4 データクラウド研究の潮流と最新動向

宮崎　純◉奈良先端科学技術大学院大学
鬼塚　真◉NTT サイバースペース研究所

クラウド環境におけるデータ管理とは？

近年，クラウドコンピューティング，Web サーチエンジン，データマイニング，大規模データのアーカイブ，データサイエンスなど，大規模データに対するコンピューティング技術が着目されている．たとえば，Google では 2008 年時点で 1 日に 20PB（20×10^{15}B）規模のデータを分析処理しており[1]，また国内でも NTT ドコモはペタマイニングプロジェクトにおいて大規模データの分析を進めている．このように大規模データを活用して埋もれた知識を発見することによって，企業の経営戦略に活用したり，またコンシューマ向けにはパーソナライズやレコメンデーションによって的確な情報を利用者に提供するサービスに活用することができる．

このような大規模データを管理・活用するための技術は従来も取り組まれており，代表的にはデータを多角的に分析する OLAP（Online Analytical Processing）に関する技術や，コンビニにおける販売データを利用して在庫管理を効率化するといったデータマイニング技術などが挙げられる．しかし，現在のクラウドコンピューティングはスケールメリットを活かすため，PB スケールのデータ規模，あるいは数千台を超えるクラスタ規模に拡大しており，従来の技術が想定し得ない規模に達している．一方で，並列計算という観点ではスパコン系の従来技術も挙げられる．しかし，スパコン系の技術は非常に高価なコンピュータを必要とするため，クラウド環境では多数の安価なコモディティ PC（ノード）をネットワークで接続し，スケールアウトさせることが有力な解決策となってきた．

上記の背景に基づき登場したのがクラウドに関するデータ管理技術（データクラウド）であり，図 -1 にクラウド環境におけるシステム構成の例を示す．

上位層は応用層であり，オンライン系のサービス（たとえば Web メールや SNS など）や，オフライン系の分析処理（利用者のログ解析や PageRank 計算など）がある．中間層は応用サービスを実現するための大規模データを分散処理により分析する層である．下位層はデータ管理層であり，上位の層に対して構造データをアクセスできる機能を提供する分散データストア NoSQL（Not only SQL）[2]や，ファイルシステム相当の機能を提供する分散ファイルシステムなどがある．また，NoSQL はバックエンドとして主にローカルファイルシステム，関係データ

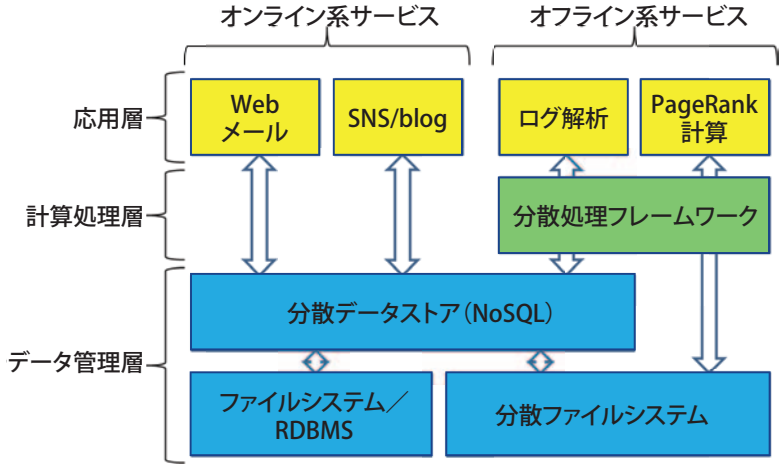

図 -1　クラウド環境におけるシステム構成

ベース管理システム（RDBMS），あるいは分散ファイルシステムを利用して構成される．

本稿では，図 -1 において特に大規模データを扱う NoSQL 系の技術に関してストレージの構成方法と重要な機能について説明し，分散処理系の技術に関しては代表的な技術である MapReduce とその高速化の研究動向を中心に説明する．

分散データストア系の技術

クラウド環境におけるデータ管理層には，従来の RDBMS のような複雑でデータの一貫性に関して厳格なシステムは必須ではなく，単純な CRUD，すなわちデータの作成（Create），読出し（Retrieve），更新（Update），削除（Delete）を高スループットで処理でき，かつ，スケールアウトするシステムが重要である．このような背景から NoSQL と呼ばれるシステムが研究開発されている．

