情報処理学会論文誌　データベース　Vol.8　No.1　73–87 (Mar. 2015)

Apache Hiveを用いたスケーラブルな機械学習機構の構築

油井 誠[1,a)]　小島 功[1,b)]

受付日 2014年9月20日, 採録日 2014年11月5日

概要：我々は Apache Hive 上で動作する機械学習ライブラリ Hivemall をオープンソースソフトウェアとして公開している．Hivemall はオープンソースの機械学習フレームワークとしてデータ量に対するスケーラビリティが最も高いものの 1 つであり，Hadoop Distributed Filesystem（HDFS）に格納されたデータを入力とした機械学習処理を効率的に扱えるという特徴から Hadoop/Hive に精通する開発者やデータ分析の専門家から注目を集めている．本稿では，Hivemall によるスケーラブルな機械学習を実現するうえで得られた実践的な知見，およびその実現手法を述べる．KDD Cup 2012, Track 2 の広告クリックスルー率の予測タスクを用いた評価実験により，学習速度に定評のある State-of-the-art の機械学習フレームワークに対して Hivemall がより短い学習時間で同等以上の予測精度を出せることを示し，さらに計算ノードの追加によって学習時間を短縮できることを示す．

キーワード：機械学習，オンライン学習，Hadoop，MapReduce，確率的勾配降下法，ロジスティック回帰，オープンソースソフトウェア

Building Scalable Machine Learning Framework on Apache Hive

Makoto Yui[1,a)]　Isao Kojima[1,b)]

Received: September 20, 2014, Accepted: November 5, 2014

Abstract: We have released a machine learning library for Apache Hive, named *Hivemall*, as an open source software. Hivemall is one of the most scalable machine learning frameworks avaiable as an open source software and is getting attention from data scientists and developers who are familiar with Hive/Hadoop because Hivemall is well suited for analyzing data on the Hadoop distributed file system (HDFS). In this paper, we present practical findings in achieving scalable machine learning with Hivemall and explain the implementation details. We conducted a series of experimental evaluations using a commercial advertisement dataset provided in the KDD Cup 2012, Track 2. The experimental results show that our scheme has competitive classification performance and superior training speed compared with state-of-the-art scalable machine learning frameworks for the regression task. We also show the scalability of Hivemall to the computing nodes through an experiment.

Keywords: machine learning, online learning, Hadoop, MapReduce, stochastic gradient descent, logistic regression, open source software

1. まえがき

個人の検索履歴や商品購入履歴など膨大な情報をマーケティング活動や広告クリック率の改善に利用する取り組みが国内外の事業者で進められている．膨大なデータを集約して分析するとき，従来は商用の並列データベースである Teradata や Netezza などのデータウェアハウスアプライアンスを利用することが一般的であったが，近年では非構造化データの扱いに適するオープンソースの MapReduce [1] 実装の Apache Hadoop [2] と Hadoop に基づくデータウェアハウス環境である Apache Hive [3] の採用が広がっている．並列データベースではデータの書き込み時にスキーマに基づいた物理的なデータ配置や索引付けが行われる

[1] 独立行政法人産業技術総合研究所情報技術研究部門
Information Technology Research Institute, National Institute of Advanced Industrial Science and Technology, Tsukuba, Ibaraki 305–8568, Japan
[a)] m.yui@aist.go.jp
[b)] isao.kojima@aist.go.jp

（Schema on Write）のに対して，Hive/Hadoop はデータ投入時は分散ファイルシステム HDFS [4] にファイルをブロック化して書き込むだけで，データの読み出し時にデータのスキーマを解釈する（Schema on Read）．一般的に Schema on Write のストレージフォーマット [5], [6] でも TSV/CSV のような単純なフォーマットに比べると複雑であり，読み込み時の負荷は依然存在する．特に，索引を利用しない全量のシーケンシャルスキャンを前提とするような処理では，両ストレージ戦略で読み込み負荷に大差が生じないのに対して，書き込み負荷は Schema on Read が大幅に有利である[*1]．この Schema on Read という Hive の特徴と耐障害性のある数千台まで高いスケーラビリティを確保可能であるという HDFS の性質は，Volume（データ量），Variety（データの種類），Velocity（データの生成速度）の面で管理が困難なデータを扱ううえで有利に働くことから，Hadoop とその分散ファイルシステムはビッグデータの蓄積場所としてデファクトスタンダードとなりつつある．

Hadoop の普及と HDFS に蓄積されたデータの増加は，HDFS に蓄積されたビッグデータに対して高度なデータ分析（機械学習やデータマイニング）を行う需要を喚起している．巨大なデータを扱う国内外のサービス事業者で Hadoop を利用したデータ蓄積からデータ解析のプロセスが一般化しており [8], [9]，HDFS 上に格納されたデータの局所性を考慮する形で効率的な機械学習を実現する機械学習フレームワークが求められている．

こうした需要に応え，我々は Apache Hive 上で動作する機械学習ライブラリ Hivemall を開発し，オープンソースソフトウェアとして GitHub 上に公開している [10]．Hivemall の開発では，1) SQL 類似クエリを介して対話的に使用できること，2) データストアとして HDFS に対応すること，3) データ量に対するスケーラビリティを有することの 3 つを満たすことを開発要件としている．そのうえで，問合せ処理の並列実行や大規模データ処理に不可欠な耐障害性といった要件を満たすインフラとして Apache Hive を採用している．

すでに Hadoop 上で動作する代表的な機械学習フレームワークとして Apache Mahout [11] が存在するが，現在の Mahout（version 0.9）には，1) 本格的に利用するのには API を呼び出すプログラミングを行いプログラムをコンパイルする手順を踏む必要がある，2) クラス分類を例にとると Random Forest 以外のアルゴリズム（たとえば，Passive Aggressive [12] やロジスティック回帰）では MapReduce が利用されていない逐次処理の実装になっていて並列化がなされていないなどコード品質にばらつきがある，3) 最先端のアルゴリズムがサポートされていない（オンラインクラス分類を例にとると Passive Aggressive までがサポートされているが，Confidence Weighted [13] などの最新の学習アルゴリズムがサポートされていない），といった問題がある．これに対して，Hivemall は次の特徴を有する．

- SQL に類似したクエリによってインタラクティブに機械学習を用いたデータ分析を試行できる．
- データ量に対するスケーラビリティが高い．Hive の並列データ処理機構を利用して機械学習を効率的に実行する．
- Confidence Weighted [13], Adaptive Regularization of Weight Vectors [14], Soft Confidence Weighted [15] など最新のオンライン機械学習の研究成果を取り入れている．

こうした特徴を有することから Hivemall は国内外の開発者やデータ分析の専門家から注目を集めるに至っている．

本稿では，Hivemall によるスケーラブルな機械学習を実現するうえで得られた実践的な知見，およびその実現手法を述べる．ここでスケーラブルとは，1) 訓練データの大きさを示す指標である訓練事例数および特徴次元数について取扱い可能な大きさに上限が発生しないこと，2) 計算リソースの追加に応じて学習器の学習時間を短縮できることを指す．これまでの研究では機械学習を集約関数として実装することが定石となっていた [16] が，Hivemall ではユーザ定義のテーブル生成関数を用いることで，より関係演算に適した形で機械学習の一連のフローを取り扱うことを実現した．KDD Cup 2012, Track 2 の広告クリックスルー率の予測タスクを用いた評価実験により，学習速度に定評のある最先端の機械学習フレームワークである Vowpal Wabbit [17] と Bismarck [18] に対して，Hivemall がそれぞれ 3.46 倍と 8.42 倍短い学習時間で同等以上の予測精度を出せることを示し，さらに Hivemall を利用した機械学習が計算リソースの追加によって学習時間を短縮できることを示す．

本稿の構成は次のとおりである．2 章で Hivemall でも採用しているオンライン学習手法の基本的な事項について説明し，その並列処理手法について述べる．3 章で MapReduce 上でスケーラブルな機械学習を実現するうえでの関連研究を述べる．4 章で Hivemall の構成とその特徴的な機構を示し，5 章で Hivemall で採用した Hive により大規模機械学習を実現するうえでの諸技法を述べる．6 章で提案手法を評価し，7 章でまとめる．

2. 勾配降下法とオンライン学習

教師あり学習の代表的な手法として，Support Vector Machine（SVM）[19] や勾配降下法（Gradient Descent）を

[*1] 大量データを扱うデータウェアハウスではスケーラビリティに劣る索引によるアクセスメソッドやランダムアクセスを避けることが基本であり，関係データベースを利用したデータウェアハウス処理に Schema on Read を採用する研究も存在する [7].

