特集号 招待論文

OSSを活用したTwitterデータ提供システムの構築

関 堅吾[†1]　金子 崇之[†1]　山下 真一[†1]

[†1] （株）NTTデータ

2013年10月現在，Twitterはアクティブユーザ数が2億人以上，1日の投稿ツイート数は5億件以上[1]と，最も活発なWebサービスの1つである．（株）NTTデータが運営する「Twitterデータ提供サービス」は，そのような大量の公開ツイートをFirehose APIを通じて収集し，すべての日本語ツイートを，特性の異なる複数のWeb APIによりユーザに提供するサービスである．Firehoseを利用するシステムは，トラフィックの継続的な増加，ツイート数の瞬間的な急増，データの再取得の難しさなど，さまざまな課題に対処する必要がある．本論文では，OSSを全面的に活用し，これらの課題に対応したシステムの事例を紹介する．

1. はじめに

（株）NTTデータは，2012年9月に米Twitter社と契約を締結し[2], 同年12月から「Twitterデータ提供サービス」を開始した[3].

このサービスは，Twitter社が提供するAPI「Firehose」[4]から，Twitterに投稿されたすべての公開ツイートを即時に受信し，公開ツイートのうち投稿内容に日本語を含むツイート，もしくは投稿者の言語設定が日本語であるツイート（以下，これらを「日本語ツイート」と総称する）をシステムに蓄積するとともに，後述するAPIを通じて利用者に提供する．

本システムは一部の構成要素を除き，全面的にOSSを用いて構築した．これは，自分たちが仕組みを理解し，必要に応じて解析・修正できる，透明性の高いソフトウェアでシステムを構築することで，高品質のサービスを提供するためである．

本論文では，まずTwitterデータ提供サービスの概要とシステム構成，採用したOSSの利用方法を紹介する．次に，今回の構成を選択した背景として，Firehoseを利用するシステムが抱える課題と，本システムがそれらの課題にどのように対処したかを説明する．次いで，大規模データを蓄積・検索する上での工夫を紹介し，最後に，今回の開発を通じて得られた知見を紹介する．

2. Twitterデータ提供サービス

2.1 サービス概要

Twitterデータ提供サービスで公開している，4種類のAPIを**表1**に示す．

サンプルホース，フィルターホースは，Firehoseから受信したツイートデータを，即時にユーザに配信するAPIである．前者は日本語ツイート全体のうち10%を，後者はユーザが指定した検索条件に合致する日本語ツイートすべてを配信する．TwitterのStreaming APIと同様に，投稿されたツイートと投稿者の情報をJSON形式に整形し，HTTP Streamingで配信する．

リアルタイムサーチ，ヒストリカルサーチは，過去に

表1　Twitterデータ提供サービスで公開しているAPIの一覧

	通信方式	即時性	過去ツイートの検索範囲	指定可能な検索条件と演算子
サンプルホース	Streaming	ほぼ即時に配信	不可	指定不可
フィルターホース	Streaming	ほぼ即時に配信	不可	キーワード，除外キーワード 単純なAND, ORの組合せ
リアルタイムサーチ	REST	ほぼ即時に検索結果に反映	直近32日分	キーワード，アカウント名，投稿日時 AND, OR, NOT, 括弧
ヒストリカルサーチ	REST 検索結果はStreamingでダウンロード	3時間程度遅れて検索結果に反映	直近13カ月分	キーワード，アカウント名，投稿日時 AND, OR, NOT, 括弧

 2014 ©Information Processing Society of Japan

投稿されたツイートを検索するためのAPIである．前者はTwitterのREST APIと同様，HTTPリクエストによるクエリに対し，一定数までのツイートをHTTPレスポンスとして返却する．検索可能な期間は直近32日分である．後者は，蓄積基盤上に格納された過去13カ月分のデータから，該当するツイートをバッチ処理で抽出する．バッチ処理の検索条件の指定はREST APIで行い，バッチ処理完了後に抽出したデータをHTTP Streamingで配信する．

2.2　システム構成

システムの構成を**図1**に，採用したOSSの一覧を**表2**に示す．システムは大きく主系と受信蓄積系に分かれる．主系はさらに受信部，蓄積部，配信部の3つに分かれる．

本節では，データの流れに沿って主系の各構成要素について説明した後，受信蓄積系について説明する．

2.2.1　受信部

受信部は，Firehoseからツイートデータを受信し，日本語ツイートのみにフィルタリングして蓄積部と配信部に渡すサブシステムである．受信サーバとフィルタリングサーバ，日本語アカウントDBから構成される．

受信サーバはFirehoseに接続し，Streamingで受け取ったデータをツイート単位に分割し，ラウンドロビンでフィルタリングサーバの入力キューに投入する．受信サーバの冗長化にはApache ZooKeeper[5]を使用している．

フィルタリングサーバは，入力キューからツイートデータを取り出し，日本語ツイートであれば蓄積キューと配信キューの両方に投入する．併せて，そのツイートのアカウントを「過去に日本語ツイートを投稿したアカウント」（以下，日本語アカウントと呼ぶ）として日本語アカウントDBに登録する．フィルタリングサーバ上のキューはすべてRabbitMQ[6]を使用している．

日本語アカウントDBに蓄積した情報は，3.2.1.5節で説明するツイート流量の絞り込みに利用している．このDBは2台のPostgreSQL[7]の間で同期レプリケーションを行い，それらをPacemaker[8] + Heartbeat[9]によって現用ー待機構成にしている．実際にはこれらを組み合わせたソリューションであるPG-REX[10]を使用している．

2.2.2　蓄積部

蓄積部は，受信部のキューから取り出したツイートデータを，時間ごとに集約して蓄積基盤に格納するサブシステムである．格納したツイートデータはヒストリカルサーチで利用する．蓄積部は出力サーバ，蓄積基盤，バッチサーバから構成される．