NoSQL は分散システムであり，分散システムにおける CAP 定理と呼ばれる限界の影響を受ける．CAP 定理とは，Consistency（一貫性），Availability（可用性），Partition tolerance（ネットワークの分断耐性）の 3 つの要求を同時に満たすことができない，という定理である．NoSQL に関しては，もしネットワークが分断すれば，一貫性もしくは可用性のどちらかを選択しなければならない，という解釈が相応しいと考えられる．どちらを選択するかはシステムの設計方針やアプリケーションに依存する．

厳格な一貫性が要求される金融関係のアプリケーションには ACID トランザクション，すなわち Atomicity（原子性），Consistency（一貫性），Isolation（分離性），Durability（持続性）のすべての特性を満足する厳格なトランザクションが必須であるが，ACID トランザクションをスケールアウトさせることはきわめて困難である．このため，NoSQL では ACID トランザクションではなく，緩い一貫性の BASE トランザクションが採用されることが多い．BASE とは，Basically Available，Soft-state，Eventually consistent（結果整合性）の頭文字を取ったものであり，basically available とは高可用性，soft-state とは厳密でない緩い状態の許容，結果整合性とは一時的にデータが一貫していないかもしれないが時間が経過すれば一貫した状態になることである．このような緩いトランザクション処理の概念は，分散システムである NoSQL をスケールアウトさせることを容易にする．BASE トランザクションは，特に CAP 定理の可用性がより重要であるアプリケーションに有効である．しかし，NoSQL には BASE トランザクションが必須であるという意味ではない．

以下では，このような NoSQL を実現するための技術の動向について述べる．

NoSQL のストレージ構成

始めにストレージ設計のための構成要素（図 -2 参照）を解説した後で，実際のシステムがどのような技術を組み合わせて構成されているかについて述べる．

大量のデータを多数のノードに分散して管理するためには，P2P（Peer to Peer）で培われたキーバリューストアの技術が利用される．データは，検索条件のためのキーと，コンテンツを表すバリューのペアからなり，キー値から計算される ID により range partitioning，すなわち ID の範囲を区切り，それぞれの範囲に対応するノードにデータが分散される．Consistent hashing では，リング状のオーバレイネットワーク上にノードが配置され，ID はキーのハッシュ値が利用される．探索はキーを利用して行われるが，方法は通常のハッシュ探索と同じである．しかし，ハッシュを利用した方法ではノードにデータが均等に分散される利点はあるが，検索は完全一致のみ可能で，データベースでよく利用される範囲検索ができない．範囲検索を可能にするために，ID としてキー値そのものを利用してデータを分散させる Mercury や，木構造のオーバレイネットワークを利用する BATON があるが，ノード間でデータの偏りが発生するため負荷均衡化が行われる．これらの P2P を利用するキーバリューストアは，ノードの追加や離脱を動的に行うことができる．このため，サービスを継続しながらスケールさせることが可能で

