リアルタイムなソーシャルメディア分析システムのデータバースト時のストリーム処理制御と性能評価

榎 美紀†‡　吉田 一星†　小口 正人‡

†日本アイ・ビー・エム(株) 東京基礎研究所
‡お茶の水女子大学理学部情報科学科 〒112-8610 東京都文京区大塚 2-1-1
E-mail: †enomiki@jp.ibm.com

あらまし Twitter に代表されるマイクロブログサービスでは，リアルタイムにメッセージが多く発信されそれらが多くのユーザーに再共有されて情報が拡散すると，その拡散規模や内容によっては，現実社会に与えるインパクトも大きい．それゆえ，ソーシャルメディア上で今何が広く拡散しているのか，を知ることは，企業や団体にとって重要である．本論文では，情報拡散データをストリームデータとして処理してリアルタイムに分析するためのシステムのフレームワークを紹介する．ソーシャルメディアデータの特性のひとつに，データのバースト性がある．定期的に発信されるセンサーデータ等と異なり，時間や現実社会での出来事の影響を受けて，ストリームの量が変化することが多々ある．そのような場合，普段稼動しているシステムの処理能力が追いつかずに，性能劣化を引き起こす可能性がある．そこで本論文では，各ツイートのデータ分析における重要度を計算し，重要度が低いと判断されたツイートをフィルタリングして処理するデータ量をコントロールする手法を提案する．

キーワード ソーシャルメディア，情報拡散,インメモリデータベース, Twitter

Performance Evaluation of Real-time Query Processing of Information Diffusion on Social Media

Miki ENOKI†‡　Issei YOSHIDA†　and Masato OGUCHI‡

†IBM Research - Tokyo
‡Ochanomizu University　2-1-1 Otsuka, Bunkyou-ku, Tokyo, 112-8610, JAPAN
E-mail: †enomiki@jp.ibm.com

1. はじめに

Twitter のようなリアルタイム性の高いソーシャルメディアでは，現実社会で起きた災害やイベントについてユーザーが即座に情報を発信したり，逆に，インパクトのある情報がソーシャルメディア上で発生して現実社会に影響を及ぼす現象も多く発生する[1]．それゆえ，ソーシャルメディア上で今どんな情報が広く拡散しているのか，をリアルタイムに知ることは，企業や団体にとって炎上防止や流行把握のために重要である．これらは現状では，人手で人海戦術によるモニタリングを実施するか，あらかじめ登録したキーワードのバーストを発見して異常検知する商用サービスを利用することが近年の企業のソーシャルメディア活用の傾向である[2]．

ソーシャルメディアの情報をリアルタイムに分析してバーストやイベントを検知する研究は多く存在する．Twitter のメッセージ内容を解析して特徴的なキーワードを抽出し，その頻度のバースト性により，今何がトレンドとなっているかをモニタリングする[3,4]．位置情報やメッセージ内容を分析して，そのキーワードやツイート発信場所のバースト性により，今どんなイベントが発生しているかを発見する[5]．これらのサービスや研究は，各ツイッターのメッセージに出現する「キーワード」の増減の情報を基にした分析である．必ずしもツイート間には繋がりはなく，同じキーワードを話題にしているという状態を分析対象にしている．

対して，我々が分析対象とするのは，メッセージの再共有(リツイート，RT)で広がっていく情報拡散である．あるツイートが多数のユーザーに RT されて広く拡散したメッセージは,多くのユーザーが興味をもち,インパクトを与えた情報であるといえる．また，あるトピックに関する複数のツイートの拡散データを対象にして，それらのツイートをよく RT しているユーザー達や，逆によく RT されているユーザー等,「キーパ

ーソン」を見つけることにより,「どのようなユーザーが興味をもっているか」「誰が話題の中心になっているか」「どのようなユーザー間の流れを介して情報が拡散しているのか」という事を発見できることが期待される．発見したユーザーはユーザープロファイリングなどの分析を行うことにより[6]，人となりを深く分析可能になる．我々はこのような情報拡散データの分析を実現するためのシステム構築を目的としている．

ソーシャルメディア上では，あるトピックが瞬間的に大きく話題になると多くのユーザーが一斉にツイートを発信し，バースト的な状態を起こすことがある．例えば，大規模な震災が発生した瞬間やオリンピックなどのスポーツの試合で盛り上がった瞬間，各国の選挙投票日などがあげられる．そのような場合，システムは何千何十万のメッセージを同時に処理することになり，普段稼動しているサーバーのキャパシティの限界に達して処理が遅延したりデータを欠損してしまうような危険性が生じる．そこで我々は，各ツイートの重要度を計算し，重要度が低いと判断されたツイートをフィルタリングして処理するデータ量をコントロールする手法を提案する．実際にバーストしたツイートデータを用いて，本手法の有効性を確認する．