出力サーバは，フィルタリングサーバ上の蓄積キューに格納したツイートデータを取り出し，その内容を随時ローカルファイルに書き出す．そして，5分おきに1ファイルとして，後述する蓄積基盤上のステージング領域に出力する．蓄積キューと出力サーバの間は，それぞれの蓄積キューにすべての出力サーバが接続するメッシュ

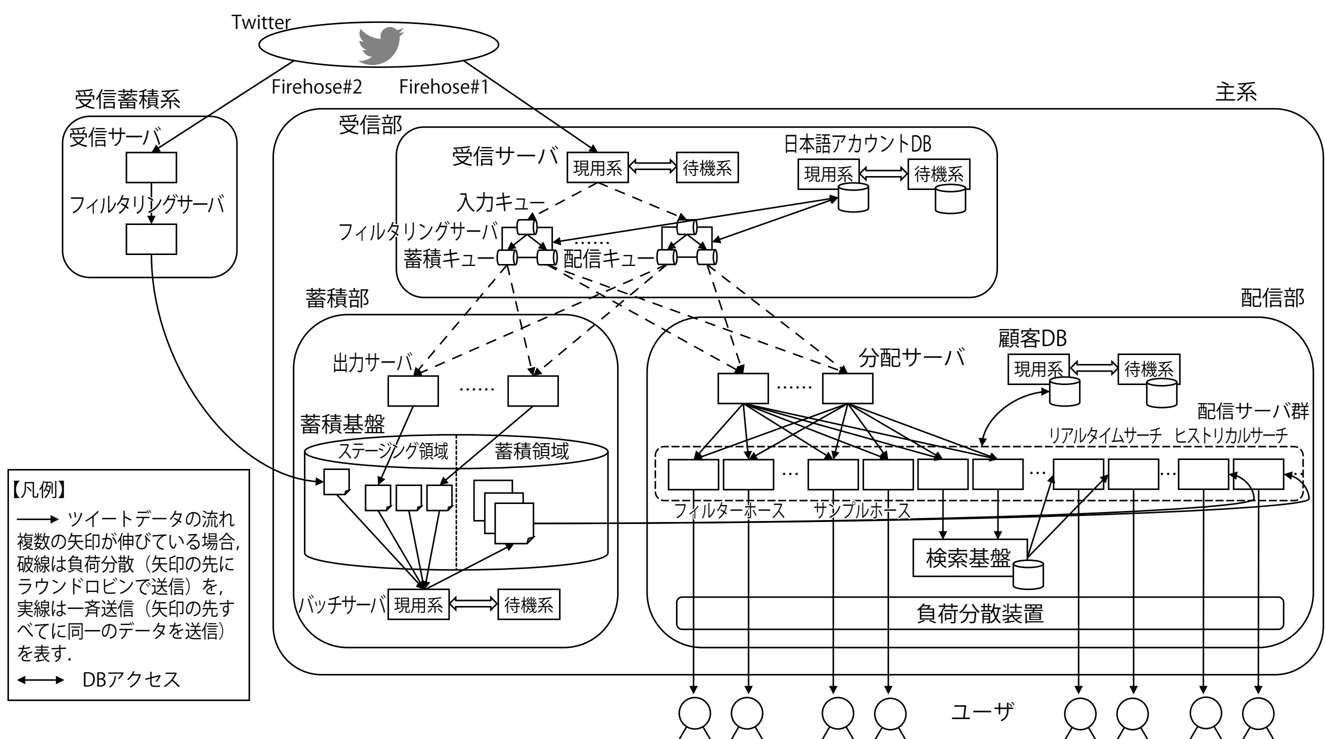

図1　Twitterデータ提供サービスのシステム構成

表 2　本システムで採用した OSS の一覧

OSS	利用しているサーバ	用途
Apache Hadoop	蓄積基盤	ツイートデータの格納，ヒストリカルサーチ
Apache HTTP Server	配信サーバ	ツイートデータの配信
Apache Tomcat	配信サーバ	ツイートデータの配信
Apache ZooKeeper	受信サーバ	現用－待機構成
	蓄積基盤	現用－待機構成 (NameNode HA)
Guava	フィルターホース配信サーバ	ツイートデータの絞り込み（ハッシュ関連ユーティリティに含まれるブルームフィルタを使用）
Pacemaker + Heartbeat	日本語アカウント DB	現用－待機構成
	顧客 DB	現用－待機構成
	バッチサーバ	現用－待機構成
PostgreSQL	日本語アカウント DB	日本語アカウント情報の格納
	顧客 DB	ユーザ情報の格納
RabbitMQ	フィルタリングサーバ	ツイートデータのキューイング，出力サーバ・分配サーバに対する負荷分散
ZeroMQ	分配サーバ	配信サーバへのツイートデータの一斉送信 (Publish)
	配信サーバ	分配サーバから送信されたツイートデータの受信 (Subscribe)

構成となっている．

蓄積基盤にはApache Hadoop[11]を使用している．Hadoopクラスタ上のファイルシステムであるHDFSには，出力サーバがファイルを出力するステージング領域と，ステージング領域上のファイルを一定時間ごとに集約したファイルを置く蓄積領域の，2つのディレクトリを設けている．ヒストリカルサーチでは，蓄積領域に移動したファイルを使用する．

バッチサーバでは，蓄積基盤のステージング領域に格納されたツイートデータを蓄積領域に移動する処理や，ヒストリカルサーチの検索処理，各種の集計処理などを定期実行している．

2.2.3　配信部

配信部は，ユーザからの接続を受け付け，APIの種類や検索条件に応じたツイートデータをユーザに配信するサブシステムである．分配サーバと，配信サーバ群，検索基盤，顧客DB，負荷分散装置から構成される．