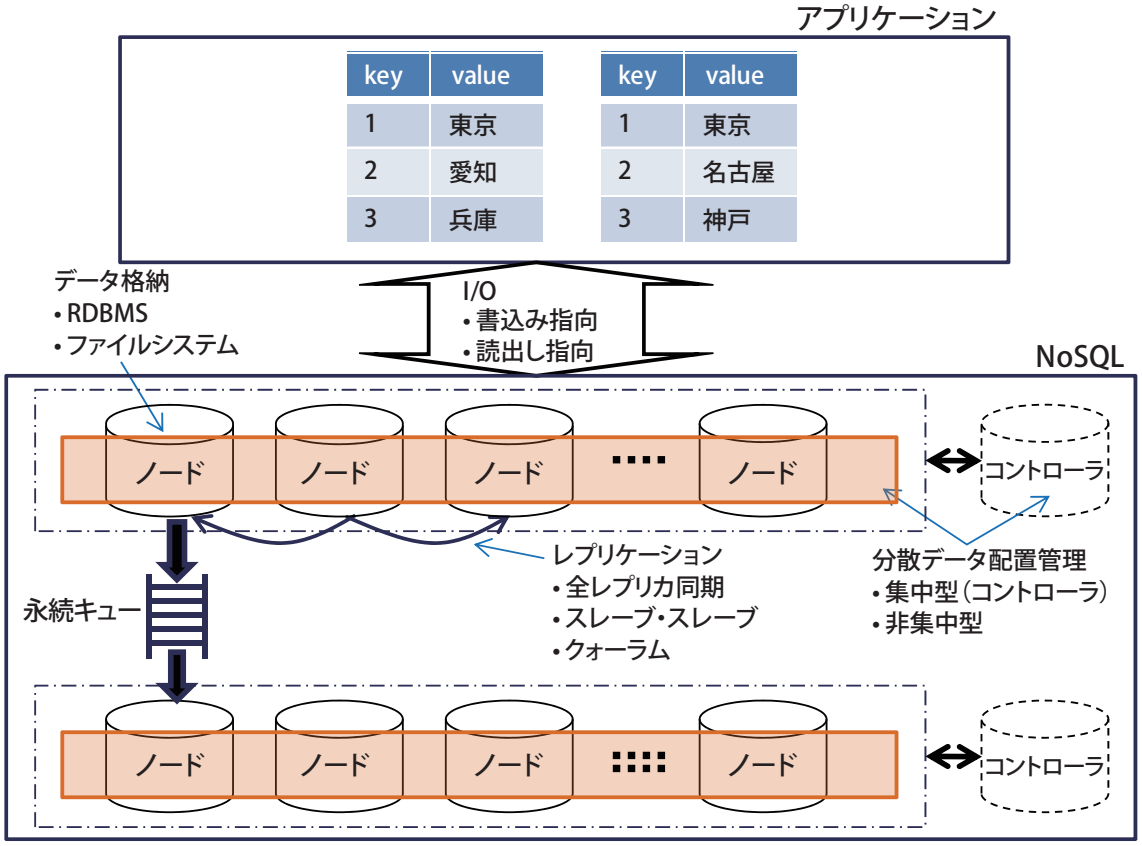

図 -2　NoSQL の構成

あり，非集中型データ配置管理という点でも，分散システムである NoSQL のデータ管理に適している．

そのほかにも，データ格納ノードとは別に，データ配置管理を集中的に行うコントローラを設置する方式もある．コントローラではデータのキー値から計算される ID の範囲とそれに対応するノードの対応関係を表すマップが格納される．CRUD 要求は，コントローラ中のマップを利用して，適切なデータ格納ノードに要求をルーティングすることで実現される．ID 計算にはハッシュやキー値そのものを利用する場合があるが，後者の場合は範囲検索が可能であり，検索条件に合致するデータを格納しているノード集合に適切に要求をルーティングすることで実現される．また，コントローラにファイル属性やディレクトリ構造等を保持するディレクトリメタデータを置くことで，システム全体を巨大な分散ファイルシステムとしてサービスを提供することもでき，Google の GFS（Google File System）はその代表例である．

多くの NoSQL では，キーとバリューのペアというきわめて簡単なデータ形式により，大量の分散データの管理が容易となることから，キーバリューペアの集合をデータベースにおけるテーブルとして扱っている．しかしながら，ユーザの観点からは従来の RDBMS の複数属性からなるタプル形式の方が利用しやすい．そこで，1 つのテーブルをタプルの 1 つの属性として見なし，複数のテーブル間で共通のキーを持つデータをタプルとして扱う手法が採られる．これは RDBMS におけるカラム指向ストレージの手法と同じである．もしテーブル中に 2 つ以上の同一キーを持つデータがある場合は，非正規形のタプルとして扱うこともでき，柔軟なデータ構造を作成することが可能である．カラム指向のキーバリューストア以外にも，従来の RDBMS と同様のロー指向ストレージを用いて NoSQL を構成する選択肢もある．

各データ格納ノード内でのデータの管理には，データを RDBMS 中に格納したり，tablet と呼ばれる数 MB ～数十 MB のファイルに分割して入れてファイルシステムに格納したりする方法がある．特に後者の tablet による方法では，データ格納ノード間で，tablet 単位でデータを移動させることによりノード間のデータ移動のオーバヘッドを小さくでき，負荷均衡が容易となる．

