（3）アドホックな解析の生産性が低い

- Hadoop の MapReduce 処理は Java で記述する必要がある．Java は，簡単な処理であってもシンプルに記述するのが難しい言語仕様であり，解析方法の改善や機能追加などが容易に行えない．
- HDFS に保存したデータのみを対象とするため，

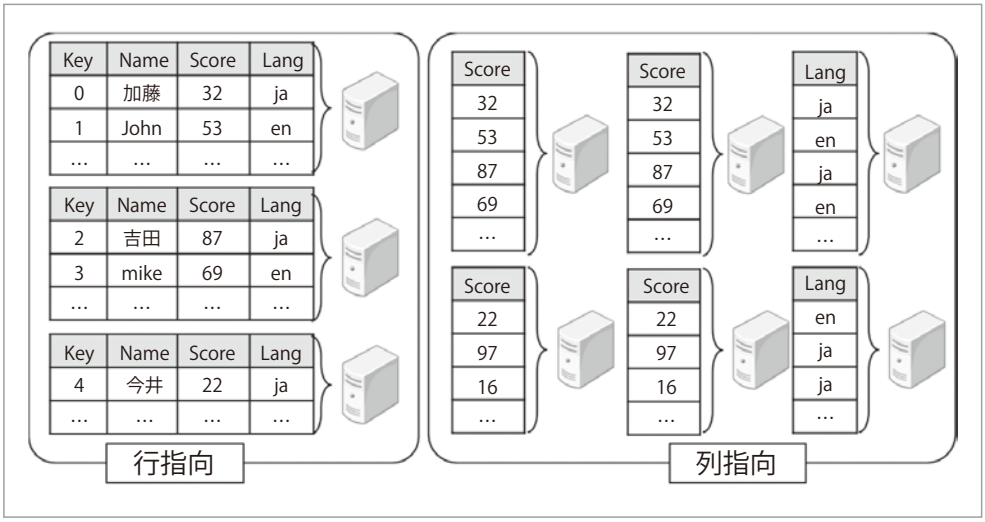

図 -2　行指向(左) と列指向(右) データ構造の比較．列指向データ構造の場合，同一種類のデータが連続して保存される．

HDFS を使わない既存システムに溜め込んだデータを使う場合は HDFS にインポートしなくてはならない．

これらの問題については 1. レスポンス性能，2. 高度な解析処理，3. 解析処理の生産性 の 3 つの観点から解決が試みられており，OSS として提供されている．本稿では，それらの OSS を通して解決方法を解説する．

レスポンス性の向上

Hadoop はスループットが高い反面，レスポンス性の向上には向かない．たとえば Reduce 実行時には必要なデータが揃っていなければならないため，Map 処理の結果が集まるのを待つ必要がある．サーバを追加して Map 処理の性能をスケールアウトさせても Reduce 処理が全体を律速してしまうため，性能向上には限界がある．

しかし，大規模データ処理においては，スループットよりもレスポンス性が要求される場面は多い．たとえば，データ解析等の場面ではその場の仮説に基づいて任意の操作（クエリ）を記述して集計をすることが普通である（これをアドホッククエリと呼ぶ）．株取引のように応答時間が重要な競争力となるような分野や，対話的にデータ解析を行いたいという状況がこれにあたる．こうした背景から Hadoop でもレスポンス向上技術が望まれるようになり，それを実現する技術が OSS で生まれている．ここでは，アドホッククエリを高速化する OSS と，メモリを活用して I/O 時間を高速化する OSS を紹介する．また，次々にくるデータの処理（ストリーム処理）を実現する OSS についても紹介する．

アドホッククエリの高速化

大規模データの解析は試行錯誤を繰り返すことが多く，アドホッククエリのような対話インタフェースの需要は大きい．

このアドホッククエリを高速化する方法として Google が 2010 年に発表したものが，列指向型超並列データベース Dremel[2] である．Dremel について書かれた論文では 2 つの技術が紹介されている．列指向データ構造と，それに対する分散クエリシステムである．