分配サーバは，フィルタリングサーバ上の配信キューに格納したツイートデータを取り出し，下流の配信サーバ群にそのデータを一斉送信する. 配信キューとの間は，出力サーバと同様，各キューにすべての分配サーバが接続するメッシュ構成である．分配サーバとストリーム系配信サーバとの間の一斉送信は，ZeroMQ[12]のPub-Sub配信[13]によって行う．

配信サーバ群のうちサンプルホース配信サーバ，フィルターホース配信サーバ（以下，これら2つの配信サーバを「ストリーム系配信サーバ」と総称する）は，分配サーバから受け取ったツイートデータをユーザに配信する．前者はツイートをサンプリングレート（通常は10%）でサンプリングし，後者は，ユーザから指定された条件に合致するツイートのすべてを配信する．リアルタイムサーチ配信サーバ，ヒストリカルサーチ配信サーバは，それぞれ検索基盤・蓄積基盤と連携して条件に合致するツイートデータを抽出し，結果を返却する．配信サーバはすべてWebサーバにApache HTTP Server[14], WebアプリケーションサーバにApache Tomcat[15]を使用している．

検索基盤は，サンプルホース配信サーバから，サンプリング率100%（＝サンプリングなしの全量）での配信を受け取り, 検索インデックスを即時に更新する．通常，本システムがツイートデータを受け取ってから数秒後にはリアルタイムサーチの結果に反映される．我々は，他のサービスでOSSの検索エンジンであるApache Solrを使っていたが，今回のような常時配信されてくる大量のデータを即時にインデックス化できるかどうかの指標を持っていなかった．一方，検索エンジン分野に強みを持つ（株）Preferred Infrastructureでは，このようなデータの取り扱いにも知見を持ち，即時にインデックス化する技術についても実現の目処が立っていた．そこで，サービスを早期に立ち上げるため, 検索基盤にはOSSではなく，同社のSedue for BigData[16]を採用した．

顧客DBは，ユーザが利用できるAPIや契約期間などを管理しているDBであり，配信サーバへのリクエストはこのDBの情報を使って認証する．日本語アカウントDBと同様, PG-REXを用いた現用－待機構成としている．

負荷分散装置は, ユーザからのリクエストを受け付け，URL中に含まれるAPI名に従い，対応する配信サーバにそのリクエストを振り分けることで，背後の配信サーバに対する負荷分散と冗長化を担う．

2.2.4 受信蓄積系

受信蓄積系は，Firehoseと主系との経路間障害や，主系で致命的な障害（受信サーバの両系同時故障や災害など）などが起こった場合に備え，主系とは別経路でツイートデータを収集しておくためのシステムである．Firehoseからツイートデータを受け取り，日本語ツイートのみにフィルタリングした後，そのデータをファイルに出力し，主系に転送する．主系で受信したデータと受信蓄積系で受信したデータは，最終的に主系で重複を排除した形でマージする．災害の回避に加え，Twitter側のシステムと通信経路上近くなることを考慮し，地理的に近い海外のデータセンタに構築している．

3. 本システムの開発における課題と対策

本章では，前章で述べたシステム構成を採用した背景として，Firehoseを扱う上で重要となった課題を挙げ，それらの課題に本システムがどのように対処しているかを説明する．Firehoseそのものを扱うわけでなくとも，類似の課題を抱えているシステムを設計する際の参考になるものと期待する．

3.1 本システムの開発における課題

本システムの開発にあたり，特に重要な課題は以下の3点であった．

3.1.1 トラフィックの継続的な増加

Twitterの公式ブログによると，2007年には1日あたり5,000件だった投稿ツイート数が，2008年には30万件，2009年には250万件，2010年には5,000万件，2011年2月には1.4億件と，指数関数的に増加している[17][18][19]．2013年10月時点で，Twitterへの1日の投稿数は5億ツイートを超えており，成長の勢いは衰えていない[1]．

5億ツイート／日を単純に秒間平均ツイート数（以下，TPSと呼ぶ）に均すと，6,000TPSを処理する必要がある．さらに，1日の間には流量の波があるため，ピークの時間帯にはその数倍のトラフィックを処理する必要がある．

さらに，Firehoseを含むTwitterのStreaming APIでは，クライアント側の受信速度が遅いとTwitter側から強制的に接続が切断されてしまう[20]．そのため，システムは現在の流量に対して十分な処理能力を備えるだけでなく，Twitterの成長に見合った拡張性を確保することで，切断を防ぐ必要がある．

3.1.2 ツイート数の瞬間的な急増

前述のトラフィックはあくまで平常時のものであり，Twitterでは実世界のイベントを契機としてツイート数が瞬間的に急増することがある．2013年10月時点で，過去最高のTPSはアニメ映画のテレビでの再放送に伴う2013年8月の143,199TPS, 2位は2013年の元旦に記録された33,388TPSである[21]．前者は前節で述べた平均TPSの24倍である．システムはこのような瞬間的なバーストを吸収する必要がある．

3.1.3 データの再取得の難しさ

Firehoseからは，Twitterに投稿されたツイートの情報が即時に配信される．これは，一度データを受信し損ねると，そのデータを再度取得する手段が基本的には存在しないことを意味する☆1．そのため，システムは単一障害はもとより，さらに大規模な障害に対しても，ツイートデータの収集を継続できる必要がある．

3.2 課題に対する対策

前節で挙げた課題に対し，本システムが取った対策を説明する．

3.2.1 トラフィックの継続的な増加への対策

本システムでは，トラフィックの増加には基本的にサーバや機器のスケールアウトで対応し，スケールアウトが難しいサーバは個別に対策を施した．また，物理的なスイッチ構成への配慮や，システム全体のツイート流量を減らす取り組みも実施した．以下に詳細を説明する．