I/O の設計は，NoSQL の性能に大きく影響する．また，NoSQL を利用するアプリケーションによっても設計方針が異なる．データ更新が主の場合は，書込みがランダム I/O とならないよう，データを上書きせずに更新ログをシーケンシャル I/O として追記するログ構造化ファイルシステムに類似した書込み指向 I/O が採用される．しかしながら，データ読出しの際には最新のデータを再構成するオーバヘッドが生じる．一方，データ読出しが主の場合は，読出しがシーケンシャル I/O となるよう，データ更新時にはデータの上書きを行う読出し指向 I/O が採られる．

NoSQL を構成する場合，これらの選択肢が組み合わされる．Google Bigtable や Apache HBase は大規模な OLAP 等のバッチ処理を目指しており，OLAP 処理に有効なカラム指向のキーバリューストアを利用し，大量のデータ書込みに対処するため I/O は書込み指向で，更新データはシーケンシャルに追記される．それぞれのデータは分散ファイルシステム上に tablet として格納される．Amazon Dynamo や Dynamo のアイディアを利用した Apache Cassandra は，電子商取引や SNS で利用されるため，より可用性の高い非集中型データ管理方式である consistent hashing を利用したキーバリューストアで構成される．Dynamo はデータ格納に RDBMS を用いて読出し指向 I/O となっているが，Cassandra は tablet を利用した書込み指向 I/O が採られる．Yahoo! PNUTS は，SNS 等のユーザごとのログインやプロファイルの読出し処理に注目しており，RDBMS と同様にロー指向の構成であり，読出し指向 I/O が採用されている．タプルの格納は，タプルの分散方法に応じて tablet と RDBMS が使い分けられ，集中型のデータ配置管理により管理される．

構成要素	データ配置管理	テーブル構成	データ格納	I/O
選択肢（利用システム）	集中型 (1, 4)	カラム指向 (1, 2, 3)	ファイルシステム/tablet (1, 3, 4)	書込み指向 (1, 3)
	非集中型 (2, 3)	ロー指向 (4)	RDBMS (2, 4)	読出し指向 (2, 4)

1: Google Bigtable, Apache HBase
2: Amazon Dynamo
3: Apache Cassandra
4: Yahoo! PNUTS

表 -1　NoSQL の構成要素とその選択肢

以上を表 -1 にまとめておく．

＊ NoSQL の高可用化・一貫性

NoSQL は，多数のノードから構成されるため，ノード数が増えるにつれシステム障害の確率も増加する．ノード故障に対するデータの保護とサービスの継続のために，高可用化が求められる．高可用化のためには，レプリケーション，すなわち複数のノードにデータの複製を置く手法が利用される．データのコピーはレプリカと呼ばれ，もしあるノードが故障しても，データは失われず，利用可能なレプリカを使ってサービスを継続できるため可用性が高くなる．しかし，データ更新の際にはレプリカ間でデータの一貫性を保つ必要がある．銀行のような厳格なデータの一貫性が求められる場合，すべてのレプリカが一貫している one-copy serializability が要求されるが，システムをスケールアウトさせることは困難である．このため，Web メール等の応用では一貫性の条件を緩め，結果整合性を採用することで，スケールアウトを達成することが一般的である．

レプリケーションにはいくつかの手法があり，データの書込みの際に，(1) すべてのレプリカに同期書込みをする手法，(2) マスタレプリカに書込み，スレーブのレプリカへはマスタから非同期に更新ログを伝播する手法，(3) クォーラムと呼ばれる一部のレプリカに同期書込みをする手法がある（表 -2 参照）．(1) は Bigtable や HBase で利用され，レプリ

レプリケーションの方式	全レプリカ同期書込み	マスタ・スレーブ	クォーラム
利点	•データー貫性が強い	•書込み応答時間が速い	•データー貫性が強い •書込み／読出し応答時間を調整可能
欠点	•書込み応答時間が遅い	•データー貫性が弱い	•制御が複雑

