大規模リアルタイム解析エンジン Jubatus の創り方

岡野原 大輔 海野 裕也（株式会社 Preferred Infrastructure）
熊崎 宏樹 小田 哲 （日本電信電話株式会社）

概要 NTT PF 研（現ソフトウェアイノベーションセンタ SIC）と株式会社 Preferred Infrastructure（PFI）は 2011 年に大規模リアルタイム解析エンジン Jubatus をオープンソースソフトウェア（OSS）として公開し，現在様々なビッグデータ活用の現場での利用検証を進めている．本稿では Jubatus の公開までの経緯，及びその過程でどのような試行錯誤，判断があったかについて述べる．特に Jubatus はリアルタイム，分散並列，そして深い解析という三つの目標をまず掲げ，その実現に向けて様々な試行錯誤を行った．この目標を実現する際に，どのような選択肢があり現在の構成を採用していったかについて述べる．また，Jubatus は異なる強みを持った複数の企業が共同で企画／研究／開発を行い，その成果を OSS として公開するという新しい研究開発の形をとって開発されている．こうした連携の背景や，そのメリットなどについて言及する．

1. Jubatus 開発の背景

Jubatus（ユバタス）は NTT ソフトウェアイノベーションセンタ（以下 NTT SIC）と Preferred Infrastructure（以下 PFI）が 2011 年より研究開発を開始し，2011 年 10 月より OSS（オープンソースソフトウェア）として公開[1]，その後も開発を継続しているプロジェクトである．

1.1 現在のビッグデータ解析

21 世紀に入り，あらゆる分野で巨大なデータが生まれるようになった．こうしたデータはサイズが非常に大きい，生成速度が速い，データが多様であるといった特徴があり，これらをビッグデータと呼ぶ．

元々ビッグデータ解析はウェブ業界で生まれ，特に Google や Amazon などは大量のデータを蓄積し，データを活用し，ビジネス上の競争優位性を高めていった．例えば，広告配信の最適化や，検索エンジンのランキングの最適化，EC サイトのレコメンデーションなどである．

こうしたビッグデータを活用しようという動きは，ウェブ業界から他の領域に広がりつつある．例えば，自動車，工場，病院，農業，製造業，エネルギーといった分野においては大量のデータが今後生成されると考えられ，これらのデータ活用を行うことで新たなビジネス開発や課題克服につなげられると期待されている．

新しい種類のセンサの開発や通信技術の開発，データ蓄積のための基盤，クラウド技術の整備が進むにつれ，ビッグデータ解析における課題は，データの収集・蓄積から，データをいかに解析し活用するかに移りつつある．

1.2 リアルタイム化するビッグデータ

次々と生成される大量のデータをリアルタイムに解析することは，現状の課題解決や新たなビジネス開発につながると考えられている．以下にいくつか例を挙げる．

東京電力が採用するスマートメーターでは， 30 分毎に電力情報が取得され電力使用量を最適化することがリアルタイム性の実現につながる[2]．自動車における渋滞情報や地域情報，災害時の情報共有はすでにいくつか実用化されている．ストリームデータの代表例である twitter では現在秒間 1 万弱の tweet 情報が発信され[3]，これらを分析することでマーケティングやブランディング調査に活用したいというニーズは高まっている．セキュリティの場面においてもマルウェアの情報や情報流出などにおいてはいち早く兆候を見つけることが重要である．この他，金融の最適化や，広告配信の最適化などはミリ秒単位のリアルタイム性が必要とされる．

ビッグデータの場合，データを蓄積せず，その場で処理し，データ自体は捨てられるストリーム処理が重要となる．

1.3 深い分析とビッグデータ

ビッグデータ解析は，単純な変換・集計処理からより深い分析を必要としている．例えば教師データを元に学習を行い分類モデルを学習するような機械学習である．

これら機械学習に代表される深い分析は従来バッチ的に行うものがほとんどであった．しかしオンライン学習技術の急速な進歩とともに，ほとんどの深い分析はオンラインで行うことが可能となった．例えば多クラス分類の学習や，教師無し学習（Hidden Markov Model など），

外れ値検出などである．そしてその学習速度はバッチ学習を凌駕するようになってきている．

2. Jubatus の特徴

Jubatus には大きく三つの特徴がある．一つ目はリアルタイム処理，二つ目は分散並列処理，三つ目は深い解析である．これらはこれまでの機械学習のオンライン化の流れと，大規模分散並列化の二つの流れを合流したものである（図 1）．Jubatus は，一見不可能と思われるこれら三つの特徴を同時に実現するために，緩やかなモデル情報の共有とよばれる新しいコンセプトを採用している．また，Jubatus が対象とするデータは多種多様に渡る．Jubatus は特徴抽出部分と，その特徴を用いた分析部分を分け，様々なデータ分析を行うための開発が容易に行えるような工夫をしている．他のシステムとの比較を表 1 に挙げる．これらの特徴について順に説明する．