列指向データ構造とは，検索対象となるデータを保存する際に特定の属性，たとえば顧客の名前と年齢を保存するテーブルであれば年齢だけをそれぞれデータから抽出し，個別のデータ列として管理するデータ構造である．**図 -2** のように同一の属性のみを並べて保存することによって効率的な圧縮やデータ検索が可能となる．アドホッククエリの大部分はテーブルの中の特定の列のみを参照する．そのため，列指向データ構造でテーブルを保持しておくことに

より，大量の属性を持つ大規模データであってもクエリごとにテーブル内の必要な部分のみをスキャンできるため，高速化を図ることができる．Dremel ではこのデータを複数のサーバで保持することで冗長化も実現している．さらに Dremel は JSON（JavaScript Object Notation）のようなネスト構造や繰り返し構造（配列）のあるデータに対してスキーマ（データベースの構成）を定義し，最小限のビット数を割り当てるなど効率的な列構造化ができるようにしている．一方，分散クエリシステムとは，前述の列指向データ構造で保持されたテーブルを対象に，ユーザから発行されたアドホッククエリを複数のサーバで分割処理するシステムである．1 つのクエリを複数のサーバで処理できるよう再帰的に分割を繰り返した後に，クエリの処理対象となる列構造データを持っている個々のサーバに，分割された小さなクエリを発行して並列に実行する．その後，各サーバに対するクエリの検索・集計結果を再帰的に合成していくことで最終的な結果を作成しユーザに返す．これによって非常に短いレスポンスタイムが期待できる．Google は Dremel を 2006 年から運用しており，毎月 1,000 兆以上のレコードをスキャンしていると説明している．Dremel は OSS ではない．しかし，この Dremel に触発されて開発された OSS 実装に Apache Drill がある．Apache Drill は MapR が中心となって開発しており，Dremel と同様に JSON のようなネスト構造を持ったデータのクエリを高速に実行できる．複数の種類のデータストアにクエリを発行できる点が特徴である．それを可能にするために複数のストレージプラグインが用意されており，MySQL，Oracle Database，MongoDB，Cassandra といったデータストアが Drill 経由で利用できる．

アドホッククエリ処理が可能なその他の OSS の超並列列指向データベースエンジンとして，Cloudera が開発している Impala がある．C++ で実装されており，HDFS や HBase に保存したデータに対して高速にクエリすることを目的としたエンジンである．既存エンジンの 10 倍以上高速という点が売り文句となっている．

また，Facebook も OSS の MPP（Massively Parallel Processing）エンジンの Presto を開発している．データストアを抽象化し，プラグインによって具体的な機能を拡張できる仕組みになっており，クエリの実行計画を立てるプランナや計画の実行処理を複数のサーバに分配するスケジューラなど，利用者が独自の処理を反映できるのが特徴である．

Impala と Presto はどちらも活発に開発が行われている．ただし，それぞれ機能仕様が異なるため，採用にあたっては現状の仕様をよく見極め，目的にあったものを選択する必要がある．

メモリを利用した高速化

従来の Hadoop の MapReduce は，ディスク中のデータを読み込んで解析処理してディスクに書き戻すまでが 1 つの処理であった．この方法は，複数のバッチ処理を連続して実行したり，単一のバッチ処理を何度も実行したいという要求には向かない．たとえば，多くの機械学習手法では，反復法といって内部パラメータを繰り返し更新する方法を用いるが，Hadoop では処理を行うたびにディスクに書き出すため時間がかかる．この課題を解決する目的で開発された OSS が，Apache Spark である．

Spark はカリフォルニア大学バークレイ校の AMPLab で研究開発されている．Spark は，各 MapReduce 処理の終了時に計算を行ったサーバのメモリ上に実行結果を展開するため，処理の段数が増えてもディスク読み書きの負荷が上がることはない（**図 -3**）．その反面，処理もメモリ上で行うため，クラスタ全体のメモリ総量よりも大きなデータを扱う処理の場合は，ディスクに書き出す処理が加わってしまうため，速度が劣化してしまう．