表 -2　各レプリケーション方式の特性

カ間で個々のデータに対して同期して書き込むことでデータごとの一貫性を保持できる利点があるが，すべてのレプリカへの書込み完了を待つため，データセンタをまたがるような遠方へのレプリケーションには向かない．(2) は PNUTS で利用され，書込み完了までの時間は短くなるが，更新ログが完全に伝播するまでは，古いデータを読み出す可能性がある．PNUTS では timeline consistency と呼ばれる一貫性のモデルを採っており，データの古さを許容するローカルレプリカからの高速読出しや，時間がかかるが最新のデータを保証する読出しなどを提供する API があり，アプリケーションから選択できる．更新ログの伝播が完了するまで，ログもどこかのノードに格納され，ノード故障によりデータを失う可能性がある．これを避けるためにデータベースで培われたバッチ処理のための永続キューを利用して，確実にログを伝播させる手法も取り入れられている．(3) は，すべてのレプリカでなく，任意の過半数以上のレプリカ (write クォーラム) に対して，データにタイムスタンプを追加して同期書込みを行う手法である．読出しの際は古い可能性のあるローカルのレプリカを読み出すこともできるが，read クォーラム中のすべてのレプリカを読み出し，タイムスタンプの最も新しいものを採用することで最新のデータを読み出すこともできる．読出しや書込みの頻度を考慮して，読出し性能重視あるいは書込み性能重視の最適化も可能である．クォーラムは Dynamo や Cassandra で利用されている．

NoSQL ではデータの一貫性の維持も重要である．広域分散システムですべての種類のデータ更新の一貫性を保つのは困難であるため，多くのシステムではタイムスタンプを利用した単一データのアトミック更新を基本としている．しかし，実際のアプリケーションでは，一貫性を保ったまま複数のデータの更新を行う操作，すなわち ACID トランザクションが必要な場合も多い．Google のサービスである Megastore[3] では，すべてのデータに対してではなく，エンティティグループと呼ばれる密な関係のデータの集合に分割し，個々のエンティティグループ内部のみ ACID トランザクションを実現している．Megastore で興味深いのは，データ更新時のログ書込みの際に複数のノードにログを書き込むが，非集中型の Paxos アルゴリズムを改良して通信回数を削減した手法を，ログ書込み位置を決定する際の合意形成に利用している点である．データの読出しに関しても，Paxos によるデータ更新を監視することにより，故障が起きない限りローカルのレプリカから最新状態のデータを素早く読み出すことができる．

＊ NoSQL の二次索引

データベースで重要な機能として，条件に合致するデータを効率よく読み出すための二次索引があるが，NoSQL ではキーバリューペアのキーからでしかデータを検索できず，バリュー値の条件でデータを効率よく読み出す際には，キーとバリューを逆にしたペアのテーブルを作成し，転置索引と同様な機能をアプリケーション側で作成，維持する必要があった．しかし，最近，Cassandra や Megastore のように二次索引がサポートされる NoSQL が増加してきた．たとえば Megastore では，エンティティグループ内のローカルデータには一貫した二次索引，さらにエンティティグループをまたがるグローバルデータには結果整合性のある二次索引がサポートされる．

分散処理系の技術

これまで述べてきた分散データストア系の技術に加えて，大規模データを対象として高速にマイニングなどの分析処理を実行する分散処理技術が進展してきている．この分散処理技術として代表的なものが Google の MapReduce であり，Microsoft も対抗

技術として Dryad を提案し，研究プロジェクトとして取り組んでいる．MapReduce については多くの雑誌でも取り上げられておりご存知の読者の方も多いと思われるが，改めて MapReduce の基礎について説明した後に，大規模データ処理において重要な観点である高速化にかかわる研究を中心に最新動向についてまとめる．

＊ MapReduce とは?

MapReduce は Google で開発された分散処理のプログラミングモデルおよび分散処理システムの両方を指す．また Apache Hadoop プロジェクトにおいても MapReduce 実装が公開されているため，Web 系の企業を始めとして大手の企業も含めて，MapReduce を利用した大規模データの活用に取り組むようになってきている．MapReduce は，本来 Web 検索エンジンのバックエンドにおいて，大規模データの処理（PageRank 計算・転置索引構築）で用いるために開発されたシステムである．プログラミングインタフェースの観点から見ると，開発者は map 関数と reduce 関数を実装するだけで分散プログラムを開発することができ，分散処理特有の複雑さ（処理をどう分散するか，あるいはコンピュータやネットワーク障害時にどう対処するか）がプログラマから隠ぺいされるという特徴を持つ．また，分割統治法に基づいたアプローチによって大規模データを分割することで処理を分散し，分散した処理の結果を束ねることで最終結果を得ている．具体的には，分割されたデータの各入力レコードに対して map 関数が呼び出され，結果を束ねる際に reduce 関数が呼び出される仕組みになっている．たとえば文書群を対象に単語の頻度計算をする MapReduce プログラムでは，map 関数においては文書から単語を抽出して（単語，頻度 1）のキーとバリューのペアを出力し，reduce 関数では単語ごとに頻度を積算するようプログラムを記述する．