表 1　Jubatus と既存技術との比較

	Jubatus	Hadoop	CEP	RDBMS
大規模データ蓄積	対象外	◎ HDFS	対象外	○ 中規模
バッチ機械学習	○	○	×	◎ SPSS
ストリーム処理	○	×	◎	×
分散機械学習	◎	○ Mahout	×	×

2.1 リアルタイム処理

リアルタイム処理は，データを投入した際に，そのデータのモデルへの反映および，分析が瞬時に行われることを意味する. 例えば, スパムデータの分類問題の場合，スパムであると判定された教師データを学習データとして与えれば，その情報がモデルに反映され，次に得られたメールの予測にはその学習結果が反映される．

これを実現するために，Jubatus ではオンライン機械学習を利用したストリーム処理を実現している．ストリーム処理では，データを貯める必要はなく，データは一度だけ投入すればよいという特徴がある. この特徴により，大量のデータは保存する必要がなく，情報を貯めるコストが不要となるという利点がある．

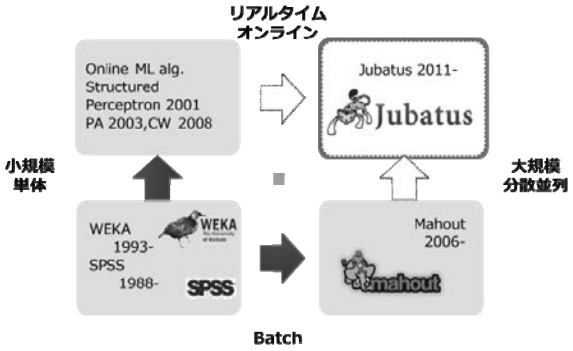

図 1　Jubatus の位置づけ．Jubatus はオンライン化と大規模分散並列化の二つの流れを汲んでいる．

2.2 分散並列処理

二つ目の特徴は分散並列を実現していることである．特にこれは近年の大規模データ処理基盤の殆どが採用しているスケールアウトによる性能向上を意味する．

Jubatus は 1 台では速度的にもしくは容量的に処理性能が足りない場合，マシンを追加することで処理性能を上げる事が可能である．これは後で述べるモデル情報の共有処理以外は，全て各ノードのローカル情報だけで処理ができるためである．ノード情報のレプリカを持つことで，耐障害性も達成することができる．

2.3 深い解析

三つの目の特徴は機械学習に代表される深い解析を実現するということである．

通常何らかの解析処理を行う場合，データを観察し要約した情報を利用して解析を行う．こうして要約された情報をモデル情報と呼ぶ事にする．

Jubatus では各ノード毎にこのモデル情報を格納し，データによって更新したり，それを利用して分析ができるようになっている．また，異なるノード間でモデル情報を共有する仕組みが備わっている．このモデル情報を利用して，様々な深い解析処理を Jubatus は実現する．

2.4 緩やかなモデル情報の共有

これら三つの特徴を同時に備えた大規模データ処理基盤システムは現時点で Jubatus 以外には存在しない．

Jubatus はこれら三つの特徴を備えるために，緩やかなモデル情報の共有を採用している. これは Jubatus では全てのサーバで同時刻に同じ結果が返される保証をせず，緩やかにモデル情報が共有され時間がたつにつれ同じ結果が返されるようになる仕組みである．

元々多くの統計分析では 100%常に同じ結果を返すこ

とが保証されているわけではない．例えば分類問題において精度が 97%であるものが 96.9%になったとしても実用上大きな問題はないだろう．Jubatus はこうした背景から緩やかなモデル情報の共有を採用した．各ノードのモデル情報は 1ー10 秒程度の一定間隔で交換し合い，時間が経つにつれてこれらのモデル情報が同一の情報を持つように収束していく．

2.5 特徴抽出と分析の分離

従来のデータ解析は特徴抽出が終わった後の数値データのみを対象にしており非定型データは扱えなかった．

Jubatus は様々な種類のデータを扱えるように元のデータから特徴情報を抽出する部分をプラグイン化している．例えばテキストデータからの特徴情報抽出では形態素解析や辞書情報などを利用した特徴抽出が行えるようになっている．この他にも行動履歴データやセンサデータ，画像や映像，音声などのメディアデータなど多種多様なデータからの特徴抽出が行えるようになっている．

そして，一度特徴が抽出されれば様々な深い解析を実現できるようになっている．

2.6 Update-Mix-Analyze (UMA)

Jubatus の処理は Update, Mix, Analyze の三つの組合せで構成される．アルゴリズムを設計する際はこれらの操作を定義するだけでよく，分散ロジック，データ共有方法，耐故障性などについては考えなくても良い．この考え方は MapReduce における Map 操作と Reduce 操作に似ている．