また，Hadoop がディスクにデータを書き出しているのは主に耐故障性のためであり，ディスクへの書き出しを行わないことで高速化を実現している Spark には，当然 Hadoop と同一の耐故障性はない．しかし Spark は RDD（Resilient Distributed Dataset）という仕組みを用いることでこの問題を

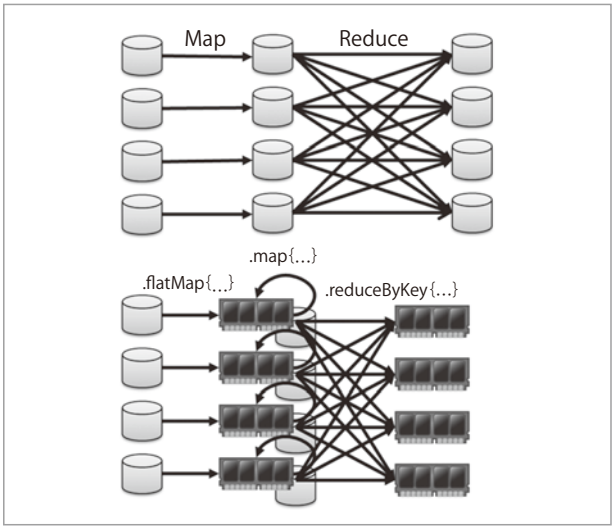

図 -3　Hadoop（上）と Spark（下）での反復処理の比較．Hadoop では処理のたびにディスクに書き出す必要があるが，Spark ではメモリ上にキャッシュされるため，ディスクアクセス回数は最小限に抑えられる．

解決している．RDD では，HDFS 等のデータソースから読み出したデータを MapRuce のときと同様にある程度の大きさごとに 1 つの処理単位としてまとめ，その処理単位に対して 1 プロセッサを割り当てて処理を行う．もしその処理を行っているサーバが故障などの要因でシステムから離脱した場合には，該当単位のみを再びデータソースから取り出して部分的に再実行する．これにより全体の再実行を回避し，耐障害性と高速化を高いレベルで両立させている．

ストリーム処理

Hadoop によってバッチ処理を高速化することが可能になったが，大規模データ処理のすべてがバッチ的に行われるわけではない．その 1 つが次々にやってくるデータを処理する，ストリーム処理の分野である．

分散環境でストリーム処理を実現する OSS に Apache Storm がある．Storm は，元々 BackType が開発していたが，同社が Twitter に買収されてから OSS として公開された．Storm は，蓄えられたデータの処理ではなく，Twitter のサービスのように生成され続けるデータの処理を目的に開発されている．ある処理を行った後に何の処理をするか

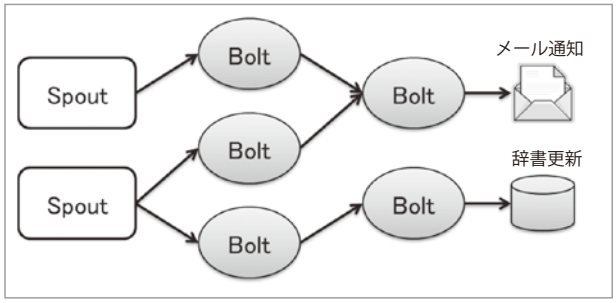

図 -4　Storm の処理トポロジの例．1 つの処理単位を表す Bolt が処理のネットワークを構成し，そこにデータ源である Spout からデータを「流す」ことで大規模な処理を実現する．流した結果，異常をメールで通知することや，継続的に辞書を更新するといった処理ができる．

という処理トポロジを決めて，そこにデータを流す（**図 -4**）．トポロジは Spout とよばれるデータの生成源のノードと，Bolt とよばれる処理を行うノードからなり，各ノードの出力結果をどの Bolt の入力にするかを記述する．Nimbus とよばれるプロセスが全体を管理し，各サーバ上の Worker プロセス上でトポロジの各処理を実行する．これによってデータが追加された際に即座に結果を更新できるため Spark よりもさらに速いデータの更新が可能となる．