3.2.1.1 サーバのスケールアウト構成

スケールアウト構成を採用したのは，フィルタリングサーバ，出力サーバ，分配サーバ，配信サーバ，負荷分散装置である．これらは台数を追加することでほぼ線形にスループットが向上する．Firehoseからのツイート流量の増加にはフィルタリングサーバ，出力サーバ，分配サーバの追加で，ユーザ数の増加には配信サーバ，負荷分散装置の追加で対応できる．なお，これらの中でも分配サーバの構成には特に注意を払った. 分配サーバでは，配信キューから取り出したツイートデータを，下流のストリーム系配信サーバに一斉送信する．そのため，分配サーバが送信するデータ量は，Firehoseからのツイート流量の増加だけでなく，ストリーム系配信サーバの追加によっても増加する．したがって，分配サーバのネットワーク出力がボトルネックになる可能性がある．

☆1 正確には，Firehoseにはcountというクエリパラメータが存在し[22]，これを指定することで一定件数までは過去に遡ってツイートデータを取得できる．本システムでは，短時間の切断はこの機能を使って欠損したデータを回復している．

そこで，ストリーム系配信サーバの追加時には分配サーバのネットワーク流量も確認し，必要ならば後者も追加する設計とした．また，分配サーバとストリーム系配信サーバの間の一斉送信には，RabbitMQよりも軽量で高速な通信ライブラリとしてZeroMQを採用した．一方で，ZeroMQはキューに残っているデータの数を外部から取得できないなど，管理機能ではRabbitMQに劣る面もある．本システムにおけるこの問題の回避策については3.2.2.1節で述べる．

3.2.1.2　受信サーバの処理軽減

受信サーバは，日本語ツイートのみにフィルタリングする前のFirehoseの全データを受信するため，他のサーバよりも大量のデータを処理する．同一のTCP接続から送られてくるデータを複数のサーバに分けて受信することは難しいので，スケールアウトも困難である．

そこで，受信サーバはFirehoseからのデータ受信とフィルタリングサーバへの送信のみに専念し，日本語ツイートのフィルタリングなどの処理はフィルタリングサーバに任せる設計とした．また，ネットワークインタフェースを増設し，受信と送信を別々のインタフェースで行うようにした．

3.2.1.3　フィルターホース配信時のマッチ処理の高速化

フィルターホース配信サーバは，分配サーバから随時送信されてくるデータのうち，ユーザがHTTPリクエストのクエリパラメータで指定した検索条件に合致するものを，下位のTCP接続が切断されるまで，HTTPレスポンスでユーザに送信し続ける．1リクエストには多い場合で数千の検索キーワードが指定されるため，マッチ処理がボトルネックになる可能性があった．実際に，各キーワードを線形探索する実装で性能を検証したところ，検索キーワード数が多くなると，処理が追いつかなくなった．

対策としては，マッチ処理そのものの高速化と，マッチ処理の並列・分散化の2つを考えたが，後者は実現方式が複雑になるため，まずは前者の対策を実施した．

マッチ処理の高速化にはブルームフィルタ[23]を用い，以下のように実装した．

①HTTPリクエスト受信時に，クエリパラメータで指定された各キーワードの先頭2文字を，あらかじめブルームフィルタに登録する．

②分配サーバからツイートデータを受信したら，ツイート本文をバイグラムに分割し，分割結果のいずれかがブルームフィルタに合致するかどうかを判定する．

（ア）合致しない場合，そのツイートにはキーワードが含まれないことが保証されるため，ユーザには配信しない．

（イ）合致した場合，3文字目以降が合致しない場合があること，またブルームフィルタに偽陽性があることから，全キーワードを厳密に線形探索し，検索条件に合致すればユーザにツイートを配信する．

実際にFirehoseから取得したツイートデータと，そこから抽出したキーワード（本文を形態素解析し，名詞と判定された語から無作為に抽出したもの）を用いて検証したところ，本システムで採用したサーバよりも低スペックなサーバで，1,000キーワード指定時に20,000TPSを超える処理性能が得られた．これは実際の日本語ツイート流量に対して十分な性能であったため，並列・分散化の対策は不要と判断した．

3.2.1.4　ツイートデータの流れを妨げないスイッチ構成

物理ネットワークに対しては，各スイッチのバックプレーン容量は十分だが，スイッチ同士の接続部分（リンクアグリゲーションで8ポートを集約している）がボトルネックになることが予見された．そこで，データフロー上隣接するサーバをなるべく同じスイッチに接続し，できる限り同一スイッチ内で通信が完結するネットワーク構成とした．特に，ヒストリカルサーチで利用するMapReduceジョブによりHadoopクラスタ内で大量の通信が発生する蓄積基盤は，受信から蓄積・配信に至るデータの流れを妨げないよう全台を専用のスイッチ群に収容した．

3.2.1.5　削除メッセージ量の絞り込み

Firehoseからは，通常ツイートに加えてツイートが削除されたことを示すメッセージ（以下，削除メッセージと呼ぶ）も送信されてくる．このメッセージには，削除対象のツイートIDと投稿者のアカウントIDのみが含まれる．削除メッセージを受信した際は，指定されたツイートを本システムから削除し，またユーザにもその旨を通知する必要があるため，通常のツイートデータと同様に削除メッセージを蓄積部・配信部に送信する．削除メッセージもすべての言語のツイートに対応するものを受信するが，本システムで蓄積・配信するデータは日本語ツイートのみであり，日本語ツイートに対応する削除メッセージだけを送信したい．しかし，削除メッセージにはツイート本文やアカウントの言語設定が存在しない[24]ため，その削除メッセージが示すツイートが，日本語ツイートかどうかを判別できない．