Update は与えられたデータに基づき，モデルを更新する操作である．この時のモデルはローカルに存在するモデルである．Mix は与えられた複数のモデルを混ぜる操作である．Analyze は与えられたデータを，現在のモデルを利用して解析し，解析結果を返す操作である．

Update と Analyze はユーザが呼び出す操作なのに対し，Mix はシステム側が自動的に呼び出す操作である．

3. 企業連携による研究開発と OSS

前章で述べた通り，Jubatus は他ソフトウェアが真似できない領域の問題を解くことが可能である．しかし筆者らは Jubatus を商用ライセンスとして提供せずオープンソースソフトウェア（OSS）として公開している．

筆者らがこれを OSS にする理由は，分散環境におけるオンライン機械学習処理を普及させることが，開発リソースの投資に対するレバレッジを効かせるために最適だと判断したからである．現在，Hadoop をはじめ分散処理基盤の多くは OSS として公開され，多くの開発者，研究者が関わってエコシステムを形成し，それと共に多くのユーザがこれまで触れることができなかった分散処理基盤に触れることが可能となっている．これにより，これまでみられなかった新しい利用事例やデータ活用の形が生み出されている．Jubatus も同じように OSS として公開することにより，多くのユーザに触れてもらい新しい利用事例やデータ活用の形を生み出していきたいと考えている．共同開発を開始するにあたって 2 社間で合意したのは，この「開発した成果は OSS 化しお互いに世の中に普及させる努力をする」という点である．

3.1 開発体制

共同開発体制の中には三つのサブグループが存在する．その内訳は，新しいアルゴリズムや方式を実際に実装してみて検証するグループ，それらを OSS としてリリースしたり，メンテナンスしていくグループ，そしてビジネスに結びつけていくために基盤部分以外で必要な周辺を作るグループである．本稿では，基盤部分の開発を担当している，前の 2 グループについて述べる．

コードの管理は git/github で行い，全員が参照，更新が可能である．状況の共有は常時グループチャットで行われる．チケット&議事録管理は Redmine で行われ，そして，週に一度顔を付き合わせてのミーティングを行っている．基本的にはオンラインでの開発だが，リリース直前は，Debian のバグスカッシュパーティを模した，一個所に集まった集中作業が行われることもある．実情に合わせた設計方針が立てられ，機動力があるメリットがある一方，長期的な開発ロードマップの共有は各開発者が密に連携をとって確認しあう必要がある．

企業同士の共同開発では，しばしば会社ごとの責任分界点を決めてインタフェースを設計し，お互いその中を開発するという体制が取られる．しかし，Jubatus では基本的には（当然，ある程度属人化しているとはいえ）開発に携わっている人すべてが，すべてのコードに対して責任がある．そのため，局所最適ではなく全体を見渡したリファクタリングが合意しやすいという面があると考えている．

Jubatus の開発では，git flow 式の master ブランチ，develop ブランチ，各種サポートブランチの他に両社が共有するプライベートレポジトリを運営している．Jubatus は枯れた技術を OSS 化するのではなく誰も実現したことがない技術を開発することを目標としている．そのため，実現可能性がまだ怪しい方式を検証し，評価するためにプライベートレポジトリを利用している．実際に検

証してみた結果，想定していたほどの精度・性能が出ずにお蔵入りしたコードも存在する．また，知的財産の観点から OSS として外部に出せるものかどうかを検討するためのバッファの意味合いもある．Jubatus はロードマップとしてこれからリリースする予定，開発する予定の技術を提示しているが，その詳細がこのプライベートレポジトリの存在によって，コミュニティに見えにくくなってしまうことは課題であると認識している．これに対して，定期的にロードマップをアップデートすることで利用者への今後のバージョンアップ，変更などの周知を図りながら開発に対するフィードバックを得られるように進めていきたいと考えている．

3.2 クライアントとライセンス

Jubatus のライセンスは，本体が LGPL v2.1，クライアント（IDL により自動生成される）が MIT ライセンスとなっている．例えばクライアントから Jubatus を使うだけの場合や，プラグインの形で新しいアルゴリズムを追加した場合でも，追加した部分をプロプライエタリなソフトウェアとして再配布することができる．

4. 開発の試行錯誤

本章では，これまでの章とは異なり，Jubatus 開発までの試行錯誤を，時系列順に追いながら記していく．

4.1 2011 年春 Jubatus プロジェクト発足

遡ること 2011 年 4 月，Jubatus のプロジェクトが始まった．当初は，はっきりとしたゴールもコンセプトもなかった．その当時あったのがオンライン機械学習については PFI，大規模分散処理については NTT SIC が得意していた領域であったのでこの二つを組み合わせれば新しい技術が生み出せるのではないかというざっくりしたアイデアだけだった．ビッグデータというキーワードも，当時は一部の企業が叫んでいる程度であった．そうした中でも筆者らは次の時代の世界を想像し開発を始めた．