データを逐次的に処理する場合，問題となるのは対故障性である．Storm はもしデータを処理する Worker が異常終了した場合でも即座に Worker を再起動し，何事もなかったかのように処理を継続できる．その際にもストリームを喪失することがないよう，処理完了を確認するまでデータを配信元で保持・再送する仕組みを備えている（At-Least-Once 処理保証）．また，単純にデータを再送するだけでは同じデータを 2 度以上処理する状況が発生し，誤った集計結果になる可能性がある．この対策として，Storm 上で動作する Trident というフレームワークで重複排除を行っている．Trident では Storm 内を通る個々のデータにユニークな ID を取り付け，同一の ID を 2 度観測した際には不必要な再送が行われたと判断して無視する．再送と重複排除を組み合わせることですべてのデータが 1 度だけ通ることを保証している（Exactly-Once 保証）．

高度な解析処理と大規模データ処理への適用方法

Hadoopは，「単語を数えるためにある」と比喩されるくらい単純な集計処理に強い．しかし近年，大量のデータからより効率的に知見を得るために，高度な統計解析処理や，機械学習を適用したいというニーズが高まっている．統計解析や機械学習のツール群も数多く開発されており，OSSも数多く存在するが，これらは必ずしも大規模データ解析向けに開発されたものではないためHadoopにすぐ適用できるわけではない．ここでは，高度な解析処理を実現するOSSを紹介しながら，大規模データ処理との相性や適用方法などを解説する．

単体動作するデータ解析エンジン

膨大な統計ライブラリと高い利便性により，現在最も利用されている統計パッケージの1つにRがある．RはAT&Tのベル研究所で開発されたSを参考にして作られたOSSである．Rはプログラミング言語の形態をとっており，データの入出力処理から，膨大な数の統計解析用のライブラリの利用，そして可視化処理までも記述できる．一方，既存のプログラミング言語であるPythonで利用できる数値解析用ライブラリのSciPyも人気が高い．SciPyもRと同様に大量の統計ライブラリを提供するが，それ自身もPythonのライブラリであるため，Pythonで書かれた既存のライブラリと同時に，かつ組み合わせて利用できる．

多くの機能を有するRやSciPyとは対照的に，単機能ながらシンプルなエンジンもよく使われる．たとえば国立台湾大学が開発しているlibsvmである．libsvmは，最も有名な機械学習手法の1つであるサポートベクトルマシン関連の手法を実装したOSSであり，機械学習を利用することの多い自然言語処理や画像処理などの分野で長年利用されている．

しかし，これらの統計解析・機械学習ソフトウェアは，いずれも単一のサーバ上で動かすことを想定して作られているため，大規模環境で動かすことは基本的に難しい．

大規模環境向けのデータ解析エンジン

近年では，扱うデータ量の増加に対応できる大規模環境での機械学習やデータ解析ソフトウェアが出てきている．Apache Mahoutもその1つである．Mahoutを利用すると，単一のサーバでは処理しきれなかったペタバイトクラスのデータを利用した機械学習を，Hadoopの上で並列実行することができる．ただし，処理性能の面を優先したい場合は，「メモリを利用した高速化」の節で紹介したSparkを基盤として用いた方が良いと考える．事実，Spark上で動作する機械学習ライブラリのMLlibはMahoutよりも100倍高速ということを謳っている．また，「ストリーム処理」の節で紹介したStormを基盤として利用した，Yahoo!が開発した機械学習ライブラリのSAMOAも，具体的なベンチマーク結果は出ていないものの，高速な機械学習を実現している．

機械学習の分野の中には，オンライン学習と呼ばれる手法がある．一般的な機械学習は，対象となるデータ集合すべてを使って処理を行うが，オンライン学習では，ランダムにサンプリングされたデータのみを使って学習を行う．そのため大規模なデータストアの仕組みは必要なく，学習対象となるデータを貯めずに次々に処理を行っていくストリーム型のフレームワークとは相性が良い．