そこで，フィルタリングサーバにおいて，削除メッセージ中のアカウントIDが日本語アカウントDBに登録されていた場合のみ，削除メッセージを蓄積・配信部に送るようにした．これにより，システムの上流で削除メッセージの量を絞り込んだ．

3.2.2 ツイート数の瞬間的な急増への対応

本システムでは，ツイート数の瞬間的な急増に対して，キューによる吸収とリングバッファによるタイムアウトの自動調整で対応した．以下に詳細を説明する．

3.2.2.1 キューによる吸収

本システムでは，ツイート数の瞬間的な急増は，サーバ間に置かれたキューで吸収する．これらのキューのサイズは，開発当時の最大TPSが，少なくとも5分間連続で送られてきても耐えられるだけのサイズを確保した．実際に，2013年10月時点で過去最高の143,199TPSを記録した際にも，データ溢れは発生していない．

さらに，万が一キューで吸収しきれない量のデータが送信された場合に，一部のボトルネックに引きずられてシステム全体のデータの流れが滞り，結果的にシステム全体がダウンしたり，Twitter側からの切断が発生したりすることのないよう，フェールソフトな設計にしている．以下に詳細を説明する．

まず，フィルタリングサーバの配信キューにはRabbitMQのx-message-ttl[25]オプションを設定し，一定時間を超えてキューに滞留しているデータは自動的に破棄されるよう設定している．一方，入力キュー・蓄積キューにはx-message-ttlを指定していない．これは，フィルタリングサーバが過負荷に陥った場合に，配信キューよりも入力キュー・蓄積キューを優先して守るためである．ツイートデータが蓄積基盤に格納されれば，後からそのデータをユーザに提供することも可能であるが，配信を優先して蓄積に失敗した場合，そのデータは失われてしまう．

また，データが破棄される兆候を事前に検知するため，キュー長が一定の閾値を超えた場合，アラートを上げるようにしている．この閾値は，サーバの増設などの対策を取るための時間を確保するため，通常の配信量であれば，キューから数秒間データが取り出されなければ直ちに到達する程度の低めの値に設定している．

次に，分配サーバと配信サーバの間の一斉配信に使っているZeroMQにはHWM[26]を設定することで，一定件数を超えてキューに投入されようとしたデータが自動的に破棄されるようにしている．こちらもデータが破棄されたことを検知するための監視を行っている．RabbitMQとは異なり，キュー内のデータ数を取得することはできないため，分配サーバの出力件数と配信サーバの入力件数を一定間隔で集計し，両者の比較を行っている．

なお，破棄されたデータは，必要に応じて蓄積基盤から抽出し，提供できるようにしている．

このように，ミドルウェアの持つ機能を活用し，システム全体としてのフェールソフトを実現している．

3.2.2.2 リングバッファによるDBアクセスのタイムアウトの自動調整

すでに説明したとおり，フィルタリングサーバでは，日本語アカウントDBへの登録や削除メッセージの絞り込みを行っている．ツイート数の瞬間的な急増により日本語アカウントDBが過負荷に陥ると，データの流れが滞る可能性がある．そこで，ツイートの流量に応じて，DBアクセスのタイムアウト時間を自動的に調節する仕組みをリングバッファを用いて実装した．

リングバッファは，先頭と末尾が連結された，循環して利用される固定長のバッファで，投入されたデータは，一定件数のデータが追加で投入されると押し出される．押し出されるまでの時間は，単位時間あたりの投入件数が多ければ短く，少なければ長くなる．

フィルタリングサーバは，入力キューから取り出したデータをまずリングバッファに投入し，データがリングバッファにとどまっている間に，日本語アカウントDBへのアクセスを行う．DBアクセスが終わる前にリングバッファから押し出されたデータは，流れを止めないよう，そのまま蓄積キュー・配信キューに投入する．このような方法で，ツイート量の急増時に，DBアクセス待ちでデータが滞留するのを防いでいる．

3.2.3 データの再取得の難しさへの対策

メンテナンスや単一障害で処理が止まることのないよう，本システムからはSPOFを排除している．そのために用いた方式の一覧を**表3**に示す．これらの方式のうち，ZooKeeperによる現用－待機構成について以下に補足する．

受信サーバでは数秒以内の高速な切り替えを実現するために，ZooKeeperを利用した．まず，先行して起動したサーバが現用系として，ZooKeeperとの接続が維持されている間のみ存在するエフェメラルノードをZooKeeper上に作成する．もう片方のサーバは，起動後，すでにZooKeeper上にノードが存在することを知り，そのノードをウォッチする．現用系がダウンしてエフェメラルノードが消えると，ウォッチしていた待機系がダウンを検知し，今度は自分が現用系としてエフェメラルノー

ドを作成し，動作を開始する．元の現用系のダウンが一時的なものであった場合も，すでにZooKeeper上にエフェメラルノードが存在するため，自主的に待機系に降格する．

また，ZooKeeperのクラスタが一貫性を保ちながら動作し続けるには，メンバの過半数が生存している必要がある．そのためZooKeeperクラスタを3台のサーバで構成し，1台がダウンしても，残りの2台が過半数を維持して運用を継続できるようにしている．

4. 蓄積基盤の開発における課題と対策

前章では，システム全体を通じて最も重要と考えた3つの課題を挙げ，それらに対して本システムが採用した対策について説明した．本章では，蓄積基盤に焦点を当て，Hadoopクラスタの設計やMapReduceジョブの開発において発生した課題と対策について説明する．

4.1 蓄積基盤の開発における課題

4.1.1 適切な圧縮方式とファイルフォーマットの選択

本システムでは，Twitterがサービスを開始した2006年以降のすべての日本語ツイートデータをHDFS上に格納している☆2．そのデータ量は膨大であり，効率的に格納するにはデータを圧縮する必要がある．また，ヒストリカルサーチを含む一般的なMapReduceではディスクI/Oがボトルネックになることが多いため，圧縮によってI/O効率を改善することで，処理時間も短縮できる．