2. 情報拡散分析システム

2.1. 情報拡散データ

ある一つのツイートに対して，それを RT しているリツイートデータを関連付けて蓄積していくと，1 つの情報拡散データとなる．RT をエッジとし、RT したユーザーとされたユーザーをノードとするグラフ構造を拡散ネットワークと呼ぶ．拡散ネットワークを可視化すると，拡散の規模や拡散経路が視覚的に捉えられて直感的に理解しやすくなる．

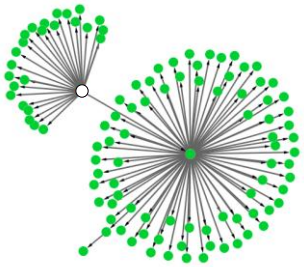

図 1　拡散ネットワーク

Figure 1　Information diffusion network

図 1 はオリジナルのツイートを発信したユーザー(中心が白色のノード)と，そのツイートを RT したユーザー(色の付いたノード)をネットワークのノードとし，エッジは情報の流れを表している．例えば，ユーザー@b がユーザー@a のツイートを RT した時，@a のノードから@b のノードへ向かうエッジがはられる．つまり，1 エッジが 1 リツイートに該当する．

2.2. 情報拡散分析システム概要

本研究のシステム構成を図 2 に示す．Twitter から発信されるツイートをリアルタイムに取得し，インメモリのデータストアに一時的に格納する．Twitter から日々発信されるツイートは膨大な量であるため，拡散データの収集対象とするツイートをフィルタリングしても良い．例えば，特定のユーザーが発信するツイートの拡散や特定のキーワードが含まれるツイートの RT のみを格納するように指定する．

システムのデータストアには，リレーショナルデータベース，グラフデータベースやキー/バリューストアが候補としてあげられる[7,8]．分析ユーザーが人気のリツイートやユーザーを発見するためには集約やソートの処理が必要であり，キー/バリューストアよりも，SQL を用いて複雑なクエリが実行できるリレーショナルデータベースのほうが分析の幅が広がると考えるため，本研究ではインメモリデータベースを採用する．

一方で，データストアに格納した場合，サーバーのメモリサイズには限りがあるため，延々と蓄積し続けることは現実的ではない．また，すっかり RT されなくなった鮮度の低い拡散データなどが残り続けて分析対象になってしまうことが懸念される．そこで，各ツイートの RT の拡散が収束する時点を推定して，今も RT され続けているアクティブな拡散データはデータストアに出来るだけ格納し続け，拡散が収束したデータはデータストアから退避するようなメンテナンス処理を実施する[9]．データストアから退避されることになった拡散データは，HDD のデータベースに格納して，Historical データを対象としたオフラインの分析に利用するか，そのまま破棄する．

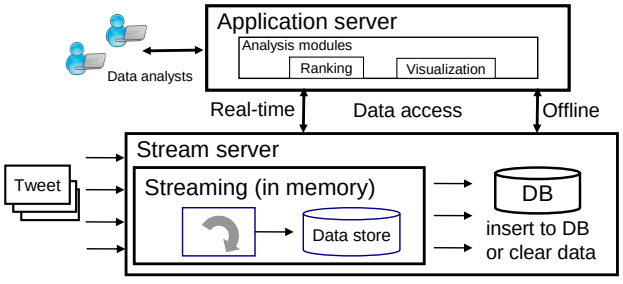

図 2　情報拡散分析システム

Figure 2　Infomation diffusion analysis system

アプリケーションサーバーには，拡散ネットワークを分析するための複数のモジュールが入り，分析ユーザーはインタラクティブに分析を実施する．例えば，分析ユーザーは特定のツイートの集合に対して，多く

RT されている人気のユーザーやツイートを発見する．定期的にクエリを発行してモニタリング目的に使用したり，分析ユーザーが指定したツイートの拡散ネットワークを生成し，可視化することも可能である．

3. 代表的な問合せパターン

3.1. インメモリデータベース

データベースにはリツイートのデータを格納していく．インメモリのデータベースの実装はいくつか存在し，オープンソースの H2, Apache Derby や，商用の TimesTen, solidDB などがある[10].

データベースには，RETWEET テーブルと，ORIGIN_TWEET テーブルを生成する (図 3)．RETWEET テーブルには，RT のツイートの ID(TweetID)，オリジナルツイートのツイート ID(RTID)，RT を発信したユーザー名(Dst)，RT 元であるオリジナルツイートを発信したユーザー名(Src)，RT 発信時刻(Time)，使用言語(Lang)，位置情報(Location)等を属性として持たせる．1 レコードが 2.1 節の 1 エッジに該当する．

RETWEET table

TweetID	RTID	Time	Src	Dst	Lang	Location	‥
100	1	Time data	u1	u2	Ja	GPS	‥
101	1	Time data	u2	u4	Ja	GPS	‥
‥	‥	‥	‥	‥	‥	‥	‥

ORIGIN_TWEET table

TweetID	Time	User	Msg	RTcount	‥
1	Time data	u1	message	29	‥
2	Time data	u5	message	14	‥
‥	‥	‥	‥	‥	‥