また，ストリーム型でかつ分散協調して機械学習を行う手法も存在し，これを取り入れたのがNTTとPreferred Infrastructureが共同開発したJubatusである．Jubatusでは，連続してくるデータに対して，分散して動作する各サーバが独立に個々の学習処理を行うように設計しているため，きわめて高い並列度を実現し，単位時間あたりの学習データの処理数も高い．加えて学習処理の結果をサーバ間で合成・共有するため，単一のサーバで学習する場合と比べて複数のサーバの計算能力を活かした学習が行える．この合成・共有処理は低い頻度でしか行わないため，全体の処理性能の低下は限定的である．そのため同じ精度を達成するのに必要な学習時間が少

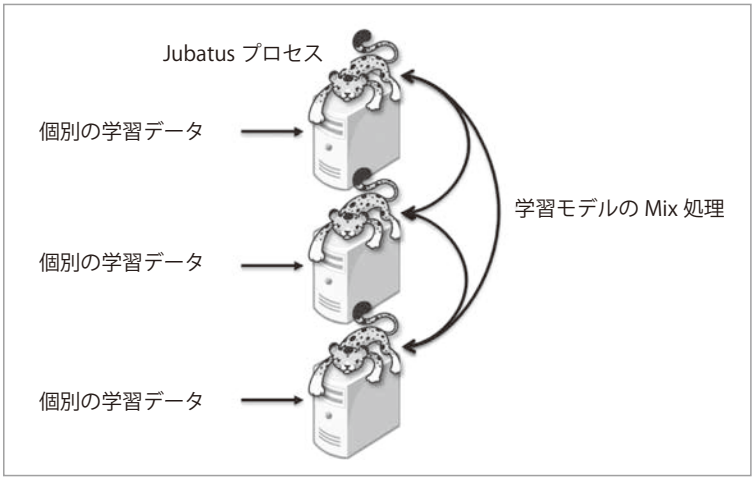

図 -5　Jubatus の処理方式．サーバは学習処理をそれぞれ独立してメモリ上で行い，不定期に内部パラメータ情報をサーバ間で交換する Mix 処理を行う．

なくて済む（**図 -5**）．

長い間大規模データ処理とは関係なく発展してきた機械学習ではあるが，現在では両者が融合し，大規模機械学習という形に変化して発展が進んでいる．たとえば，MapReduce 上の機械学習を推進してきた Mahout も，効率向上のため MapReduce から離れることを宣言したことは記憶に新しい．今後は，Spark や Storm などの新しいフレームワークの上で発展していくことが予想される．

解析処理の生産性向上のための記述言語

従来，Hadoop の MapReduce を利用するためには Java でプログラムを書く必要があった．しかし，Java という言語は，たとえばアクセスログの読み込み 1 つとっても煩雑な記述が必要であり，解析内容を何度も変えて手軽に行うアドホックな解析のプログラムを書くのは効率が悪い．

これに対し，最近では解析処理の生産性を向上できる簡便な言語が開発されている．ここでは，その中の大規模処理専用に開発されたドメイン特化言語について紹介する．

MapReduce 処理を記述する言語として最初に出現したものに Apache Pig がある．Pig Latin[3] という言語でワークフローを記述すると，Java で記述された MapReduce プログラムを生成・実行する．直接 Java で記述する場合と比べて圧倒的に少ない量の記述で実現できるため生産性が高い．この際，何が Map 処理に対応し，何が Reduce 処理に対応するかは隠蔽されるため，利用者が MapReduce を意識する必要はない．その反面，必ずしも最適な MapReduce コードが生成されるとは限らないという欠点もある．しかし多くの場合は多少の計算資源の効率よりもコードを書く人間の人的リソースこそがボトルネックになるため，それを解消できる Pig は，AOL，Twitter，Salesforce，Yahoo!，LinkedIn などを始めとする多くの企業で採用されている．