＊ MapReduce による分析応用例

ここでは標準偏差の計算および PageRank の計算の具体例を用いて，MapReduce プログラムの設計方法と処理の高速化の方法について説明する．

【標準偏差の計算】

標準偏差 σ は以下のように定義される．

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2}$$

ここで，$x_1, \cdots x_n$ は入力値，n は入力データ値の件数，μ は入力データ値の平均値である．この標準偏差の計算はどのように MapReduce 上で実装することができるだろうか？　直観的な 1 解法は，2 つの MapReduce ジョブを用いる方法である．1 つ目のジョブにおいて入力値の総和と件数を積算して平均値 μ ＝ 総和／件数を計算し，2 つ目のジョブにおいて平均値を用いて上記の式の右辺全体を演算することで標準偏差を計算することができる．さらに，以下に述べる 2 つ目の解法では，1 パスのデータ処理の方法を用いることで 1 つのジョブで標準偏差を計算することができる．

$$\sigma_{ab}{}^2 = \frac{n_a\sigma_a{}^2 + n_b\sigma_b{}^2}{n_a + n_b} + n_a n_b\left(\frac{\mu_b - \mu_a}{n_a + n_b}\right)^2$$

$$\mu_{ab} = \frac{n_a\mu_a + n_b\mu_b}{n_a + n_b}$$

$$n_{ab} = n_a + n_b$$

ここで，a と b は入力データ値の部分集合を表し，ab は a と b の入力データの和集合を表す．上記の 3 つの式を用いることで，入力データを分割した単位ごと（上記の式では a，b に該当）に局所的な標準偏差 σ，平均値 μ，値の件数 n を計算した後に，それらの結果を束ねることで入力データ全体の標準偏差を得ることが可能となる．

このように，MapReduce の処理における高速化の観点の 1 つはジョブ数の削減であり，ジョブ数の削減に応じて大規模な入力データをアクセスする回数を削減することができる．ジョブ数を減らす課題における技術動向としては，学習系の研究においては代表的な 10 種類の機械学習のアルゴリズムが 1 つの MapReduce ジョブによる 1 パスのデータ処理

で実現できることが報告[4]されている．また並列プログラミング言語の研究においては，リスト準同型の定理を用いることで計算処理を分散プログラム化する研究や，関数の連続した呼出しを融合するなどの効率化に関する研究[5]がなされている．

【PageRank の計算】

2つ目の例は，Web ページが利用者にアクセスされる確率を表す PageRank の計算であり，PageRank は次の式に基づき計算される．

$$v = (1-c)Av + cu$$

ここで，v は全 Web ページの PageRank ベクトル，A は Web ページのグラフ構造を表す隣接行列，u は利用者がランダムにアクセスする Web ページを表すベクトル，c は利用者が Web ページ内の URL をクリックせずランダムに選択した Web ページに遷移する割合を表している．上記の式を収束するまで繰り返して実行することで，Web ページが利用者にアクセスされる確率である PageRank ベクトル v を得ることができる．直観的には PageRank は Web ページの重要度を表していると言える．

この PageRank ベクトルはどのように MapReduce 上で計算することができるだろうか？　直観的な1解法は，上記の式の1回の計算を1つの MapReduce ジョブに割り当ててジョブを繰り返し実行する方法である．この方法に対する改善方法の1つの例として PEGASUS の方法が挙げられる．PEGASUS の方法では，Web ページのグラフ構造を表す隣接行列をブロックに分割し，ブロック内の行列演算を1つの MapReduce ジョブに割り当てることで，ジョブ数を削減する（図 -3 参照）．また，PageRank のようなグラフ構造に特化した処理をモデル化する技術として，Pregel が提案されている．Pregel では Web のグラフ構造を直接モデル化し，グラフのノードにおいてどのように処理を実行するかをプログラミングすることで，大規模グラフにおける処理を簡潔に記述することができる．