しかし，不適切な圧縮形式を選ぶと，データの展開に時間を要し，かえってMapReduceが遅くなる場合がある．また，スプリット可能でない形式でデータを圧縮する場合，ファイルフォーマットとの組合せによっては，ブロック単独ではデータを展開できなくなってしまう．その結果，ブロック単位で複数のノードに分散して保持している圧縮ファイルを，1台のノードに集約，展開してから，そのノード上ですべてのMapReduceタスクを実行することになり，並列分散処理のメリットを活かせない．これらの問題を避けるため，適切な圧縮形式とファイルフォーマットの組合せを選択する必要があった．

4.1.2 JSONのパース・構築のオーバヘッド

Firehoseからはツイートデータが JSON形式で配信されてくる．単純にJSON形式のままデータを格納すると，ヒストリカルサーチのMapReduceを実行するたびに，検索条件で指定されたフィールドが条件に合致するかを判断するため，格納されたJSONをパースする必要があり，オーバヘッドが大きい．

一方，データの格納時にJSONをパースし，分解した個々のフィールドの値を格納しておく方法も考えられる．この場合，検索時にJSONをパースする処理は不要になるが，本システムからユーザへのツイートデータ配信時に各フィールドの値からJSONを構築するコストがかかってしまう．

このようなJSONのパースや構築に要するコストを回避できるよう，データの格納形式を工夫する必要があ

☆2　表1に挙げた4種類のAPI以外に，ユーザの要望に応じて任意の条件でツイートを抽出する任意データ抽出というメニューがあり，このメニューで利用するために2006年以降の全日本語ツイートをHDFS上に格納している．

表3　本システムで採用したSPOF排除方式の一覧

サブシステム	構成要素	方式
受信部	受信サーバ	ZooKeeperによる現用－待機構成
	フィルタリングサーバ	スケールアウト可能な複数台構成
	日本語アカウントDB	PG-REXによる現用－待機構成
蓄積部	出力サーバ	スケールアウト可能な複数台構成
	蓄積基盤(Hadoopクラスタ)	NameNode: Quorum Journal ManagerベースのNameNode HA JobTracker: Pacemaker + Heartbeatによる現用－待機構成 レプリカ数3で運用することにより，2台までの同時故障ではデータ欠損なし
	バッチサーバ	Pacemaker + Heartbeatによる現用－待機構成
配信部	分配サーバ	スケールアウト可能な複数台構成
	配信サーバ	スケールアウト可能な複数台構成
	負荷分散装置	スケールアウト可能な複数台構成，VRRPによる冗長化
	顧客DB	PG-REXによる現用－待機構成
共通	ZooKeeperサーバ	3台でアンサンブルを構成
	ネットワークインフラ	LANスイッチを2系統用意し，各サーバは両方の系統に接続して，それらのネットワークインタフェースをボンディング 系統間はリンクアグリゲーションで接続し高速化・冗長化 ルータ，ファイアウォール等もすべて二重化し，VRRP等で現用－待機構成
	ツイートデータ収集	主系と受信蓄積系の2系統でツイートデータを収集

った．

4.1.3　検索範囲が長期間にわたるクエリの複数同時投入

ヒストリカルサーチでは，直近13カ月分の全日本語ツイートから条件に合致するデータを検索できる．検索範囲が1カ月程度であれば，平常時はMapReduceが開始してから数分～十数分で検索が終わる（検索条件やその他のジョブによっても所要時間は異なる）．しかし，検索期間が数カ月以上のクエリは完了するまで数十分～数時間を要し，その間に大量のリソースを占有する．サービス仕様上，このようなクエリが複数同時に投入されるため，個々のクエリを独立したMapReduceとして順次処理する方式では処理が追いつかず，未処理のクエリが溜まっていく危険性があった．

4.2　課題に対する対策

前節で挙げた課題のそれぞれについて，本システムで採用した対策を以下に説明する．

4.2.1　SnappyとSequenceFileの組合せによる圧縮とスプリットの両立

4.1.1節で挙げた課題に対処する方法の1つは，スプリット可能な圧縮形式であるbzip2やLZOを使うことである．しかし，前者は他の形式に比べると展開処理が低速であり，性能への影響が懸念された．また，後者はライセンスの関係上Hadoopに同梱されないため導入に追加作業を要し，またデータ格納時にインデックスを作成するための前処理が必要になる．

そこで，本システムでは，追加作業なしで導入でき，処理速度と圧縮効率のバランスに優れた[27]Snappy[28]を圧縮形式に採用した．Snappy自体はスプリット可能な圧縮形式ではないが，SequenceFileと組み合わせることでこの制限を克服できる．SequenceFileは任意のキーと値の組合せを1レコードとするバイナリ形式のファイルフォーマットであり，ブロック単位での圧縮をサポートしている．そのため，ファイル中の境界情報は残したまま，ブロックの内容だけを圧縮することで，複数のノードでMapReduceタスクを分散実行できる．そこで，本システムではSequenceFileに格納したツイートデータをSnappyによってブロック単位で圧縮する方式を採用した．

4.2.2　SequenceFileのキーと値への格納データの工夫

4.1.2節で挙げた課題に対処するため，SequenceFileのキーと値の設計を工夫した．

まず，キーにはJSONをパースして得られたツイート本文やアカウント名，ツイート日時など，ヒストリカルサーチで検索対象となるフィールドを，あらかじめ定めたセパレータで連結し格納した．MapReduceジョブでは，合致するレコードを探すにはキーのみを参照すればよく，ツイートデータのJSONをパースする必要がなくなった．