一方で大規模データ解析のために，検索・集計を行う言語として人気が高いのは，SQL を模倣した言語である．たとえば，Hadoop を動かす上で最もよく使われるドメイン特化言語の 1 つが Apache Hive である．Hive は HiveQL という SQL を模倣した言語で問合せを書くと，MapReduce 処理に変換・実行し，問合せに関する集計処理を行う．Hive も Pig 同様，Map 処理と Reduce 処理がどうなっているか意識する必要はない．SQL を模倣した言語にはほかに，「メモリを利用した高速化」の節で紹介した Spark 向けに Spark SQL が活発に開発されている．Hive も Spark SQL も JDBC（Java Database Connectivity）[☆1] ドライバや ODBC（Open Database Connectivity）ドライバが提供されているため，他の SQL データベースと同様にアプリケーションから利用することができる．

大規模データ解析向け OSS の今後

Hadoop を先駆けとして，大規模データ解析は OSS を中心に，この 10 年で大きく進化してきた．しかし，今では当たり前の技術である MapReduce

☆1 http://www.oracle.com/technetwork/java/javase/jdbc/index.html

や Hadoop も，出現当初は必ずしも研究者たちに好意的に受け入れられたわけではなかった．分散処理基盤の研究開発自体の歴史は長く，企業およびアカデミアの研究者の両方から，MapReduce は過去の技術開発の再発明に過ぎないという指摘も多かった．ところが，Web 関連の企業を中心に，大規模な並列分散処理の要望が出てきたことによって状況は一変した．Yahoo! や Facebook，Twitter など大規模サービスを展開する各社が利用を開始し，また，Cloudera や Hortonworks など，Hadoop や関連ソフトウェアの開発，サポートを積極的に行う企業も多数現れた．結果として，複数の企業が協力する形で Hadoop コミュニティが形成され，デファクトスタンダードの地位を確立していった．現在では，この動きは分散処理技術でビジネスを行う企業にとって無視できない規模となっている．

今までの紹介を見て分かるように，Hadoop に限らず大規模データ処理向け OSS の大半は，企業が主導して開発している．商用製品ではなく OSS として開発することを戦略的に行っていることは注目に値する．そこには，かつてのハッカーの趣味の活動という OSS のイメージはほとんどない．企業が，すべて自分たちで開発して特許やライセンスによって知的財産を守ったりビジネスを行ったりする形から脱却し，OSS を活用して自分たちが実現したい技術開発に注力したり，解析技術によってパートナー企業と新しいビジネスを行う道を選択し始めたということだろう．そのため，コミュニティで中心的な立場に立ち，開発の方針や仕様の策定で主導権を握りたいという思惑も見え隠れする．これも，一種のオープンイノベーション戦略と見ることができるだろう．

大規模データ解析の分野は，まだまだ進化の途中である．今後も技術動向や各社の動向をウォッチし，最新技術の恩恵とビジネスチャンスを逃さないようにしていく必要がある．

参考文献

1）Dean, J. and Ghemawat, S. : MapReduce : Simplified Data Processing on Large Clusters, In Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation, pp.137-150（2004）.
2）Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M. and Vassilakis, T. : Dremel : Interactive Analysis of Web-Scale Datasets, In Proceedings of the VLDB Endowment, pp.330-339（2010）.
3）Olston, C., Reed, B., Srivastava, U., Kumar, R. and Tomkins, A. : Pig Latin : A Not-So-Foreign Language for Data Processing, in Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp.1099-1110（2008）.

（2014 年 10 月 2 日受付）

■ **海野裕也**　unno@preferred.jp

2008 年東京大学大学院情報理工学系研究科修士課程修了，修士（情報理工学）．同年，日本アイ・ビー・エム（株）入社，東京基礎研究所配属．自然言語処理，テキストマイニングの基礎技術の研究に従事．2011 年から現職．Jubatus をはじめ，機械学習や自然言語処理，テキストマイニングの研究開発に従事．

■ **熊崎宏樹**　kumazaki.hiroki@lab.ntt.co.jp

2012 年名古屋工業大学大学院創成シミュレーション工学専攻修了，修士（情報工学）．同年日本電信電話（株）入社．ソフトウェアイノベーションセンタ配属．Jubatus の研究・開発・応用に従事．