用いた経験損失の最小化手法がある．

一般的に機械学習タスクは，特徴ごとの重みベクトル w の最適化問題ととらえることができる．$D = \{x_i, y_i\}_i^n$ という訓練データセットについて，i 番目の訓練例について x_i を特徴ベクトル，y_i を目的のラベル，ℓ を損失関数とすると，教師ありのクラス分類器は経験損失を最小化する $f : X \to Y$ を導出することが目標となる．ここで，関数 f は学習モデルである特徴ごとの重み w を入力にとり，経験損失 L は次のような形で表される（式 (1)）．

$$L = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i; w), y_i) \tag{1}$$

バッチ学習である勾配降下法は，損失関数 ℓ の傾き ∇（一階微分）から関数の最小値を探索するものであり，各エポック t ごとに次のようにすべての訓練例を利用して重みベクトルを計算する．ここで，η_t は t における学習率（$\eta_t > 0$）である．

$$w_{t+1} = w_t - \eta_t \frac{1}{n}\sum_{i=1}^{n} \nabla\ell(f(x_i; w_t), y_i) \tag{2}$$

一方で，オンライン学習手法である確率的勾配降下法（Stochastic Gradient Descent（SGD））では，次のように重みベクトルを各訓練例ごとに更新する．

$$w_{t+1} = w_t - \eta_t \nabla\ell\left(f(x; w_t), y\right) \tag{3}$$

バッチ学習の勾配降下法がモデルの更新にすべての訓練例を利用するのに対して，確率的勾配降下法は各訓練例だけを見てモデルを更新するため，訓練例の数に対して線形の訓練時間で学習ができるという利点がある．このため，大規模の機械学習ではバッチ学習アルゴリズムである勾配降下法などを近似する確率的勾配降下法などのオンライン学習アルゴリズムが採用されることが多い [20], [21]．

確率的勾配降下法の学習が収束するためには，学習率 η_t について次の 2 つの条件を満たす必要がある [22], [23]．

$$\sum_t \eta_t = \infty \tag{4}$$

$$\sum_t \eta_t^2 < \infty \tag{5}$$

式 (4) は解から遠い初期点からスタートしても解に収束するために必要である．また，確率的勾配降下法では各ステップの更新幅が大きすぎては overshoot を繰り返して学習が収束しないことがあるが，エポック t が進むほど解に近づいていくことを保障するために式 (5) の条件が必要である．このため，学習率 η_t は t が増加するごとに減少する（$\eta_t \to 0$（$t \to \infty$））ものが採用されることが多い．

2.1　マージン最大化を行うオンライン学習アルゴリズム

訓練例の数が無限かあるいは不定という設定のオンライン学習に確率的勾配降下法を利用した場合，学習率 η の設定が困難であるという問題があり，学習率を用いずに直接データを線形分離する識別関数を求めるアルゴリズムである Passive-Aggressive（PA）[12] やその発展系 [13], [14], [15] が好んで利用される．これらのアルゴリズムの目的関数はマージン最大化によってデータの識別平面を直接求めるものであり，その代表例である PA はヒンジ損失の最小化を行うため，オンライン Support Vector Machine（SVM）とも評される．

Confidence Weighted（CW）[13] は重みベクトルの分散を考慮してモデルの更新を行う．CW では，重みベクトル $\mathbf{w}$ にガウス分布 $N(\mu, \mathbf{\Sigma})$ を導入し，平均と共分散を同時に更新していく．データを 1 つ受け取るたび（特徴ベクトル $\mathbf{x}_t$ とラベル y_t），式 (6) の更新式に従いパラメータを更新する．ここで，μ はその時点で最も尤もらしい重みベクトルであり，$\mathbf{\Sigma}$ はその重みにどれくらい自信があるのかを示す共分散行列である．CW は分散の大きいパラメータはまだあまり自信がないと見なして更新時に思いきって大きめに更新する．逆に分散が小さいパラメータはある程度正確に学習できていると見なして小さめに更新する．こうした特徴から学習の収束が早く，数少ないイテレーションで学習が収束に近づくといった特徴がある．

$$(\mu_{t+1}, \mathbf{\Sigma}_{t+1}) = \arg\min_{\mu,\mathbf{\Sigma}} KL(N(\mu, \mathbf{\Sigma})|N(\mu_t, \mathbf{\Sigma}_t)) \\ s.t. \quad Pr_w(y_t\mathbf{w} \cdot \mathbf{x}_t \geq 0) \geq \eta \tag{6}$$

式 (6) の KL はカルバック・ライブラー情報量（KL-divergence）を示しており，制約式の部分はガウス分布に従って重みベクトルをサンプリングしたときに受け取ったデータを識別できる確率が η 以上になることを条件付けている．カルバック・ライブラー情報量は確率分布間の距離を測る指標であり，CW の更新式は制約式を満たすガウス分布の中で前のガウス分布から一番形を変えずに済む新しいガウス分布への更新の最適化問題となっている．

CW はデータにノイズが含まれていたとしても，必ず η 以上の確率で分類できるようにパラメータを更新する．η の値を小さくすればノイズ耐性は強まるが，正確なデータから学習をする際にもガウス分布の更新が緩やかになるため学習の収束速度も遅くなるという問題があり，急なパラメータ変更を防ぐ AROW [14] や，ある程度 $(1 - \eta)$ 以上の誤分類を許容しつつパラメータの急激な変化を防ぐ SCW [15] といった改良が提案されている．SCW ではソフトマージン SVM [24] のようにソフトマージン最大化が行われる．**表 1** に Hivemall でサポートしているオンライン学習アルゴリズムの特徴をまとめる．

Hivemall ではバッチ学習を近似する確率的勾配降下法に基づいたアルゴリズムと，表 1 の直接データを線形分離するアルゴリズムの双方をサポートしている．Hive におけるそれぞれの手法を並列化するにあたっての要領は同一のた

表 1　Hivemall でサポートするオンライン学習アルゴリズムの特徴
Table 1　Characteristics of the online learning algorithms supported in Hivemall.

Algorithm	信頼度の利用	ノイズ耐性	ソフトマージン
PA [12]	×	×	×
CW [13]	○	×	×
AROW [14]	○	○	×
SCW [15]	○	○	○

め，本稿では確率的勾配降下法に基づくアルゴリズムに焦点を当ててその並列化の要点を述べる．

2.2　Parameter Mixing によるオンライン学習の並列処理

バッチ学習アルゴリズムは学習の収束に多数のイテレーションが必要であり，無共有型（shared-nothing）の並列計算機での実行には課題が残っていた．これに対して，近年では訓練例ごとにモデルを逐次更新していく確率的勾配降下法（SGD）や PA を利用したオンライン学習 [23], [25] を大規模な機械学習に利用するという動きがある．オンラインアルゴリズムはパラメータが連続的に更新されるため並列処理が難しいが，Iterative Parameter Mixing [21] により epoch ごとにパラメータを交換することで，訓練例をすべてメモリに保持することなく短い収束時間で大規模な学習を行うことができる．これに対し，epoch ごとのパラメータを交換を行わない学習手法は Parameter Mixing [26] と呼ばれる．

Hivemall では SGD ベースのロジスティック回帰について Iterative Paramter Mixing もサポートしているが，決定木の集団学習を行う Random Forest アルゴリズム [27] にならい，入力データを増幅したうえでアンサンブル学習の一種であるバギングを行うことで，1 パスの Parameter Mixing でも Iterative Parameter Mixing のようにデータのばらつきに対処する効果が得られるように工夫した．これは入力データ量に対しても map 数を増やすことで計算時間を抑えられる MapReduce ならではの手法である．4.3 節で技術詳細を述べる．

3.　関連研究

本章では，MapReduce による機械学習手法と関連する集約関数を用いた機械学習の処理手法を議論する．

3.1　MapReduce による機械学習の並列処理

機械学習を MapReduce を利用して並列処理する手法はこれまでにもいくつか提案されている．

Chu らは，Statistical Query Model で記述可能な機械学習のバッチ学習アルゴリズムは MapReduce による並列処理が可能であり，多くの機械学習手法が Statistical Query Model で表現可能であるとしている [16]．実際に文献 [16] のアルゴリズムが Apache Mahout [11] に実装されている．Statistical Query Model は和の形（Summation Form）に変形できるため，各々の項を複数の計算ノード（Mapper）に振り分けて処理をさせ，最後に Reducer に集約して和をとればよい．この手法は機械学習を集約演算の一種ととらえることで，部分集約 [28] による並列化を行う手法といえる．

機械学習は一般的に多数の繰返し（イテレーション）を必要とするため，入出力が分散ファイルシステムを介する MapReduce とは相性が悪い．同一データを入力として繰返し計算が行われるアルゴリズムのために，Spark [29] や Twister [30] は，中間結果を分散ファイルシステムではなくメモリ内に保持するインメモリ MapReduce 手法を提案している．これらの手法は中間データがメモリ内に収まる場合には効率的に動作するが，巨大データへの対処という点で特徴次元数に対するスケーラビリティに課題が残る．また，機械学習では訓練例の学習器への入力順によって偏りのある学習結果を生じることが知られており，各イテレーションごとに入力データを順不同に整列（shuffle）してから学習することが一般的となっているが，各イテレーションごとの shuffle 処理によるオーバヘッドはインメモリ MapReduce 手法にも依然として残る．

本稿では，4.3 節および 6.3 節で shuffle 処理によるオーバヘッドに関して詳細に分析し，Hivemall に導入されているイテレーションを除去してイテレーションと同等の効果を得ることができる代替案を述べる．

3.2　集約関数を用いた機械学習手法

機械学習をデータベースのユーザ定義関数（User Defined Function（UDF））またはユーザ定義集約関数（User Defined Aggregate Function（UDAF））として，データベース内に閉じた形で分析処理を行うものとして代表的な研究に Madlib [31] と Bismarck [18] がある．