そして，値にはツイートデータをJSON形式のまま格納した．これにより，MapReduceの抽出結果をユーザに配信するときには，値をそのまま送信すればよく，その都度データをJSON形式に再構築する必要がなくなった．

4.2.3　複数クエリのまとめ込み

4.1.3節で挙げた課題への対応として，一定の時間内に投入された複数のクエリを1つのMapReduceジョブとしてまとめ込む方式を採用した．処理フローを**図2**に示す．

複数のクエリが同時に投入された場合，投入されたクエリの数が多いほど，また個々のクエリの検索期間が広いほど，それらのクエリが対象とする期間が重なる可能性が高くなる．そこで，それらのクエリの検索期間のORを取った期間（連続していない場合もある）を処理対象とする1つのMapReduceジョブにまとめることで，重複したデータの読み込みやジョブの起動・停止のオーバヘッドを排除した．

MapReduceの結果はカスタムPartitioner[29]とComparatorによって，個々のクエリごとに連続したファイルにソート済みの形で出力する．ヒストリカルサーチ配信サーバはクエリに対応する結果ファイルのみを読み込み，ユーザに結果を返却する．

また，検索期間に応じてMapReduceジョブの実行頻度と優先度に差をつけ，検索期間が短いクエリをまとめ込んだジョブは高い優先度と短い間隔で，長いクエリをまとめ込んだジョブは低い優先度と長い間隔で実行されるようにした．このようにして，検索期間が長いクエリが同時に複数投入された場合でも，検索期間が短いクエリが所定の時間内に完了するようにした．

5．おわりに

本稿では，Twitterデータ提供サービスの概要とシステム構成，OSSの利用方法を紹介した．また，今回の開発における課題に本システムがどのように対処しているかを説明した．

第1章でも述べたように，本システムでは高品質なサービスを実現するため，全面的にOSSを採用した．その結果，以下のようなメリットを享受し，使用性（理解性・

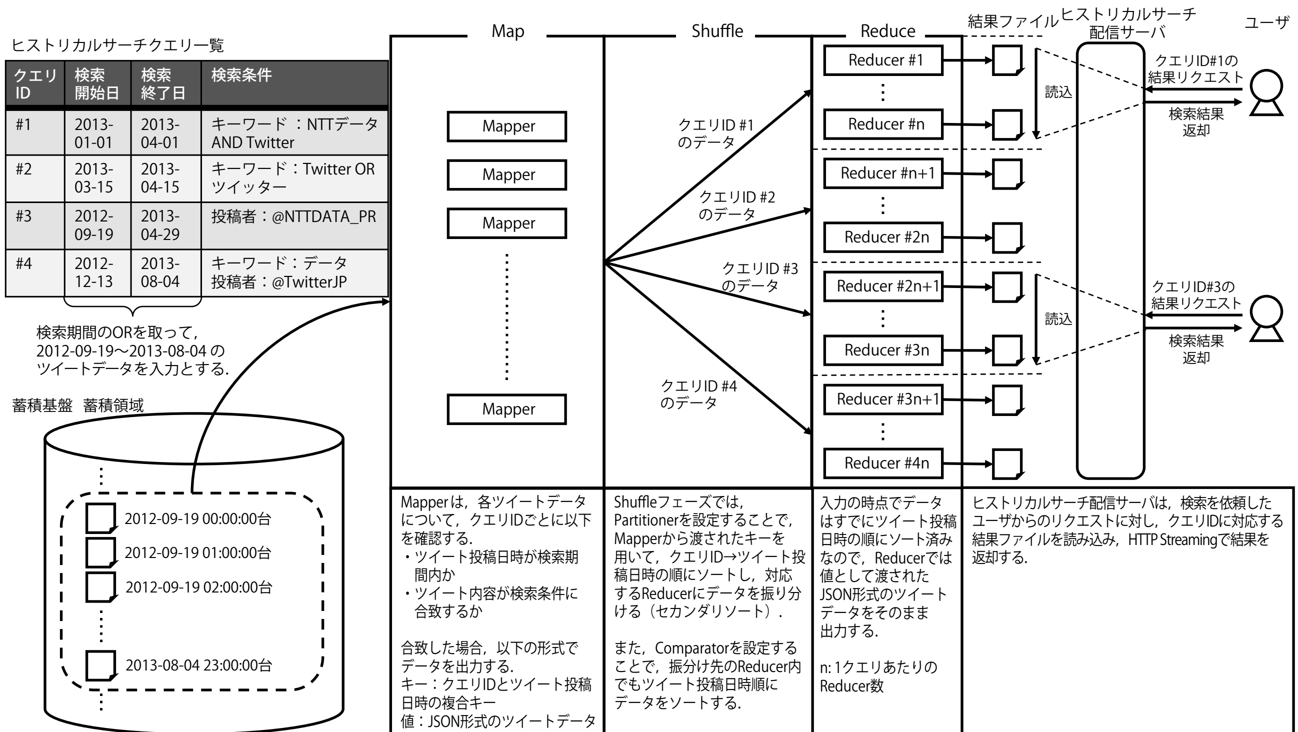

図 2　複数クエリのまとめ込みによる，ヒストリカルサーチの MapReduce ジョブ効率化

習得性・運用性）と保守性（解析性・変更性・試験性）[30]に優れたシステムを構築することができた．

- ソースコードがすべて公開されているため，自分たちで調査・修正できる範囲が広がる．問題解決までの時間が短縮でき，運用対処ではなく根本原因の対処までたどり着くことができる．
- 不具合や修正情報などが課題管理システムなどで公開されており，問題が発生したときに類似の事象や回避策の有無を調べることができる．
- バグ報告やパッチ投稿などのコミュニティ活動を通じ, 企業の枠を超えた外部の力を活用できる．また，社内の技術者のスキルやモチベーションの向上に繋がる．