4.1.1 Jubatus の設計と思想

最初からはっきりとした設計があったわけではなく細かい粒度で実装を行い，実験をしながら全体像を模索する必要があった．

主なモジュールは 4 つと決まった．メインの分類学習器，モデル情報を蓄えるストレージ層，生の非構造データを変換する特徴変換器，そしてデータのやり取りを行うサーバである．実装言語は全員が使える言語であることと，速度を重要視して C++となった．また，実装を助けるためのライブラリとして，PFI 社内で整備されていた pficommon という汎用ライブラリを利用することと，OSS 化を睨んで pficommon を外に公開する手続きが始められた．

最も基本的な機械学習の問題設定である多値分類問題を最初の学習問題として選んだ．多値分類問題であればオンライン学習や分散学習の既存研究もあったため実現性が高いと判断したからだ．実際に分類器自体のプロトタイプは 2 日で完成し，動作を確認した．

次に筆者らは，分類器中で保持されるモデルパラメータを保持する部分をストレージ層として抽象化し，複数のストレージを切り替えられるように実装した上で様々な分散モデルを検証した．一つ目はモデルデータを各サーバのローカルメモリに持っておき，サーバ間で通信してモデルの同期を取る方法である．この方法を検討したのには，MapReduce 上で Iterative Parameter Mixture (IPM) [4]と呼ばれる方法がうまくいくことがわかっていたためである． 二つ目は，既存の分散データベースにモデル情報を持つ方法である．具体的には HBase 中に重み情報を保持するという案を検討し，実装を行った．

この事前検討の中では，既に多くの評価が行われているデータセットである Pascal Large Scale Learnig Challenge [5]の webspam を利用し，アルゴリズムやクライアントの台数，サーバの台数などの組合せを順番に実験するスクリプトを構成して実験を行った．この実験は，IPM で行うと数時間で終わる実験セットが，分散 DB を用いると年単位でかかることがわかり，後者のアプローチが現状の技術水準では不適当であることが明らかになった．

4.1.2 特徴抽出機構：fv_converter

データ解析部分と，特徴抽出部分を分離するという方針は開始時から決まっていた．一般的に多くの機械学習ライブラリは，ベクトル形式のデータを入力として受け付けるため，テキストや画像などのデータは一度ベクトル形式に変換する必要があった．我々は，こうしたベクトル形式への変換部分が事前に用意されていないことが，機械学習技術の普及において大きな足かせになっていると認識していた．データ中のどういった特徴を利用してベクトルを構築するかは精度面で極めて重要であるにも関わらず，それをサポートする仕組みがあまりにも普及していないのだ．

実装の際はできるだけわかりやすい実装にしようとこころがけ，入力データとしてキーと値の集合が来ることを想定した．値の部分には任意の非構造データを入れら

れることを想定したが，最初のプロトタイプはセンサーデータなどで使う数値データと，自然言語処理などで使う文字列データのみをサポートした．特徴変換の仕組みも，「どのキーのデータに対して」「どういう処理を行うか」だけで表現できるようにした．「どのキー」は完全一致や正規表現で示せるようにし，「どういう処理」はプリセットでたくさん用意してユーザが簡単に選べるようにした．

サーバへのアクセスはまずは HTTP を用いることにした. 特に今回利用した pficommon では C++で簡単に HTTP サーバを書く仕組みがあったため，それを利用することとなった. パフォーマンスに関する懸念はあったものの，サーバが安定しない初期において，人が見て理解できることが最優先された．かくして，ひと通り生のデータを入れれば分散し学習する最初の Jubatus はできあがった．

4.2 2011 年夏 新機能開発と OSS 開発

2Q の目標は分散レコメンドエンジンの研究開発と，1Q の成果の OSS としての公開，そして Jubatus 公開の準備であった．

4.2.1 お蔵入りとなった Anchor Graph

とりわけ難航したのが，レコメンドエンジンの開発であった. 当時 PFI 内で注目していたのが, Anchor Graph [6] であり，これを元にレコメンドエンジンを大規模かつリアルタイムで実現することが目指された．

当初は大分楽観的であった．しかし，ことは順調に進まない．まず，Anchor Graph をオンラインの問題設定で利用すると, 手法そのものにランダム性が含まれており，挙動が安定しないことがみられた．サーバからのレスポンス時間も安定せず，分散になるとなおのことである．そして致命的なこととして，学習が進んでいくにつれ誤差が累積し，精度が全く出ないという問題が出た．こうして一度作った分散オンライン Anchor Graph は評価を行った上でお蔵入りとなった．