ロジスティック回帰，SVM，パーセプトロンをはじめ線形識別モデルをとる機械学習手法の多くが，勾配降下法を利用した凸最適化で解けることが分かっており，Bismarck は確率的勾配降下法と Iterative Parameter Mixing を利用した機械学習のための統一フレームワークを提案している．Bismarck では勾配の計算をユーザ定義集約関数として実装している．集約処理は結合法則と交換法則を満たすとき部分集約による並列処理が可能であり，たとえば集約処理の一種である平均を求める演算 $avg(V)$ は結合法則と交換法則を満たす $sum(V)$ と $count(V)$ の計算に分解することで並列処理が可能である [28]．多くの関係データベースやその MapReduce によるデータウェアハウス環境 [3], [32] で，ユーザ定義集約関数は *Transition*，*Merge*，*Final merge* の 3 つのステップからなる．並列データベー

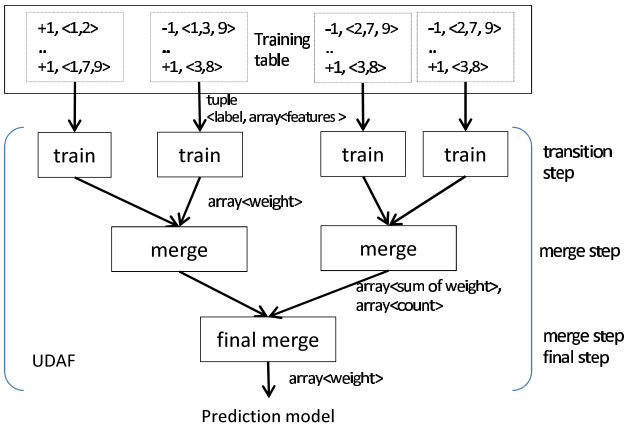

図 1　UDAF により実装された SGD に基づく分類器
Fig. 1　SGD classifier implemented as a UDAF.

スや Hive/Impala [32] は，これらのインタフェースに従った集約関数を並列処理する [33].

図 1 に，Bismarck における機械学習の学習処理の実行形態を示す．Transition フェーズでは，各学習器が訓練例を入力としてとって各々の学習モデルを生成する．このステップの実行はデータ並列の実行が可能である．次の Merge フェーズでは，Transition フェーズで生成された各学習モデルを入力として，それぞれの特徴の重みの合計と総計のペアを出力する．最後に，Final merge フェーズで Merge フェーズの出力結果から各特徴ごとの重みの平均を計算する．

このユーザ定義集約関数を用いたアプローチの問題点は，逐次実行される Final merge フェーズの fan-out によってスケーラビリティが制限されることである．また，図 1 の各オペレータはデータを 1 つずつ受け取りながら逐次的に識別関数を学習していくため出力結果のみを保持すればよいが，ユーザ定義集約関数を用いた場合，Final merge オペレータで最終的にスカラー値として実行結果を出力する必要があるため，特徴数に応じたメモリ空間の確保が必要であり特徴数に対するスケーラビリティの確保が困難である．特徴数が問題となる場合，Feature hashing [*2] やトピックモデリングによって特徴空間の次元削減を行うこともできるが，実特徴のトラッキングが難しくなるという問題が生じるのと，データによっては次元削減が困難な場合があるため万能ではない．また，Dremel [35] などと異なり，MapReduce は多段の集約をサポートしていないため，実際には Merge と Final Merge ステップに相当する N-way のマージ処理が Reducer サイドでの逐次的な処理となる．

我々は Hivemall における機械学習の実現手段として，初期段階では従来のユーザ定義集約関数に基づくアプローチも検討したが，Hadoop 環境では各 map/reduce タスクで確保できるメモリ量の問題から断念した．Hadoop 環境では各 map/reduce タスクで確保できるメモリ量は一般的に推奨されている値として $\frac{Total\ memory\ reserved\ for\ mapreduce}{\#mapslots+C*\#reduceslots}$（ただし，$C$ は 1.0〜1.3）程度であり，24 GB のメモリを搭載した 2 ソケットの Quad コアプロセッサ（hyper-threading 有効）環境では，各 map/reduce タスクに割当て可能なメモリ量は map スロット数（$\#mapslots$）を 7，reduce スロット数（$\#reduceslots$）を 3 としたとき，たかだか 2 GB である．また，現在の Hadoop/Hive の実装では各タプルを出力する際にタプルをメモリ内に実体化したのち HDFS に書き出す処理をしていることからも，スカラー集約でメモリ不足が発生することが頻出した．

本稿では，4.1 節で，この問題を解決するために関係代数演算の並列処理に適したテーブル生成関数に基づく機械学習の実現手法を導入する．

4. Hivemall の構成と特徴

本章では，まずスケーラブルな機械学習を実現するうえでの諸問題を述べ，4.1 節以降でこれらの問題に対処する Hivemall の採用したアプローチを述べる．

スケーラブルな機械学習を実現するうえでの課題

一般的に，スタンドアロンのプログラムとして動作する教師あり機械学習 [36], [37] や集約関数に基づく機械学習機構 [18], [31] は次のようなデータフローをとる．機械学習の学習フェーズでは，`(vector<feature>,label)` を各インスタンスとする訓練データ（ここで `vector<feature>` は特徴の可変長配列）を入力として，特徴ごとの重みを保持する `map<feature, weight>`（ここで `map<K,V>` は `K` を `V` にマッピングする辞書）をモデルとして出力する．予測フェーズでは，学習済みのモデル `map<feature, weight>` をメモリ内に展開して，そのモデルを利用してテストデータの各インスタンス `vector<feature>` ごとに `label` または `probability` を出力する．

このようなデータフローでは，機械学習の入力データをスケール（特徴数の増加と訓練例の増加）させた場合，次のような問題が生じる．

- 学習フェーズで，最終的に単一の Reducer がスカラー値として重み `map<feature, weight>` を返すために特徴数が多い場合にメモリ不足が発生する．また，集約関数による機械学習の実装は 3.2 節に説明したように MapReduce での実行に向かない．これに対処する方法としては，文献 [8] のようにアンサンブル学習を導入して各弱学習器でサンプリングを行って出力するモデルのサイズを抑えることが考えられるが，学習したモデルの精度が犠牲となる可能性がある．

[*2] Feature hashing または Hashing trick と呼ばれるテクニックでは，高次元の特徴ベクトル x をハッシュ関数 $\phi(x)$ により低次元のベクトル z に写像（$\phi : x \to z$）する．各訓練例が多数の疎な特徴からなる場合，ハッシュ関数によって次元を削減してもうまく機能することが知られている [34]．Hivemall や Vowpal Wabbit [17] では Feature hashing を用いた Feature Engineering をサポートしている．

- 予測フェーズでは，学習済みのモデル `map<feature, weight>` から，テスト事例の特徴ごとに辞書を参照して重みを算出する．テスト事例の特徴数が膨大な場合に索引がメモリに保持しきれない可能性があるのと，1つ1つの索引参照処理が逐次的に行われるため特徴数 × テスト事例数に応じた時間がかかる．
- イテレーションは機械学習に必要な不可欠なものとされているが，MapReduce では各イテレーションで HDFS を介した入出力が行われるため，イテレーションを要するプログラムの実行に不向きである．また，3.1 節で説明したように，各イテレーションごとに shuffle 処理によるオーバヘッドが生じる．

4.1　テーブル生成関数を用いた関係演算に適した機械学習

データ量に対するスケーラビリティの問題に対処するために，Hivemall ではより関係演算に適した形でモデルを特徴と重みの 2 カラムからなるリレーション `relation<feature, weight>` として出力するように学習を行う．具体的には，ユーザ定義集約関数（UDAF）の代わりに，ユーザ定義テーブル生成関数（User-Defined Table-generating Function（UDTF））を利用することで関係に基づく機械学習のデータ処理を行う．このことにより，並列データベースの技法を踏襲する Hive における関係演算の並列化の恩恵を享受することが可能となる．

UDTF は行ごとに任意の行を出力するような働きをするリレーションを返す関数である．ここで，行ごとに任意の行を出力するとは，1 つの行を受け取って N（$N \geq 0$）以上の行を出力すること，出力行に含まれるカラム数 K が $K \geq 1$（K は関数を通じて固定）であることを指す．Hive の問合せ処理器が UDTF の *process* メソッドをタプルを 1 行ごとに供給する形で呼び出す．UDTF 側から結果を出力するには，システム側であらかじめ用意されている *forward* メソッドを出力タプルごとに呼び出す [38]．

図 2 に UDAF ベースのアプローチと UDTF ベースのアプローチの実行形態を比較して示す．UDTF ベースのアプローチでは，学習は map-only の MapReduce ジョブとして実行される．学習モデルの算出に至る手順は次のとおりである．*process* メソッドが問合せ処理器から呼び出されるごとに，オンライン学習を行う各弱学習器が勾配を計算し，ローカルに学習モデルを更新する．問合せ処理器からのタプルの供給が終わった段階で *close* メソッドが呼び出されて，各弱学習器が各特徴ごとに *forward* メソッドを呼ぶ形でタプルを出力する．

学習タスクは**図 3** に示す HiveQL クエリによって実行される．この内部問合せが map-only のジョブを実行する部分である．図 3 の外側の SELECT 文がバギングを用いたモデルの平均化を行う部分であり，モデル平均化は

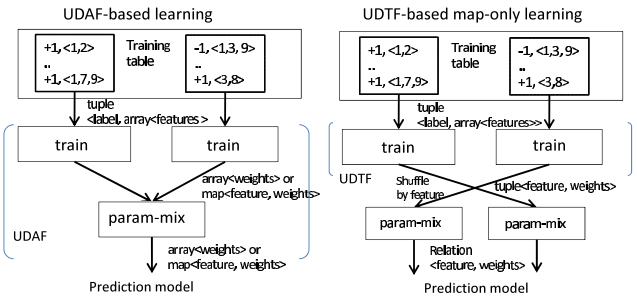

図 2　UDAF と UDTF によるアプローチの比較
Fig. 2　Comparison of the UDAF- and UDTF-based approaches.

```
1  CREATE TABLE model AS
2  SELECT
3    feature, -- reducers perform model averaging in parallel
4    avg(weight) as weight -- bagging
5  FROM
6   (SELECT
7      logress(features,label,..) as (feature,weight)
8    FROM train
9   ) t -- map-only task
10 GROUP BY feature -- shuffled to reducers
```

図 3　予測モデルを生成する HiveQL クエリ
Fig. 3　HiveQL query to generate a prediction model.