＊ MapReduce 系技術の研究動向

2つの例題を用いて MapReduce 処理の高速化の具体例を説明したが，他の研究動向を含めて代表的な技術的観点を以下に整理する．

【ジョブ回数の削減】

機械学習アルゴリズムの1パスのデータ処理化[4]のほかに，決定木・回帰木の学習処理において学習対象データへの複数回のアクセスを1回に集約させることで，MapReduce のジョブ回数を削減する方法として PLANET が提案されている．PLANET では幅優先探索の順で学習木を構築することで，学習木における同一の深さの全ノードの処理を1回の MapReduce ジョブで実行することを特徴としている．

【Shuffle データ量の削減】

map タスクの結果を reduce タスクに送信（Shuffle）するデータ量を削減する方法の典型例として，reduce 関数に相当する処理を map 関数内で実行する local aggregation による方法[7]がある（MapReduce フレームワークで提供される combine 関数も local aggregation の1種である）．また，PageRank の計算や k-means の処理のように，変化しない入力データに対して複数回の MapReduce ジョブを繰り返し処理する問題を対象として，データのキャッシュを利用する技術 Haloop が提案され

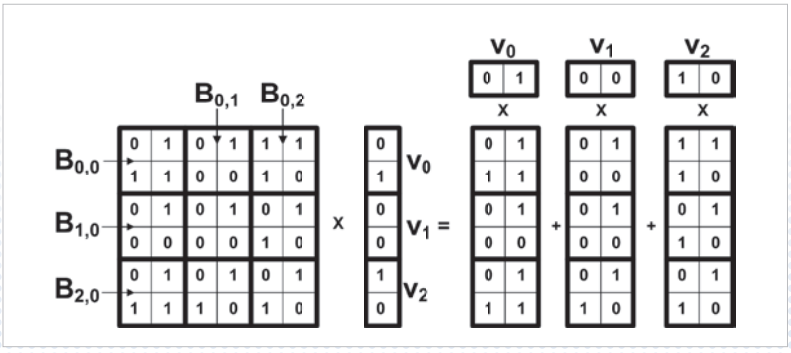

図 -3　PEGASUS での処理方法
$B_{i,j}$ は 2 × 2 の場合の行列ブロック，V_i はベクトルブロックを表し，ブロック単位で行列演算が行われる（文献 6）より引用）．

ている．Haloop では，map タスクあるいは reduce タスクを実行する各計算機に入力データを事前に配布しインデックスを構築しておくことで，各ジョブの実行時に HDFS アクセスあるいは Shuffle を実行することなく事前配布したデータから必要なデータを利用することを特徴とする．

【データ格納構造の最適化】

RDBMS の技術を用いてデータの格納構造を最適化する工夫もある．Hive の RCFile フォーマットでは，カラム指向ストレージの技術を利用することで無駄なカラムのアクセスに伴う I/O コストを削減することが可能である．また HadoopDB では，MapReduce のデータ格納層において分散ファイルシステムの代わりに RDBMS を用いることで，データアクセスの高速化を実現している．

【複数処理を対象とした最適化】

複数の MapReduce ジョブにおいて，map タスクの入力および出力データを共有することで，データ読出しコストおよび中間データ量を削減する高速化手法として MRShare が提案されている．また SQL 相当の処理を MapReduce 環境で実行する場合に，複数の SQL に共通する結合処理を高速化する方法として，Hadoop++では結合する複数テーブルを対象として，結合キーが一致するレコードを事前に同一パーティションに格納することで Shuffle を利用しない結合処理を実現している．ほかに，RDBMS で利用されていたセミジョイン等の技術を活用する方法や，ラグランジェの未定乗数法を用いて Shuffle コストの最適解を探索する方法[8]が研究されている．

【適用領域に特化した処理モデル】

適用領域に特化した処理モデルとしては前述のグラフ処理に特化した Pregel のほかに，データフローに基づいたプログラムを処理する Dryad や，SQL に近い記述のプログラムを MapReduce に変換して処理する Hive や Pig などが挙げられる．