4.2.2 計算モデルの模索

こうした過程の中で, 個別に分散学習の方法を考えず，より汎用的で抽象的な計算モデルを構築できないかという検討が始まった．MapReduce はデータを分散して個別に処理を行う Map 処理と，処理済みのデータをマージする Reduce 処理の二つで分散処理を実現する. 多くのバッチ型のデータ処理がこの単純なモデルで記述できるということが人気の秘密であろう．同様に多くのオンライン学習を分散化させる計算モデルはどのようなものかを考え Update-Mix-Analyze モデル，緩いモデル共有といった概念は生まれてきた．

4.2.3 ZooKeeper に管理される Jubatus

さて，分散システムの開発も重要な局面を迎える．分散システムの構築には，参加するノードの管理が欠かせない．最初の実装ではサーバ一覧はクライアント側で保持し，クライアントがランダムにサーバを選んで接続するという設計であった．しかし，実際の分散システム内ではサーバが故障することを前提にした実環境での利用を見据える必要がある．そこで，クラスタに参加しているメンバの一覧を ZooKeeper [10]で保持する方針に変更した．ZooKeeper は Apache Software Foundation が分散環境における様々なサービスを一元的にまとめたソフトウェアである．この機能の実現には Ephemeral ZNode という ZooKeeper の機能を利用し，サーバの故障や追加が自動的に ZooKeeper 中のメンバ一覧に反映するようにした．

また, クライアントサーバ間の通信インタフェースは, HTTP/JSON と MessagePack-RPC [8]の 2 種類をメンテナンスし続けていたが，将来的にネットワークシステムの中枢に Jubatus を導入するときにパフォーマンスが重要となる事を考えて MessagePack-RPC に一本化した．MessagePack-RPC は，データのシリアライズ形式として MessagePack を採用した RPC の仕様である．サポートするプロトコルが減ったとはいえ，多くの言語のクライアントをサポートするのはコストが高く現実的ではないと判断し, リリース時にサポートする言語は, C++と Python だけにせざるを得なかった．少し先の話になるが，リリース後まもなく Java のサポートも開始した．

ZooKeeper 上のサーバ一覧情報にアクセスするためには，図 2 左のように ZooKeeper のクライアントライブラリを利用する必要がある．しかし，各プログラミング言語用の ZooKeeper クライアントを利用するのではアプリケーションが開発しにくいと判断し，サーバ一覧の管理やメンバ発見の機能は JubaKeeper という別サーバを ZooKeeper と同一のマシン上に起動することとした．これにより，ZooKeeper に直接アクセスするのは JubaKeeper だけとなり，各言語のクライアントから JubaKeeper に問い合わせることで ZooKeeper と直接通信する事なくサーバの一覧が取得できるようになった．

この時点で，図 2 中央のように，JubaKeeper は単に ZooKeeper を隠蔽する，Jubatus サーバ発見のためのプロパティサービスのような設計であった．そのため，クライアント側で 1)JubaKeeper を参照しサーバ一覧を取得，2)一覧をクライアント側でキャッシュ，3)実際のサーバ

にアクセスという実装を 3 種類の言語でサポートする必要があり，これがサポート言語を増やそうとする開発者の負担となることが容易に予想できた．そこでクライアントの実装をなるべく減らすために，1)，2)の機能を JubaKeeper に負担させることとし，クライアントは JubaKeeper にアクセスするだけで透過的に各サーバにアクセスできるようになった（図 2 右）．クライアントはそれが単体のサーバなのか分散している Jubatus の前にいる JubaKeeper なのか意識しなくてもよくなる上，クライアントの実装は単に RPC を呼ぶだけになった．こうした作業はリリースの 2，3 日前に方針を変更し行われた．

4.3 2011 年秋 報道発表と初のリリース

10 月に入り，Jubatus の報道発表，リリース，そして発表会に向けた準備が進められた．

4.3.1 OSS 公開に向けた作業

公開に先立って既存システムの調査も行われた．ここまでは Mahout [11]や liblinear [12]などの，分散バッチ学習や，単体オンライン学習ライブラリとの比較という形で考えてきていたが，いざ公開する段になるとストリーム処理基盤や CEP(Complex Event Processing)システムとの差について認識する必要が出てきた．世の中に存在するストリームプロセッシング技術や複合イベント処理 (CEP) 技術，またその製品やオープンソースソフトウェアの調査を研究界，産業界を問わず広く行った．その調査を通じて，リアルタイム処理への世の中の需要があること，そして既存のどの技術も機械学習を第一に挙げてはいないということが見えてきた．このことは，需要があるのかどうかも不明瞭なままプロジェクトを進めてきたメンバにとっての励みとなった．