GROUP-BY の集約クエリによって表現され，MapReduce によって並列に処理される．最終的に出力される学習モデルは特徴（feature）と重み（weight）の 2 つのカラムからなる通常のリレーションであり，スパースな様式となっている．

このとき，map タスクの数は Hadoop/Hive の各タスクへの入力サイズの指定によって調整が可能であり，embarrassingly parallel な学習が可能である．Reducer の数も設定を通じて調整可能であり，このため UDAF ベースのアプローチと異なり，UDTF ベースのアプローチにはスケーラビリティを阻害するようなボトルネックが存在しない．以上より，テーブル生成関数による機械学習の実装は従来の定石である集約関数による機械学習にスケーラビリティ面で優れる．

4.2　LATERAL VIEW と外部結合を利用した Predict 処理

Hivemall では，機械学習の予測フェーズにおける Predict 処理を HiveQL の LATERAL VIEW と外部結合を用いて行う．

テストテーブルはあらかじめ**図 4** の問合せにより，評価例を (ROWID, feature) のペアに展開した形のもの（testing_exploded テーブル）を用いる．LATERAL VIEW はテーブル生成関数とともに利用される SQL-99 標準の LATERAL 副問合せ相当の機能を提供する HiveQL 機能である．図 4 で利用されている explode 関数は特徴ベクトルの

```
CREATE TABLE testing_exploded AS
SELECT
  rowid,
  feature
FROM
  testing
  LATERAL VIEW explode(features) t AS feature
```

図 4　テスト事例を展開したテーブルを生成する問合せ
Fig. 4　A query to generate exploded testing examples.

```
SELECT
  t.rowid,
  sigmoid(sum(m.weight)) as prob
FROM
  testing_exploded t LEFT OUTER JOIN
  model m ON (t.feature = m.feature)
GROUP BY
  t.rowid
```

図 5　学習モデルを用いた予測を行う問合せ
Fig. 5　A query to make a prediction using a pre-learned model.

配列を受け取って特徴ごとに展開して行を出力する UDTF であり，LATERAL VIEW はこの展開を行ったうえでベースとなった testing テーブルのタプルとの 1 対多の対応付けを仮想的な JOIN 処理により行うものである．

図 5 ではテスト事例テーブル testing_exploded の各行ごとに学習モデルから重みを算出したうえで，実際のテスト事例を意味する rowid ごとに重みを集計して算出している．ここでは，重みの合計を求めたうえで sigmoid 関数によって区間 $(0, 1)$ の確率値に変換している．なお，図 5 のクエリは関係演算の並列処理が行われるため，データ量が増加してもスケール可能である．

4.3　タプル増幅による繰返しの除去

確率的勾配降下法など機械学習においてイテレーションは精度の高い予測モデルを得るために不可欠なものである．一方で，MapReduce は各 MapReduce ジョブの I/O が分散ファイルシステムを介することから繰返しが必要なアルゴリズムに不向きである．

Hivemall は機械学習における繰返しの効果を，複数の MapReduce ステップを得ることなく，amplify 関数によって模擬する．amplify 関数は，第 1 引数である${xtimes} に増幅因数をとり，入力行を任意数だけ増幅して出力するテーブル生成関数である．**図 6** は，増幅因数を 3 として訓練例の各行を 3 倍に増幅したビューを生成する問合せである．

図 6 で，CLUSTER BY k は DISTRIBUTED BY k と SORTED BY k の組合せの構文糖であり，CLUSTER BY rand() は分割に用いるキー k をランダムに生成してタプルの分散に利用する．ここで，Map タスクの出力タプルを k を分散のキーとして各 Reducer に配布する．Reducer への入力は k を基に整列され，その結果 Reducer への入力は順不同なものとなる．

```
set hivevar:xtimes=3; -- set the multiplication factor to 3.
CREATE VIEW training_x3
as
SELECT * FROM (
  SELECT
    amplify(${xtimes}, *) as (rowid, label, features)
  FROM
    training
) t
CLUSTER BY rand() -- Random shuffle to reducers
```

図 6　amplify 関数を利用した訓練例のビュー定義
Fig. 6　A view definition that uses the amplify function.

図 6 で生成したビューを図 3 の学習への入力とした場合，**図 7** のように問合せは 2 つの MapReduce ステージにより実行される．ステージ 1 の Map タスクでは各タプルが増幅され，生成されたタプルが Reducer にランダムに分散される．Reducer 側で map 関数の出力ファイルが（外部）マージソートされ，reduce 関数への入力がシャッフルされる．各弱学習器の学習は reduce 側で実行され，特徴 (feature) ごとの集約処理 $sum(m.weight)$ が部分実行され，結果が HDFS に出力される．ステージ 2 の Map タスクで group by 句の評価が行われ，特徴の値に基づいて Reducer にタプルが送信される．そして，Reduce タスクで特徴ごとの部分集約結果をまとめた最終的な集約処理が行われる．この amplify 関数による訓練例の増幅と CLUSTER BY 句によるシャッフル処理を学習に利用することは，複数回機械学習のイテレーションを回すことと似た働きをする．

Mapper 側でのタプル増幅とシャッフル

図 7 のクエリの実行では，2 ステージの MapReduce が学習の実行に必要であるが，ステージ 1 の map 処理を行う map タスク数が多い[*3]と reduce 側でのその merge 処理がボトルネックとなることがある．そこで Hivemall では，map タスク内で入力行をシャッフルする rand_amplify 関数を用意した．rand_amplify 関数では，第 1 引数の${xtimes} に増幅因数をとり，第 2 引数の${shufflebuffersize} にシャッフル用のバッファサイズを指定する．図 6 の訓練例のビュー定義は，rand_amplify 関数を利用する場合，**図 8** のようになる．

rand_amplify 関数を利用した学習処理の実行プランは，**図 9** に示すものとなる．図 9 で，rand_amplify オペレータの部分が入力タプルの増幅とシャッフルを行う部分である．この map タスク内で局所的な入力タプルの増幅とシャッフルは全体のシャッフルとはならないが，経験的に十分に機能する．

6.3 節で，amplify 関数および rand_amplify 関数を利用したタプル増幅による繰返し除去手法それぞれについて実行時間と得られた精度面での向上を評価する．

[*3] Hive ではデータ量に応じた数の map タスクが実行される．

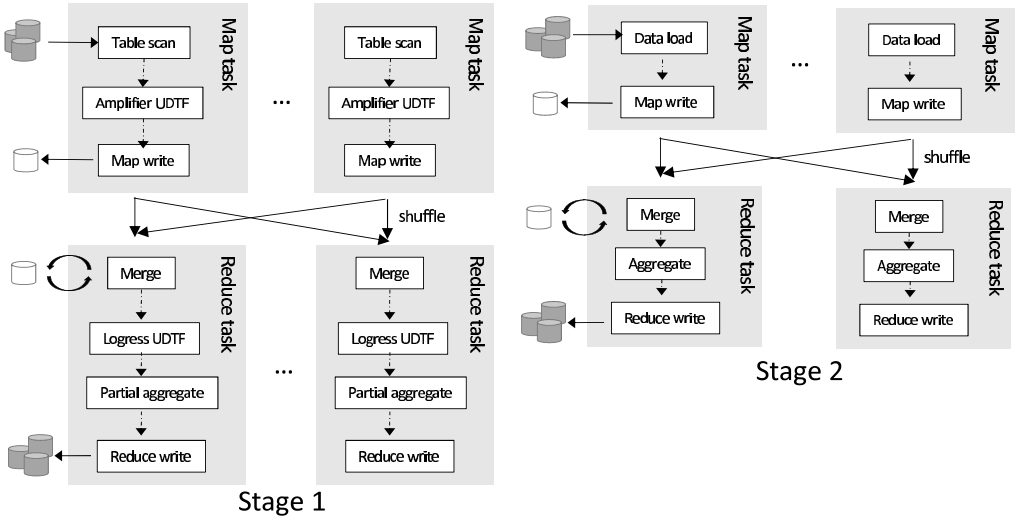

図 7　amplify 関数を利用した学習を行うクエリの実行
Fig. 7　Query execution of a training using the amplify function.

```
1  set hivevar:shufflebuffersize=1000;
2  create or replace view training_x3
3  as
4  select
5    rand_amplify(${xtimes}, ${shufflebuffersize}, *)
6     as (rowid, label, features)
7  from
8    training;
```

図 8　rand_amplify 関数を利用した訓練例のビュー定義
Fig. 8　A view definition that uses the rand_amplify function.