【その他】

MapReduce では，map タスク群を実行するフェーズ完了後に reduce タスク群を実行するフェーズが実施されるため，分析処理の種類によっては計算機資源の利用待ちが発生して資源が有効に活用できないケースがある．MapReduce Online では，パイプライン並列化を MapReduce に導入することで，資源利用を効率化している．また，異なる性能のコンピュータが混在するクラスタ環境を対象とした研究としては，コンピュータの処理性能に応じて reduce タスクの終了時刻を見積もることで，処理の遅いタスクを正確に発見して効果的に投機実行を行う LATE スケジューラがある．さらに，最新ハードウェアを活用する技術として，GPU 環境を対象に MapReduce を設計したシステム，Mars がある．

データクラウドを研究するには？

NoSQL 系の技術に関しては，数多くの文献が存在するが，概要として文献 2)，先進的な NoSQL の取り組みとして Megastore の文献 3）を挙げておく．MapReduce 系の技術に関しては，参考になる文献として文献 7）が挙げられる．この文献は転置ファイル構築などのテキストの統計処理にフォーカスしているが，MapReduce 上でのアプリケーションのデザインパターンや RDBMS における結合処理も含んでおり，MapReduce の応用例に詳しいので参考にされたい．

今後の研究の方向性としては，NoSQL に関しては，より高性能，より高可用性を実現するための洗練された分散アルゴリズムの開発が挙げられる．一概に性能を求めるのでなく，たとえばユーザの観点から性能の保証を実現する方法についてである．また，単なる CRUD の処理だけでなく，それ以外の重要な機能を発見し実現することも課題である．さらに，NoSQL ではクラスタ規模が巨大であることから，消費電力と性能をバランスさせるための技術開発等が挙げられる．

一方，MapReduce に関する今後の研究の方向性として，格納構造の最適化や複数の分析処理を対象としたクエリ最適化などの RDBMS 技術の適用による MapReduce 処理の高速化は今後も続くと考え

られる．また，NoSQL 系と同様に MapReduce 環境においても消費電力量を最小化するといった省電力化の取り組みなどが発展すると考えられる．さらに，プログラムの生産性の向上の課題も挙げられる．先の標準偏差の計算の例にあるように，対象となるアルゴリズムを 1 パスのプログラムに変換することが容易ではないといった課題，あるいは処理を高速化する MapReduce のデザインパターンを適用した MapReduce プログラムを自動生成するといった課題が研究対象として挙げられる．

参考文献

1) Dean, J. and Ghemawat, S. : MapReduce : Simplified Data Processing on Large Clusters, In CACM, Vol.51, No.1 (2008).
2) Agrawal, D., Das, S. and Abbadi, A. E. : Big Data and Cloud Computing : New Wine or Just New Bottles? : Tutorial Slides, In VLDB (2010).
3) Baker, J. et al. : Megastore : Providing Scalable, Highly Available Storage for Interactive Services, In CIDR (2011).
4) Chu, C., Kim, S., Lin, Y., Yu, Y., Bradski, G., Ng, A. and Olukotun, K. : Map-reduce for Machine Learning on Multicore, In NIPS (2006).
5) 胡　振江，岩崎英哉：スケルトン並列プログラミング，情報処理，Vol.46, No.10 (Oct. 2005).
6) Kang, U., Tsourakakis, C. and Faloutsos, C. : PEGASUS : A Peta-Scale Graph Mining System Implementation and Observations, In ICDM (2009).
7) Lin, J. and Dyer, C. : Data Intensive Text Processing with MapReduce, Morgan & Claypool (Oct. 2010).
8) Afrati, F. N. and Ullman, J. D. : Optimizing Joins in a Map-reduce Environment, In EDBT (2010).

(平成 23 年 1 月 25 日受付)

宮崎　純 (正会員) ▮miyazaki@is.naist.jp

奈良先端科学技術大学院大学情報科学研究科准教授．1992 年東京工業大学工学部情報工学科卒業．1997 年北陸先端科学技術大学院大学情報科学研究科博士後期課程修了．博士 (情報科学)．2003 年より現職．

鬼塚　真 (正会員) ▮oni@acm.org

日本電信電話 (株) サイバースペース研究所主幹研究員 (特別研究員)．博士 (工学)．1991 年東京工業大学工学部情報工学科卒．2000 ～ 01 年ワシントン州立大学客員研究員．