公開の前後は一気に忙しくなった．特許調査，ライセンスをどうするか，パッケージングはどうするか．ソースコードのクオリティはお世辞にも高いとは言えなかったが，最低限動くことだけが目指された．公開の日は，報道発表で「明日リリースします」というために，発表の翌日とすることが決まった．各社のプレスリリースを用意し，発表資料も作りこまれた．

こうした公開や発表の準備とともに開発もこれまでと同様に進められた．今期のアルゴリズム開発は，前期にうまく行かなかったレコメンダーを，LSH などのもっとよく知られた手法で作り直すこと，それから回帰や統計などの基礎的な機能を増やすこととなった．いずれも機能自体はシンプルであったため，順調に開発は進んだ．

4.3.2 メンテナンスコスト爆発を抑え込め

サーバとクライアント部分の開発に大きな変換点を迎える．最初の分類器は，設定の変更や学習などの 7 種類の API を備えていた．全 API の各言語用クライアント，および C++サーバを実装することとなり，API の維持コストは【API 数×サポートする言語の数】に比例する．将来的に API とサポート言語を増やすことを考えると，API の維持コストは悩ましい課題であった．

しかし，JubaKeeper のプロキシ化によって，クライアントライブラリは単なる MessagePack−RPC の呼び出しだけとなり，非常に簡素になった．そのため，クライアントの実装は機械的に作成できるようになり，自動生成の検討を始めた．まず MessagePack-RPC 用のインタフェース定義言語である MessagePack-IDL に注目したのだが，実用的な実装がなかった．そこで筆者らは mpidl と呼ばれる MessagePack-IDL のコードジェネレータを Haskell で実装した．この実装において，解析表現文法のパーザージェネレータである Peggy を利用して，安全に動作す

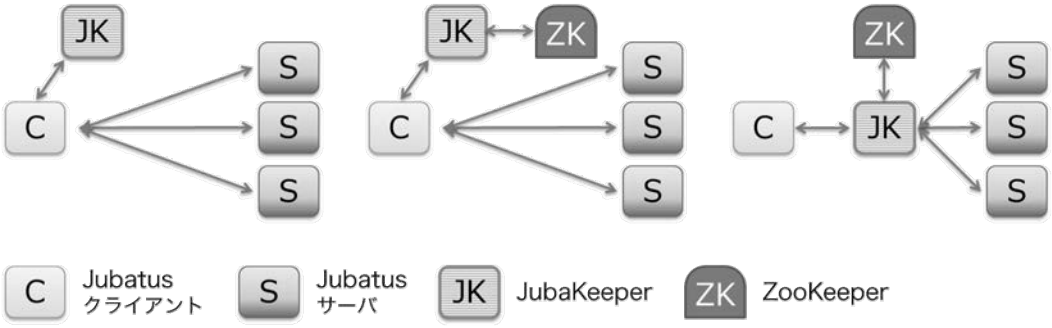

図 2　クライアント，サーバ，JubaKeeper，ZooKeeper の構成の変遷．（左）クライアントが直接 ZooKeeper を参照したため，各クライアント言語用の ZooKeeper クライアントが必要．（中央）JubaKeeper を仲介することで ZooKeeper クライアントを隠蔽．（右）JubaKeeper が ZooKeeper やサーバとのやり取りを全て仲介．

るコードジェネレータを短いコードで実現した．なお，このPeggy自体もJubatusの開発メンバによって開発されている．mpidlは今でもJubatusで利用されており，新しいアルゴリズムを追加する際に必要なツールの一つとなっている．IDLさえ記述すればクライアントコードは自動生成できるので，7種のAPIに対する実装コストは【IDLの記述＋JubaKeeperの実装＋サーバの実装】の3個所のみとなった．

残る問題はJubaKeeperとサーバの実装である．実際に複数のIDLに対してそれぞれ実装してみると，サーバもJubaKeeperもほぼ定形な実装で，両コードの自動生成の要望が高まった．そこで，IDLからサーバとJubaKeeperのコードも生成するjenerator というツールを作成した．これにより，IDLさえ記述すれば，C++のヘッダや関数のひな形を生成できるようになった（図3）．

これにより，Jubatusのサーバ開発はIDLを用いてAPIとアノテーションを指定するだけであとは自動生成されるテンプレートコードにロジックを実装するだけになった．これは，機械学習アルゴリズムには明るいが分散システムなどのシステムプログラミングは苦手な研究者などでも，容易にJubatusの仕組みを利用できるようになることを意味している．このような仕組みをJubatusフレームワークと筆者らは呼んでいる．実際に，Jubatusチームに加入したメンバが，事前知識がほとんどないままにJubatus上で動作するサーバを1日で実装できた．