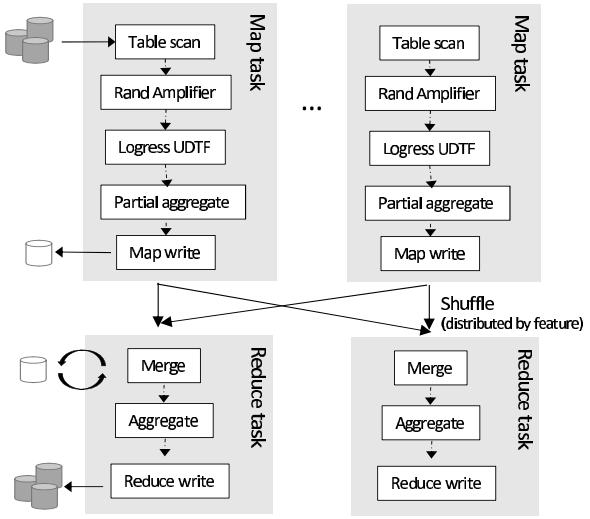

図 9　rand_amplify 関数を利用した学習を行うクエリの実行
Fig. 9　Query execution of a training using the rand_amplify function.

5.　大規模機械学習を実現するために Hivemall で採用している技巧

Hivemall では Hive に特徴的な機能を用いることで，特徴エンジニアリング（Feature engineering）や Predict 処理を MapReduce によって並列処理している．本章では，4 章で述べた項目以外の大規模機械学習を実現するうえで有益である諸技巧を述べる．

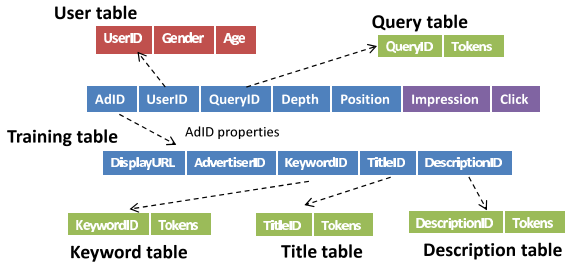

図 10　KDD Cup 2012, Track 2 のデータセットの構成
Fig. 10　The dataset composition of KDD Cup 2012, Track 2.

5.1　Hashing trick と結合演算を用いた特徴エンジニアリング

Hive は並列データベースと同様の並列ハッシュ結合 [39] をサポートしており，複数の巨大なデータソースの結合処理を行うことができる．KDD Cup 2012, Track 2 のデータセット**図 10** のように，現実の機械学習データセットはしばしば複数のテーブルから構成されるため，並列結合演算をサポートしている Hive は巨大なデータセットの特徴エンジニアリングとも呼ばれる前処理を行う用途に適している．

図 11 は図 10 の training テーブルについて user テーブルと userid で左外部結合を行っている例である．concat 関数は id などの質的変数をユニークな特徴として表現する（**図 12** がその概念を示す）ために利用しており，mhash 関数では Hashing trick [34] によって特徴空間の写像を行っている．なお，6 章の評価実験で KDD Cup 2012, Track 2 のデータセットを扱ううえで実際にこの技巧を利用した[*4]．

[*4] Hivemall の Wiki ページ [40] では KDD Cup 2012, Track 2 のタスクを扱ううえでの前処理の詳細を記載している．

```
CREATE VIEW training2 AS
SELECT
  rowid,
  ..
  mhash(concat("1:", displayurl)) as displayurl,
  mhash(concat("2:", adid)) as adid,
  ..
FROM (
  SELECT
    t.*, u.gender, u.age
  FROM
    training t
    LEFT OUTER JOIN user u
      on t.userid = u.userid
) t;
```

図 11　訓練例のテーブルを生成する前処理の例
Fig. 11　An example that generates a training table.

Label	A	B
1	1	9
-1	2	7
1	3	8

⇒

Label	A:1	A:2	A:3	B:7	B:8	B:9
1	1	0	0	0	1	0
-1	0	1	0	0	0	1
1	0	0	1	1	0	0

図 12　質的変数の二値素性への変換
Fig. 12　Conversion of explanatory variables to binary variables.

```
SELECT
transform(rowid, clicks, (impression - clicks), features)
  ROW FORMAT DELIMITED ..
using 'awk␣-f␣conv.awk'
  as (rowid BIGINT, label FLOAT, features ARRAY<INT>)
  ROW FORMAT DELIMITED ..
FROM training
```

図 13　transform 句を利用した二値分類問題への変換
Fig. 13　Conversion to a binary classification problem using the transform clause.

5.2　Transform 句を利用した特徴エンジニアリング

図 13 で利用している Hive の transform 句 [38] は，タプルを任意のスクリプトの標準入力に出力して，そのスクリプトの標準出力を入力として任意の関係を出力するテーブル生成関数の一種である．Hivemall の KDD Cup 2012, Track 2 の例 [40] では，この機能を広告のクリック回数とインプレッション数をロジスティック回帰で扱うための二値分類データセットに変換する用途に利用している．

5.3　カラム圧縮フォーマットによるディスク読み出しの効率化

KDD Cup 2012, Track 2 のデータセットから TSV 形式で訓練例のテーブルを作成すると約 12 GB である．Hive では，Optimized Row Columnar（ORC）と呼ばれる Partition Attributes Across（PAX）[6] の一形式であるカラム指向の圧縮ストレージ形式をサポートしている．カラム指向の圧縮は同様の属性を集めることで高い圧縮率を誇るため，データセットによってはディスク読み出し量を大幅に削減できる．機械学習のデータセットはこうした圧縮が有効なフォーマットである．実際に，前述の 12 GB の訓練テーブルは ORC 形式で Snappy [41] 圧縮を施したところ 7.9 GB と TSV 形式の 65.8%に消費ディスク量が縮小された．

ログデータなどのアーカイブデータなど膨大なデータを扱ううえでは効率的なデータ圧縮が不可欠になると考えられ，こうした高圧縮率のカラム指向テーブルを容易に扱えることはビッグデータの機械学習を Hive および Hivemall で行ううえでの大きな利点といえる．

6.　評価実験

本章では，6.1 節で Hivemall のクラス分類性能と学習に要する時間について競合手法と比較した性能を示し，6.2 節で Hivemall の計算ノード数に対するスケーラビリティを評価する．評価には，KDD Cup 2012, Track 2 [42] のインターネット広告のクリックスルー率（Click-Through Rate（CTR））推定タスクを用いる．

KDD Cup 2012, Track 2 のクリック率推定タスク

KDD Cup 2012, Track 2 のデータセットは KDD Cup 2012 のコンペティションのために中国の検索エンジン SOSO.com を運営する Tencent Corporation によって提供された実データセットで，149,639,105 レコードの訓練データセットと 20,297,594 レコードのテストデータセットからなる．各レコードは検索エンジンとユーザのインタラクション（セッションと呼ばれる）を表現する．

図 10 に関係を示すように，訓練データの各レコードは，AdID, DisplayURL, AdvertiserID, KeywordID, TitleID, DescriptionID, UserID, QueryID, Depth, Position, #Impression, #Click の 12 属性からなり，あるユーザ（UserID）が特定のセッションにおいて，AdID の広告に#Impression 回接触し，そのうち#Click 回広告をクリックしたことを表現する．検索語（QueryID）および広告の KeywordID, TitleID, DescriptionID は外部キーとして表現されており，hash 値によって数値化されたトークンの集合が外部キーによって参照される．テストデータの構成は，Training データから#Impression，#Click を除いたものであり，残る 10 個の属性が特徴ベクトルを構成する．図 10 の training テーブルと user テーブルを結合して質的変数を二値素性に分解すると，特徴数は 54,686,452 である．

このタスクの課題は，正確な広告クリック率の推定器を開発することだけでなく巨大なデータを取り扱う点にある．補助データ構造を含めた Training データのサイズは TSV 形式で約 12 GB であり，データサイズ，Training/Test インスタンス数のいずれも他の機械学習評価データセット [36] と比べ 1 桁大きい．このため，学習の収束時間や必要メモリ量から LIBLINEAR [37] などの単一計算機で動作する機

表 2　実験環境のハードウェア構成
Table 2　Hardware specifications of the experimental environment.

	Server specifications
CPU	E5520 @ 2.27 GHz
Sockets	2
Cores	4
Hyper threading	2
Total threads	16
Memory	24 GB
Disk	SATA 7,200 rpm × 3 (Software RAID 0/JBOD)
Ethernet	1 Gbps

械学習ツールをそのまま適用することが困難であり，データ量に対してスケールする学習手法が必要である．

KDD Cup 2012, Track 2 の広告クリック率推定タスクで，予測結果の精度評価は偽陽性率と真陽性率を軸とする ROC 曲線下の面積（Area Under Curve（AUC））[43] によって行われる．AUC の値は，広告をクリックしたことを意味する正例と広告をクリックしなかったことを意味する負例をランダムに 1 個ずつ選んだとき，正例の予測値が負例の予測値以上になる確率を意味する．このため，予測モデルの精度の評価指標となる．AUC は 0 から 1 までの値をとり，完全な分類が可能なときの面積は 1 で，ランダムな分類の場合は 0.5 となる．

実験環境

評価には 33 台構成のプライベートクラスタを利用した．各ノードのハードウェアの構成は**表 2** のとおりである．Hadoop の tasktracker と datanode を実行するワーカノードとして 32 台を利用し，残る 1 台は jobtracker と namenode を管理するマスタノードとして利用した．Datanode のディスクの構成は文献 [33] で推奨されている JBOD 構成をとり，スタンドアロンで動作する Bismarck では Linux の Software RAID 0 を構成したパーティションを利用した．