これらの具体例として，今回の開発で利用したOSSの1つであるZeroMQの不具合を取り上げたい．本システムの開発中に，大量のソケットをオープン／クローズするとZeroMQがクラッシュするという事象が発生した．コミュニティの課題管理システムを調査したところ，当時の現行バージョンに類似の事象が存在することが確認できた[31]．マルチスレッド環境下でのみ発生する事象のため，再現プログラムを作成し，coreダンプ結果やソースコードの解析結果とともにコミュニティに報告した．コミュニティからはワークアラウンドパッチの提案があり，そのパッチを適用することで事象は解消した．我々がこのパッチの有用性を報告し，最終的に次期バージョンにもパッチは取り込まれた[32]．

高品質なシステムを開発する上では，使用するミドルウェアやライブラリについても細部まで把握できていることが望ましい．OSSはそのための有効な手段であると言えよう．

謝辞　本システムの開発にあたり，NTT ソフトウェアイノベーションセンタ（当時）の北山禅氏（現NTTレゾナント（株））および石井方邦氏（現NTTコミュニケーションズ(株)）に多大なるご支援をいただきました．この場を借りてお礼を申し上げます．

参考文献

1） FORM S-1, REGISTRATION STATEMENT, Twitter, Inc., http://www.sec.gov/Archives/edgar/data/1418091/000119312513390321/d564001ds1.htm
2） （株）NTT データ : 米 Twitter 社とツイートデータ提供に関する Firehose 契約を締結 , http://www.nttdata.com/jp/ja/news/release/2012/092700.html
3） （株）NTT データ : Twitter データ提供サービスの開始について, http://www.nttdata.com/jp/ja/news/release/2012/111900.html
4） Twitter Developers, GET statuses/firehose, https://dev.twitter.com/docs/api/1.1/get/statuses/firehose
5） Apache ZooKeeper, http://zookeeper.apache.org/
6） RabbitMQ, http://www.rabbitmq.com/
7） PostgreSQL, http://www.postgresql.org/
8） Cluster Labs, http://clusterlabs.org/

9) Linux-HA, http://www.linux-ha.org/
10) PG-REX, http://sourceforge.jp/projects/pg-rex/
11) Apache Hadoop, http://hadoop.apache.org/
12) ZeroMQ, http://zeromq.org/
13) Hintjens, P.: ZeroMQ, O'reilly Media, pp. 11-16（2013）.
14) Apache HTTP Server, http://httpd.apache.org/
15) Apache Tomcat, http://tomcat.apache.org/
16) Sedue for BigData, http://preferred.jp/product/sedue-for-bigdata/
17) Twitter Blogs, Measuring Tweets, https://blog.twitter.com/2010/measuring-tweets
18) Twitter Blogs, Big Goals, Big Game, Big Records, https://blog.twitter.com/2010/big-goals-big-game-big-records
19) Twitter Blogs, #numbers, https://blog.twitter.com/2011/numbers
20) Twitter Developers, Connecting to a Streaming Endpoint, https://dev.twitter.com/docs/streaming-apis/connecting#Disconnections
21) https://twitter.com/TwitterJP/status/363494742518013952/
22) Twitter Developers, Streaming API request parameters, https://dev.twitter.com/docs/streaming-apis/parameters#count
23) Bloom, B. H.: Space/Time Trade-offs in Hash Coding with Allowable Errors, Commun. ACM 13 (7), pp.422-426 (1970).
24) Twitter Developers, Streaming Message Types, Status Deletion Notices, https://dev.twitter.com/docs/streaming-apis/messages#Status_deletion_notices_delete
25) RabbitMQ Documentation: Per-Message TTL, https://www.rabbitmq.com/ttl.html#per-message-ttl
26) Hintjens, P.: ZeroMQ, O'reilly Media, pp.77-78 (2013).
27) Holmes, A: Hadoop in Practice, Manning Publications, p.178 (2012).
28) Snappy - A Fast Compressor/Decompressor, http://code.google.com/p/snappy/
29) 中野，山下，猿田，上新，小林 : Hadoop Hacks プロフェッショナルが使う実践テクニック , オライリー・ジャパン，Hack #13「カスタム Partitioner の作り方」, pp.80-83 (2012).
30) (財) 日本規格協会 : JIS X 0129-1, ソフトウェア製品の品質―第 1 部：品質モデル (2012).
31) ØMQ issue tracker, [LIBZMQ-281] Crash on Heavy Socket Creation: Device or Resource Busy (mutex.hpp:91), https://zeromq.jira.com/i#browse/LIBZMQ-281
32) ØMQ Issue Tracker, [LIBZMQ-496] Crash on Heavy Socket Opening/Closing: Device or Resource Busy (mutex.hpp:90): https://zeromq.jira.com/browse/LIBZMQ-496

関 堅吾（非会員）sekikn@nttdata.co.jp
（株）NTT データ 第三法人事業本部 主任．2002 年京都大学大学院情報学研究科知能情報学専攻修士課程修了．修士(情報学)．同年（株）NTT データに入社．日本語解析エンジン「なずき」およびその関連製品・サービスの開発に従事．

金子 崇之（非会員）kanekotky@nttdata.co.jp
（株）NTT データ 基盤システム事業本部 課長．1999 年早稲田大学大学院理工学研究科情報工学専攻修士課程修了．同年（株）NTT データ入社．Hadoop 等の OSS を活用した開発案件に従事．

山下 真一（正会員）yamashitasni@nttdata.co.jp
（株）NTT データ 基盤システム事業本部 課長代理．2006 年岡山大学大学院自然科学研究科電子情報システム工学専攻修士課程修了．同年（株）NTT データ入社．Hadoop 等の OSS を活用した開発案件に従事．

採録決定：2014 年 1 月 21 日
編集担当：冨士　仁（NTT セキュアプラットフォーム研究所）