4.4 2011年冬 グラフ解析への挑戦

これまで技術ベースで進めてきたアルゴリズム開発であったが，この次は，実際のデータに適用しやすそうな方式を開発する，ということが決まった．

4.4.1 リアルタイムに変化するグラフの解析

リアルタイムでストリーム的に流れる情報として，Twitterのデータが候補に挙がった．話題ごとの中心（インフルエンサー）やユーザ同士の距離をリアルタイムに知ることができれば面白いのではないか，という議論が行われ，データ同士の関連を表す解析手法として，グラフ解析が候補に挙がった．ちょうど世の中ではグラフDBが流行していた時期である．グラフは一般的に効率よく分割することは難しい．さらに，中心性の解析（PageRankの計算）やデータ同士の距離（最短経路）は全体を見渡さなければならないため，リアルタイムに分析を行うことは，非常に難しい．ところが，このグラフ解析の分散処理は難航する．秋までに構築したUpdate-Mix-Ananyzeの計算モデルにうまくはまらないのだ．今までの手法はいずれもUpdate操作は各サーバで独立に行われた．局所的な更新が，大局的には大きな影響を与えない，という前提が成り立ったためだ．一方でグラフ解析の場合，ノード間に局所性をみつけるのは難しく各ノードの処理を独立で行うという前提が成り立ちづらかった．最終的に，グラフ情報の更新（リンクの張り替えなど）がオンラインでやってきて，PageRankや最短路の情報はMixの際に隣の計算ノードに波及されるという方針で固まった．これは，実質的にMix時にほとんどの処理が行われ，Update時には大きな変更を行わない，Bulk Synchronous Processing [7]の計算モデルに近い形がとられ，今まで考えていた計算モデルを逸脱した使い方をすることとなった．グラフ解析に対する解析手法だけ，それ以外と異なり，この分野の難しさが窺える．

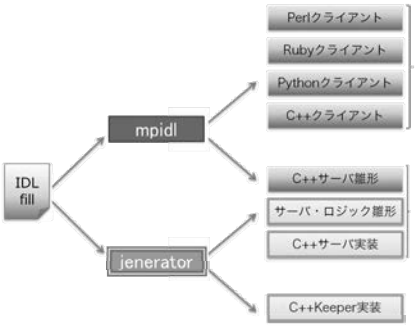

図3 IDLファイルから自動生成されるサーバとクライアントコードの関係．

実験ではランダムグラフ，スケールフリーグラフ，平面グラフなどの人工グラフを生成し，評価を行った．比較対象として用いたneo4j [9]がOut of Memoryで次々に落ちる中，Jubatusは同等の精度と，一桁速い1μ秒/ノードの更新速度を記録し続けた．

5. これからのJubatus

前章で，Jubatusが試行錯誤のもとに生まれてきたということが，伝わったかと思う．新しいアルゴリズム，先進的な仕組みの陰には，様々な試行錯誤の名残，実装上の問題，未完の機能，またこうした試行錯誤の代償として，非効率な実装や，見通しの悪い設計が残っている．これからも，Jubatusは新しい機能とともに，実運用に耐えられるような改善を行っていかなければならない．

5.1 これからの課題

まずは利便性を向上させたい．公開以降，インストールできない，動かせないという声を沢山頂いた．パッケージングやマニュアルの整備，サンプルの充実といった

作業をしていかなけれならない．Blog や Twitter などのメディアを通じた発信を充実させる必要もある．

それから，運用を見据えた運用機能の強化が必要である．可用性を上げること，サーバの動的な追加削除が行えること，十分な問題把握ができるようなログ出力を整備すること．今後，実システムで利用できるように，運用機能の整備が必要である．

新しいアルゴリズム，機械学習手法の検討は今後も続けていく．特に機械やセンサの生成する連続値の大量の時系列データを捌くのが今後の大きな課題の一つである．人間の生成するテキスト系のデータは，人口の上限に必ずバウンドされるため，今後劇的に増えづらい．一方で機械は無尽蔵にデータを生成することができる．こうしたデータを必要十分に解析できる基盤，アルゴリズム，学習手法の検討をする必要があるだろう．

5.2 Jubatus 開発に参加したい方へ

Jubatus に関わる技術分野は，単に分散システムだけではなく，機械学習はもちろんのこと，ネットワーク，自然言語処理，画像解析，データ構造など多岐にわたる．これは，今までどちらかと言えば疎遠であった分散システムと機械学習という二つの技術領域を同時に研究開発する必要があるためでもある．ぜひとも各分野の方々からのフィードバックを貰いたい．

まずは使ってみて欲しい．初期の頃はソースしか公開していなかったが，現状では RPM などのパッケージングも行われた．Jubatus はサーバ・クライアント型で動くため，サーバを起動させれば様々なクライアント言語から Jubatus を利用することができる．クライアントプログラムから，どのように利用するのかわからない，という声が大きかったため，サンプル集を整備している (https://github.com/jubatus/jubatus-example)．なるべく小さめのプログラムで，利用の仕方がわかるように書いているので，是非活用して欲しい．