6.1　競合システムとの性能比較

KDD Cup 2012, Track 2 のクリックスルー率の予測タスクを用い，Hivemall のクラス分類性能と学習に要する時間を state-of-the-art の機械学習手法と比較した．Hivemall との性能比較に用いたのは，シングルプロセスで動作する Vowpal Wabbit（VW）[17]，Hadoop Streaming によって並列に動作する Vowpal Wabbit（VW-MR），そして Bismarck [18] の 3 つである．これら 3 つの機械学習フレームワークすべてが確率的勾配降下法を用いたロジスティック回帰の実装を提供しているため，素直な比較が可能である．

Microsoft Research の John Langford を中心に開発されている VW はオープンソースの機械学習ツールとして最もスケーラビリティの高いものとして知られている．Hadoop Streaming を利用した VW の並列処理（VW-MR）は，確率的勾配降下法（SGD）によるオンライン学習と準ニュートン法の Limited-memory Broyden-Fletcher-Goldfarb-Shanno（L-BFGS）[44] を用いたバッチ学習を組み合わせた学習手法を採用している．VW-MR は，Vowpal Wabbit の配布物に含まれる *mapscript.sh* スクリプトを用いることで，map-only の MapReduce ジョブを実行する．まず，VW-MR はすべての訓練例を用いて確率的勾配降下法の単一パスを実行して，ラフな解を求める．そのうえで，L-BFGS を用いて正確な解を求める．経験損失とその勾配の計算を各 Mapper でローカルに行ったうえで，MPI 由来の AllReduce アルゴリズムによって全体的な経験損失と勾配の計算を行ってグローバルなパラメータの更新を行う．VW-MR はデフォルト設定で，単一パスの SGD（以下，この試行を VW-MR（SGD）と表記する）とそれに続いて最大 20 の L-BFGS プロセス（以下，この試行を VW-MR（SGD+L-BFGS）と表記する）を実行する [17]．利用した Vowpal Wabbit のバージョンは 7.7 で，設定にはデフォルトの設定値を用いた．本稿の実験では VW-MR（SGD+L-BFGS）の学習は L-BFGS の 10 イテレーション目で学習が収束した．

Bismarck は確率的勾配降下法を採用した In-database Analytics の代表的な実装である．損失関数は文献 [18] のとおりであり，L1 ノルムを利用した正則化項を含む．本稿の実験では Bismarck を PostgreSQL 9.2 で運用した．なお，PostgreSQL 上で Bismarck の学習プロセスはシングルプロセスでのみ動作する．Bismarck の学習のイテレーション回数は 10 回と 1 回を評価したがイテレーションを増すごとに AUC の値が悪化することがあったことから，1 回のイテレーションの結果のみを本稿では用いる．この理由として，Bismarck ではイテレーションごとに学習率 η が decay factor に基づいて減衰するような実装となっている[*5]が，デフォルト設定では decay factor が 1.0 であるため，overshoot を繰り返して学習が収束に向かっていないものと考えられる．

他の機械学習の実装として，我々は Apache Mahout [11] に実装されている SGD ベースのロジスティック回帰を評価したが，その学習は 3 時間以上かかっても終了しなかったため比較対象から外した．この理由として，Apache Mahout のロジスティック回帰の実装は MapReduce による並列処理はまったく行っておらず，メインスレッド 1 本だけで逐次的に HDFS からデータを読み出して学習する素朴な実装となっていることに起因する．また，線形分類に特化した学習速度の速い学習器として広く知られている Liblinear [37] で同様の実験を行ったところ，学習に 50 GB 以上のメモリ

[*5] http://hazy.cs.wisc.edu/hazy/victor/doc/using_bismarck.php

表 3　各手法のハイパーパラメータの特性
Table 3　Characteristics of hyperparameters.

	学習率 η	学習率 η の特性	収束性 $\eta_{t+1} < \eta_t$	正則化	バイアス項
Hivemall	$\eta_0/(1+t/N)$	$\eta_0 = 0.1$, $\eta_t \to 0.05$ $(t \to \infty)$	○	なし	あり
Bismarck	固定（0.1）	固定（iteration ごとに減衰）	×	L1	なし
VW	$0.5(1/(1+t))^{0.5}$	$\eta_0 = 0.5$, $\eta_t \to 0$ $(t \to \infty)$	○	無効*6	なし
VW-MR	L-BFGS では設定不要*7	SGD は VW と同様	○	無効*6	なし

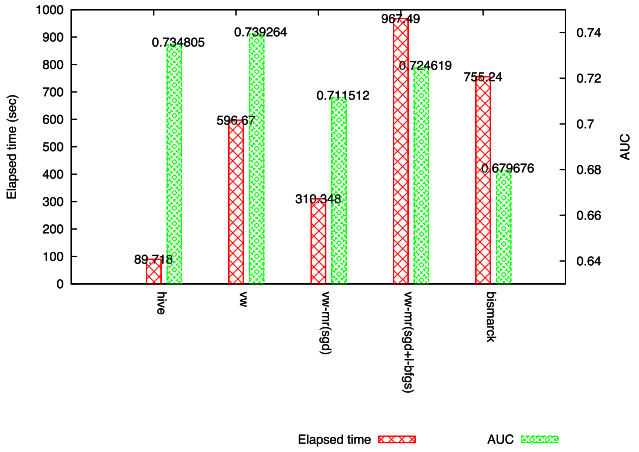

図 14　KDD CUP 2012, Track 2 のタスクを利用した性能評価
Fig. 14　Performance evaluation of the KDD CUP 2012, Track 2 task.

を必要とした．このため，表 2 より高性能なマシンで学習を行ったが，学習の終了には 4 時間以上を要した．

図 14 に，KDD CUP 2012, Track 2 のタスクを VW，VW-MR（SGD および SGD+L-BFGS），Bismarck，Hivemall それぞれの学習を行ったときの性能を示す．**表 3** に，今回の実験で利用した各手法のハイパーパラメータの特性についてまとめた．表 3 で，η_0 は学習率の初期値，N は総エポック数，t はエポック（$1, 2, \ldots, N$）である．

この実験結果より，Hivemall が競合実装であるスタンドアロンの VW に対して同等の予測精度で約 6.65 倍の学習速度で学習を終えていることが分かる．このときの Hivemall の学習速度は，秒間 2,298,010.8 タプルの処理と高いスループットを示している．Hadoop Streaming を利用した並列処理版の VW-MR（SGD）は，Hivemall に対して 3.46 倍の学習時間を要した．Vowpal Wabbit では，計算機台数を 1 台（VW）から 32 台（VW-MR（SGD））にノード数を増やしたことで得られた学習に要した時間の性能向上は 1.92 倍であった．このことから VW-MR のノード数に対するスケーラビリティは不十分であるといえる．VW-MR（SGD）の AUC が 0.711512 と Hivemall に比べて悪い理由としては，学習率の違いのほか，弱学習器の数が Hivemall に比べて多いことが理由と考えられる．VW では LibSVM 形式のテキストフォーマットを入力として用いるため，Hivemall が採用するデータ形式に比べて入力データの HDFS ブロック数が多くなり，結果として MapReduce で立ち上がる弱学習器の数が増えがちである．VW-MR では 215 の map タスク（弱学習器）が利用された．このため，個々の弱学習器が出力する予測モデルの精度が悪いものと考えられる．

VW-MR（SGD+L-BFGS）では，L-BFGS の 10 イテレーション目で学習が収束し，VW-MR（SGD）に対して AUC の向上が見られたが，単一ノードの VW で得られた AUC には劣るものであった．この原因として，個々の勾配計算で入力とする訓練例が固定であることに起因して，イテレーションごとに AllReduce による予測モデルの平均化を行っても真の最適値（global optimum）に至っていないものと考える．なお，VW-MR の予測精度に関しては弱学習器の数や学習率などハイパーパラメータの調整で改善される可能性がある．

Bismarck は学習の終了に 12.6 分を必要とし，Hivemall に対して 8.42 倍遅い学習時間となった．Bismarck の予測精度が悪い理由として，Bismarck の実装では確率的勾配効果法の学習率 η に関して，イテレーション内でつねに固定（デフォルトのステップ幅は 0.1）であり $\sum_t \eta_t^2 < \infty$ を満たさないため，学習が収束しなかったものと考える．

以上の性能差は，MapReduce により適合するように設計された我々の並列処理手法と Hivemall の競合実装に対する明らかな優越を示すものである．なお，損失関数や重みの更新式が同一で収束保障が得られる場合には，原理的に各手法のハイパーパラメータを調整したうえで収束するまで長時間かけて学習を行うことにより予測精度自体は同程度のものが得られる可能性があるが，図 14 の結果から少なくとも学習速度面で Hivemall は競合手法に優位性がある．

6.2　ノード数に対するスケーラビリティ

本節では，Hadoop クラスタのワーカノードの構成台数を変化させて Hivemall のノード数に対するスケーラビリティを調査した．**表 4** にワーカノードの構成台数を 8，16，32 としたときの，総 map スロット数，総 reduce スロット数，および Hivemall が学習に要した時間を示す．表 4 より，8 ノードから 16 ノードにノード数を増やしたときの性能向上は約 1.72 倍，8 ノードから 32 ノードに増やしたと

*6　L1/L2 の正則化オプションがあるが，デフォルトでは無効．
*7　L-BFGS では line search により η の近似解が利用される．

表 4　ノード数に対するスケーラビリティ
Table 4　Scalability to the number of nodes.

#nodes	Elapsed time（sec）	#map slots	#reduce slot
8	235.309	56	24
16	136.565	112	48
32	89.718	224	96

きの性能向上は約 2.62 倍となっており，台数に応じて性能が伸びていることが分かる．

8 ノードから 32 ノードにノード数を増やしたときの性能向上が，8 ノードから 16 ノードにノード数を増やしたときに得られた性能向上に比べると劣る理由としては次のことが考えられる．Hivemall では訓練例のテーブルのサイズから自動的に判定されて 103 個の map タスクが実行され，64 個の reduce タスクが実行される．8 ノードによる実行では，map スロット数が 56 と不十分であるため，map タスクで逐次的に実行が行われる箇所があった．これに対し，16 ノードでは map スロット数が 112 と十分であり，map サイドでの実行が完全に並列に行われる．このため，8 ノードから 16 ノードに構成を変えた場合に得られる性能向上は大きい．一方で，16 ノードから 32 ノードにノード数を増やした場合，台数を増やした効果は reduce タスクの実行時間だけに影響する．このため，16 ノードから 32 ノードへ台数増加することで得られた効果が 8 ノードから 16 ノードへ台数増加の場合と比較して小さくなったものと考える．

Hivemall のノード数に対するスケーラビリティは，ノード数（とノード数に応じる map/reduce のタスクスロットの数）が与えられたタスク数に対して十分となった段階で収束する．一方で，入力データが巨大である場合には収束に至るまでは計算ノードを追加することで学習に要する時間を短縮することができる．

6.3　タプル増幅による効果の評価

4.3 節で，タプル増幅によって機械学習の繰返しを模擬する amplify 関数と rand_amplify 関数を紹介した．本節では，KDD Cup 2012, Track 2 のデータセットを用いて，これらのタプル増幅による効果を学習結果の精度と学習時間の両面で評価する．

表 5 にタプル増幅を行わない場合（plain），amplifier 関数を利用した場合（amplifier+clustered by），rand_amplifier 関数を利用した場合（rand_amplifier）それぞれで学習に要した時間と得られた AUC を示す．amplifier 関数および rand_amplifier 関数のタプル増幅の倍数は 3 とした．表 5 に示すように，amplifier 関数を利用した場合は得られた AUC は最も高いが，実行時間が 479.855 秒とタプル増幅を行わない場合と比較して約 5.35 倍もかかってしまっている．このとき時間経過とともにどの処理に時間がかかっていたかを**図 15** に示す．図 15 で，横軸は経過時間で縦

表 5　訓練でタプル増幅を行う場合と行わない場合の AUC および実行時間の比較
Table 5　Comparison of AUC and the execution time of training methods with/without tuple amplification.

Method	Elapsed time（sec）	AUC
plain	89.718	0.734805
amplifier+clustered by	479.855	0.746214
rand_amplifier	116.424	0.743392

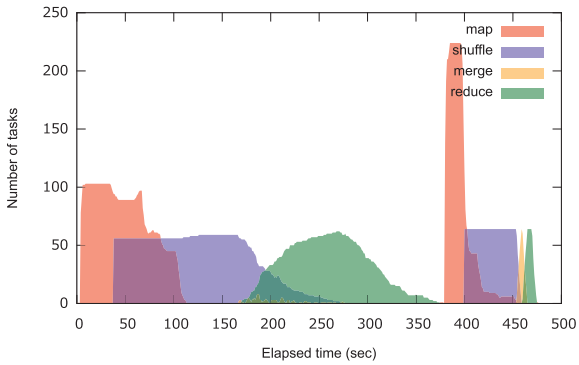

図 15　amplify 関数を利用した訓練タスクの実行
Fig. 15　Execution of a training task using the amplify function.

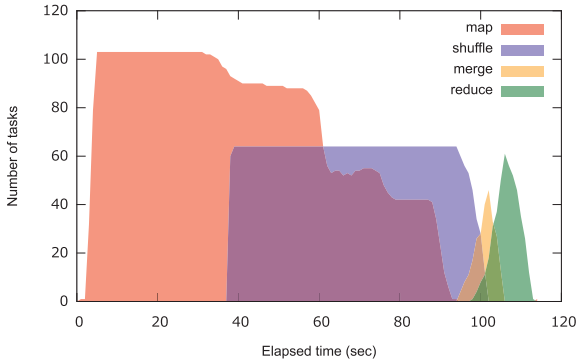

図 16　rand_amplify 関数を利用した訓練タスクの実行
Fig. 16　Execution of a training task using the rand_amplify function.

軸には map/shuffle/merge/reduce のどのタスクがどれだけ実行されていたかを示している．

amplify 関数を利用した場合の問題点は，map 出力結果を Reducer にコピーする shuffle 処理と，Reducer 側で行う map 出力結果の merge 処理がボトルネックとなっている点にある．訓練例のテーブルが非常に大きくて 100 の map タスクが利用される場合，merge オペレータは 100 個のファイルを外部マージソートによってマージして reduce 関数への入力とする必要がある．このため，amplify 関数は実行時間が重要である場合に大きな訓練例に対して利用することは難しい．

一方で，**図 16** に示すように rand_amplify 関数を利用する場合は精度こそ amplify 関数を利用した場合には劣

るが，タプル増幅を行わない場合に対して実行時間の大幅な増加なしに高い AUC が得られている．図 16 に示すように，rand_amplify 関数を利用した場合，shuffle 処理は map 処理に重なる形で実行され，多数の merge 処理と reduce 処理が並行して実行される．実行時間は約 1.3 倍に増加したが AUC で大きな向上が得られており，この結果は rand_amplify 関数をイテレーションの代わりに用いることの有効性を示すものである．

6.3.1　タプル増幅手法の有効範囲

イテレーションの代わりにタプル増幅を用いる場合，収束するまで学習を繰り返すという手法をとることができないため，収束しにくいデータや学習手法はその有効範囲から外れる．また，タプル増幅を用いる場合，あらかじめ増幅数を指定する必要があるほか，収束するまでの繰返し数が多い場合にそのコストが無視できなくなる．

Hivemall が採用するタプル増幅手法が有効となるのは，学習データが大量に存在するなどして少数のイテレーションで学習が収束する場合である．学習データが少ないケースは，単一ノードで動作する機械学習ライブラリで学習を行えばよいため，Hivemall の適用範囲にしていない．Confidence Weighted などの共分散を考慮した重みの更新を行うクラス分類手法 [13], [14], [15] には，学習の収束が速く少数のイテレーションで学習が収束するという特徴があるため，タプル増幅手法が有効に働くと考えられる．

タプル増幅による繰返し除去手法は，経験的に予測モデルの精度を上げることは分かっているが，収束に至るまで学習を行う手法ではない．収束に至るまで学習を進めるためには，Hivemall によってバッチ的に予測モデルを構築したうえで，同様の損失関数を採用するオンライン学習器で Hivemall で学習した予測モデルを更新することが考えられる [45]．

7.　むすび

本稿では，Hivemall によるスケーラブルな機械学習を実現するうえで得られた実践的な知見，およびその実現手法を述べた．

具体的に，本稿の技術面での貢献は次のとおりである．

- 従来の集約関数に基づく機械学習手法の問題点を指摘し（3.2 節），その打開策としてテーブル生成関数を利用した機械学習の並列実行を紹介した（4.1 節）．
- タプル増幅とバギングの組合せによって機械学習のイテレーションを削減して，Embrassingly parallel に機械学習を行う手法を提案した（4.3 節）．最終的に採用した map-local の shuffle を行う手法とともに，試行錯誤の過程で開発した単純なタプル増幅とその問題点についても実験結果と得られた知見を示した（6.3 節）．
- Hivemall と Hive に特有な機能を用いて機械学習の一連のフローを実行する形式を示した（5 章）．

近年のソフトウェア開発においてオープンソースソフトウェアが果たしてきた役割は大きい．データベース分野では PostgreSQL や MySQL が代表的なものである．機械学習分野でも LibSVM [36]，Liblinear [37]，scikit-learn [46]，Weka [47] といったオープンソースソフトウェアは学術利用から産業利用まで様々な用途で活用されている．機械学習ライブラリである LibSVM [36] や Liblinear [37] の解説論文は通常の研究論文と比較しても非常に多くの Citation を集めており，機械学習のメジャーな論文誌である Journal of Machine Learning Research には常設のオープンソーストラック [48] が存在する．このように機械学習分野においてオープンソースソフトウェアが果たす役割は大きい．

ソフトウェアとしての Hivemall の提供目的は，テラバイトまでスケール可能な機械学習ライブラリとして，上記のソフトウェアと同様にビッグデータを扱う各種研究開発や商業システムの諸問題の解決に広く役立つことである．実際に株式会社ロックオンとの共同研究では，テラバイトクラスの実データを利用したインターネット広告のコンバージョン率の推定を Hivemall を用いて実現している[*8]．また，複数の国内広告関連企業で広告コンバージョン率推定や広告クリック率推定に Hivemall が利用されている．今後も研究レベルの成果を公開品質に高めて Hivemall に実装する取り組みや解説ドキュメントの補充を継続的に行っていく所存である．