要望や問題はメーリングリストや github の issue 上で受け付けている．こうした情報を提供していただけるだけでもありがたい．

そして，開発に参加してみたい方．開発は github 上で行われているので，pull-request を送る形でパッチを投げて欲しい．新しい機能の追加のみならず，バグの修正や使い勝手の向上，はたまたタイプミスの修正などなんでも結構．ご自身の研究成果を Jubatus 上で実装したり，また新しい分散機械学習の仕組みを Jubatus 上で実験していただくのも歓迎である．

5.3 Jubatus の教訓

分散システムは Hadoop や ZooKeeper のように基盤となる OSS はかなり成熟しつつある．そのため新たにソフトウェアを開発するならば何かしら機能特化した分散処理を想定しなくてはならない．我々が運が良かったのは基盤からアプリケーションまで，最初から専門の異なるメンバで構成したことである．その結果として，ユニークなソフトウェアの方向性を探れたのではないかと思う．ただ，こうした取り組みを行うには一つの研究室，一つの研究グループで行うのは難しく，複数のグループでコラボレーションする必要があるだろう．その場合，コミュニケーションコストが問題になることが往々にしてあるが，例えばチャットや電話会議システムなどのテクノロジーで解決できる部分は多い．これらのツールを積極的に取り入れて，異文化をお互いに受け入れながら，自分たち流の連携スタイルを構築できたことも上手く進められた要因ではないかと思う．

また，我々が重視しているのは開発の継続性についてである．現状，多くの分散処理ソフトウェアが，最初に公開された状態で放置されていることが見受けられる．これは新しい技術を開発するために研究ベースでソフトウェアを作ることが多くなり，多くの研究開発と同じように属人化してしまいがちなことに起因していると考えている．分散処理界隈でよく聞かれるジョークとして「分散処理ソフトウェア○○の単一障害点は開発者である」という話がある．この傾向は開発に必要な技術スキルセットが増えるにつれ顕著になる．単一障害点が存在しないことが今や当たり前の分散処理技術において，業界全体で取り組むべき課題であろう．

参考文献

1) Jubatus 公式サイト http://jubat.us/
2) 東京電力スマートメーター仕様に関する意見募集の結果および「基本的な考え方」について
http://www.tepco.co.jp/corporateinfo/procure/rfc/repl/rfc-com_re-j.html
3) 1 年を振り返る：1 秒あたりのツイート数
http://yearinreview.twitter.com/ja/tps.html
4) R. McDonald, K. Hall and G. Mann, “Distributed training strategies for the structured perceptron”, NAACL 2010
5) Pascal Large Scale Learning Challenge,
http://largescale.ml.tu-berlin.de/about
6) W. Liu, J. Wang, S. Kumar and S. Chang, “Hashing with Graphs”, ICML 2011
7) Bulk Synchronous Parallel, Wikipedia
http://en.wikipedia.org/wiki/Bulk_synchronous_parallel
8) MessagePack-RPC https://github.com/msgpack/msgpack-rpc
9) Neo4j http://neo4j.org/
10) Apache ZooKeeper http://zookeeper.apache.org/
11) Apache Mahout http://mahout.apache.org/
12) LIBLINEAR http://www.csie.ntu.edu.tw/~cjlin/liblinear/

岡野原 大輔（正会員）
E-mail: hillbig@preferred.jp
2010 年東京大学大学院情報理工学系研究科コンピュータ科学専攻博士課程修了，博士（情報理工学），2006 年に株式会社 Preferred Infrastructure を共同で創業，現在同取締役副社長，統計的自然言語処理，機械学習，簡潔データ構造，大規模データ処理などに興味を持つ．

海野 裕也（非会員）
E-mail: 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．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 年より株式会社 Preferred Infrastructure 入社．研究開発部門配属．自然言語処理，機械学習の研究開発に従事．言語処理学会会員．

熊崎 宏樹（非会員）
E-mail: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士課程修了，修士（工学），同年日本電信電話株式会社入社．ソフトウェアイノベーションセンタにて分散処理の研究に従事．並行データ構造，データベース処理に興味を持つ．

小田 哲（非会員）
oda.satoshi@lab.ntt.co.jp
2005 年慶應義塾大学大学院理工学研究科基礎理工学専攻修士課程修了，修士（工学），同年日本電信電話株式会社入社．情報流通プラットフォーム研究所にて，暗号の研究に従事．2012 年よりソフトウェアイノベーションセンタにて分散処理の研究開発に従事．

投稿受付：2012 年 09 月 15 日
採録決定：2012 年 11 月 26 日
編集担当：吉野松樹（日立）