謝辞　本研究の一部は，科学研究費若手研究（B）（課題番号：24700111）ならびに基盤研究（A）（課題番号：24240015）の支援による．ここに謝意を表す．

参考文献

[1] Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Proc. OSDI*, pp.137–150 (2004).
[2] The Apache Foundation: Apache Hadoop, available from ⟨http://hadoop.apache.org/⟩.
[3] The Apache Foundation: Apache Hive, available from ⟨http://hive.apache.org/⟩.
[4] Shvachko, K., Kuang, H., Radia, S. and Chansler, R.: The Hadoop Distributed File System, *Proc. IEEE Mass Storage Systems and Technologies* (*MSST*), pp.1–10 (2010).
[5] Zukowski, M., Nes, N. and Boncz, P.: DSM vs. NSM: CPU performance tradeoffs in block-oriented query processing, *Proc. DaMoN*, pp.47–54 (2008).
[6] Ailamaki, A., Dewitt, D.J., Hill, M.D. and Skounakis, M.: Weaving Relations for Cache Performance, *Proc. VLDB*, pp.169–180 (2001).
[7] Alagiannis, I., Borovica, R., Branco, M., Idreos, S. and Ailamaki, A.: NoDB: Efficient Query Execution on Raw Data Files, *Proc. SIGMOD*, pp.241–252 (2012).
[8] Lin, J. and Kolcz, A.: Large-scale machine learning at twitter, *Proc. SIGMOD*, pp.793–804 (2012).
[9] ヤフー株式会社：Yahoo! JAPAN を支えるビッグデータ

[*8] http://www.lockon.co.jp/release/3074/

プラットフォーム技術，入手先 ⟨http://www.slideshare.net/techblogyahoo/yahoo-japan-28908739⟩.
[10] Hivemall project, available from ⟨https://github.com/myui/hivemall⟩.
[11] Apache Mahout, available from ⟨http://mahout.apache.org/⟩.
[12] Crammer, K., Dekel, O., Keshet, J., Shwartz, S.S. and Singer, Y.: Online Passive-Aggressive Algorithms, *J. Mach. Learn. Res.*, Vol.7, pp.551–585 (2006).
[13] Dredze, M., Crammer, K. and Pereira, F.: Confidence-weighted linear classification, *Proc. ICML*, pp.264–271 (2008).
[14] Crammer, K., Kulesza, A. and Dredze, M.: Adaptive regularization of weight vectors, *Machine Learning*, Vol.91, No.2, pp.155–187 (2013).
[15] Hoi, S.C.H., Wang, J. and Zhao, P.: Exact Soft Confidence-Weighted Learning, *Proc. ICML* (2012).
[16] Chu, C.T., Kim, S.K., Lin, Y.A., Yu, Y., Bradski, G.R., Ng, A.Y. and Olukotun, K.: Map-Reduce for Machine Learning on Multicore, *Proc. NIPS*, pp.281–288 (2006).
[17] Agarwal, A., Chapelle, O., Dudík, M. and Langford, J.: A Reliable Effective Terascale Linear Learning System, *Journal of Machine Learning Research*, Vol.15, pp.1111–1133, available from ⟨http://jmlr.org/papers/v15/agarwal14a.html⟩ (2014)
[18] Feng, X., Kumar, A., Recht, B. and Ré, C.: Towards a unified architecture for in-RDBMS analytics, *Proc. SIGMOD*, pp.325–336 (2012).
[19] Cortes, C. and Vapnik, V.: Support Vector Machine, *Machine Learning*, Vol.20, No.3, pp.273–297 (1995).
[20] Lin, J. and Kolcz, A.: Large-Scale Machine Learning at Twitter, *Proc. SIGMOD*, pp.793–804 (2012).
[21] Hall, K.B., Gilpin, S. and Mann, G.: MapReduce/Bigtable for Distributed Optimization, *Proc. NIPS Workshop on Leaning on Cores, Clusters, and Clouds* (2010).
[22] Bottou, L.: Online Algorithms and Stochastic Approximations, *Online Learning and Neural Networks*, Saad, D. (Ed.), Cambridge University Press (1998).
[23] Bottou, L.: Large-Scale Machine Learning with Stochastic Gradient Descent, *Proc. COMPSTAT*, pp.177–187 (2010).
[24] Chen, D.-R., Wu, Q., Ying, Y. and Zhou, D.-X.: Support vector machine soft margin classifiers: Error analysis, *The Journal of Machine Learning Research*, Vol.5, pp.1143–1175 (2004).
[25] Crammer, K., Dekel, O., Keshet, J., Shwartz, S.S. and Singer, Y.: Online Passive-Aggressive Algorithms, *J. Mach. Learn. Res.*, Vol.7, pp.551–585 (2006).
[26] Mann, G., McDonald, R.T., Mohri, M., Silberman, N. and Walker, D.: Efficient Large-Scale Distributed Training of Conditional Maximum Entropy Models, *Proc. NIPS*, pp.1231–1239 (2009).
[27] Breiman, L.: Random forests, *Machine learning*, Vol.45, No.1, pp.5–32 (2001).
[28] Larson, P.-Å.: Data Reduction by Partial Preaggregation, *Proc. ICDE*, pp.706–715 (2002).
[29] Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S. and Stoica, I.: Spark: Cluster Computing with Working Sets, *Proc. HotCloud*, p.10 (2010).
[30] Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J. and Fox, G.: Twister: A runtime for iterative MapReduce, *Proc. HPDC*, pp.810–818 (2010).
[31] Hellerstein, J.M., Ré, C., Schoppmann, F., Wang, D.Z., Fratkin, E., Gorajek, A., Ng, K.S., Welton, C., Feng, X., Li, K. and Kumar, A.: The MADlib analytics library: Or MAD skills, the SQL, *Proc. VLDB*, Vol.5, No.12, pp.1700–1711 (2012).
[32] Cloudera, Inc.: Cloudera Impala, available from ⟨http://impala.io/⟩.
[33] White, T.: *Hadoop: The Definitive Guide*, O'Reilly Media (2009).
[34] Weinberger, K., Dasgupta, A., Langford, J., Smola, A. and Attenberg, J.: Feature hashing for large scale multitask learning, *Proc. ICML*, pp.1113–1120 (2009).
[35] Melnik, S., Gubarev, A., Long, J.J., Romer, G., Shivakumar, S., Tolton, M. and Vassilakis, T.: Dremel: Interactive analysis of web-scale datasets, *Proc. VLDB*, Vol.3, No.1-2, pp.330–339 (2010).
[36] Chang, C.-C. and Lin, C.-J.: LIBSVM: A library for support vector machines, *ACM Trans. Intelligent Systems and Technology* (*TIST*), Vol.2(3), No.27, pp.1–27 (2011).
[37] Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R. and Lin, C.-J.: LIBLINEAR: A library for large linear classification, *The Journal of Machine Learning Research*, Vol.9, pp.1871–1874 (2008).
[38] Capriolo, E., Wampler, D. and Rutherglen, J.: *Programming Hive*, O'Reilly (2012).
[39] DeWitt, D.J. and Gray, J.: Parallel database systems: The future of database processing or a passing fad?, *ACM SIGMOD Record*, Vol.19, No.4, pp.104–112 (1990).
[40] Hivemall project Wiki, available from ⟨https://github.com/myui/hivemall/wiki/⟩.
[41] Google Inc.: Snappy – A fast compressor/decompressor, available from ⟨https://code.google.com/p/snappy/⟩.
[42] KDD Cup 2012, Track 2, available from ⟨http://www.kddcup2012.org/c/kddcup2012-track2⟩.
[43] Bradley, A.P.: The use of the area under the ROC curve in the evaluation of machine learning algorithms, *Pattern Recognition*, Vol.30, No.7, pp.1145–1159 (1997).
[44] Liu, D.C. and Nocedal, J.: On the limited memory BFGS method for large scale optimization, *Mathematical Programming*, Vol.45, No.1-3, pp.503–528 (1989).
[45] Yui, M. and Kojima, I.: A Database-Hadoop Hybrid Approach to Scalable Machine Learning, *Proc. IEEE 2nd International Congress on Big Data* (2013).
[46] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: Machine learning in Python, *The Journal of Machine Learning Research*, Vol.12, pp.2825–2830 (2011).
[47] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. and Witten, I.H.: The WEKA data mining software: An update, *ACM SIGKDD Explorations Newsletter*, Vol.11, No.1, pp.10–18 (2009).
[48] Journal of Machine Learning Research: Open Source Software Track, available from ⟨http://jmlr.org/mloss/⟩.

油井 誠　（正会員）

独立行政法人産業技術総合研究所情報技術研究部門データサイエンス研究グループ主任研究員．2003 年芝浦工業大学工学部工業経営学科卒業．同年（株）NEC 情報システムズ入社．グリッド・コンピューティングの研究開発に従事．2006 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．2008 年日本学術振興会特別研究員 DC2．2009 年奈良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（工学）．同年日本学術振興会特別研究員 PD．この間，早稲田大学 IT 研究機構（IT バイオ研究所）客員研究員およびオランダ国立数学情報学研究所（CWI）客員研究員．2010 年独立行政法人産業技術総合研究所入所．現在に至る．高性能データベースおよび機械学習の並列分散処理の研究に従事．日本データベース学会会員．

小島 功　（正会員）

1984 年京都大学大学院工学研究科博士課程前期修了，同年電子技術総合研究所入所．研究グループ長等を経て現在情報技術研究部門総括研究主幹．異種・分散データ統合や地理空間情報基盤に関する研究開発に従事．ACM 会員．OGC メンバ．RDA（Research Data Alliance）OAB メンバ．

（担当編集委員　田村 孝之）