図 3　拡散ネットワークデータベース

Figure 3　Diffusion network database

ORIGIN_TWEET には，ツイート ID (TweetID)，ツイート発信時刻(Time)，発信ユーザー名(User)，ツイートのメッセージ(Msg)，RT された回数(RTcount)がある．RTcount は，対応する RT を受信するたびにカウントされる．ある RT の，RT 元となるオリジナルツイートの情報を取得したい場合は，RETWEET テーブルの RTID と，ORIGIN_TWEET テーブルの TweetID を Join 結合して問合せを行う．

3.2. 拡散分析のための問合せパターン

本システムにて拡散データを対象にした代表的な問合せパターンを以下に紹介する．

(Pattern1) 指定した拡散ネットワークを取得する選択演算

拡散ネットワークを生成するため，所望の拡散データを問合せる．これにより，「ユーザー間をどのような経路で RT が拡散していったのか？」といったような情報を得ることができる．また，生成した拡散ネットワークは，クラスタリングや頻出経路発見などのネットワーク分析に応用することも可能となる．この問合せパターンに対応する SQL は以下のような選択演算となる：

```
[Query 1]
SELECT      Src, Dst
FROM        RETWEET
WHERE       RTID in (tweet ids)
```

インプットはツイート ID であり，この ID は分析モジュールにて指定されるか，もしくはデータベースから取得される．例えば，データベース内の ORIGIN_TWEET テーブルに対して，あるトピックを表す特定のキーワードが含まれるツイートのツイート ID リストを取得し，そのリストを Query 1 のインプットにすることにより，そのトピックに関連する拡散ネットワークを取得する．

(Pattern2) 指定した拡散ネットワークの集約演算

拡散ネットワークの何らかの属性値に着目した集約演算である．例えば，RT 数の多い順のツイートランキングを問合せる．この問合せパターンに対応する SQL は以下のようになる：

```
[Query2]
SELECT      *
FROM        ORIGIN_TWEET
WHERE       TweetID in (tweet ids)
ORDER BY    RTcount   DESC
 FETCH FIRST 100 ROWS ONLY
```

WHERE 句にてツイート ID を指定しない場合は，単純に現在格納している拡散データの中で，最も RT 数の多い上位 100 件が出力される．

また，RT されたユーザーに着目して集約した場合，RT された，数の多い順のユーザーランキングを取得できる．この問合せパターンに対応する SQL は以下のようになる：

```
[Query3]
SELECT      Src, count(Src)
FROM        RETWEET
WHERE       RTID in (tweet ids)
GROUP BY    Src
ORDER BY    count(Src)   DESC
FETCH FIRST 100 ROWS ONLY
```

Query 3 にて Src を指定した場合は，指定したツイートの集合の中で，最も RT された総数の多い順のユーザー(＝拡散影響力の高いユーザー)が出力される．一

方，Dst を指定した場合は，指定したツイートの集合の中で，最も RT していた総数の多いユーザー(=情報を RT しやすいユーザー)が出力される．

他に，各ツイートを発信したユーザーの居住地のランキングや，ツイート文中の語句の頻出ランキングなどの問合せが考えられる．どのような問合せが必要かは，トピックや分析ユーザーの分析シナリオに依存する．

以上のような問合せは，インメモリデータベースで処理されるためディスクベースのデータベースよりもデータ処理が高速になることが期待される．しかしながら，予めインデックスを張ったカラムに対するシンプルな問合せは高速に処理可能であるが，ソートや Join，副問合せ等メモリ上でのデータ演算が問合せコストの多くを占めるような複雑な SQL になるほど，処理能力が HDD のデータベースと同程度まで下がってしまう可能性がある [11].

本システムの場合，Pattern2 の問合せ処理は，ユーザー単位での集約，ソートの処理等が必要になるため，やや複雑な SQL となる．

4. データバースト時のキャパシティコントロール手法の提案

4.1. データバースト時に起こる性能劣化

ソーシャルメディア上では，あるトピックが瞬間的に大きく話題になると多くのユーザーが一斉にツイートを発信し，バースト的な状態を起こすことがある．たとえば，オリンピックなどのスポーツの試合で盛り上がった瞬間や，大規模な震災が発生した瞬間等である．そのような場合，システムは何千何万のツイートを処理することになり，サーバーのキャパシティの限界に達して処理しきれなくなる可能性が生じる．そこで本章では，各ツイートの重要度を計算し，重要度が低いと判断されたツイートをフィルタリングして処理するデータ量をコントロールする手法を提案する．

図 4 は日本の衆議院議員選挙の開票日(2014 年 12 月 14 日)から翌日にかけての各時間帯の，政党名を含んだツイート（含リツイート）の発信数を示している．これにより，開票開始時間である 14 日の 20 時に大きくツイート数が跳ね上がっていることが分かる．このように瞬間的なバーストが発生した時，システムを稼動しているサーバーのリソース不足等を引き起こしてしまう可能性がある．

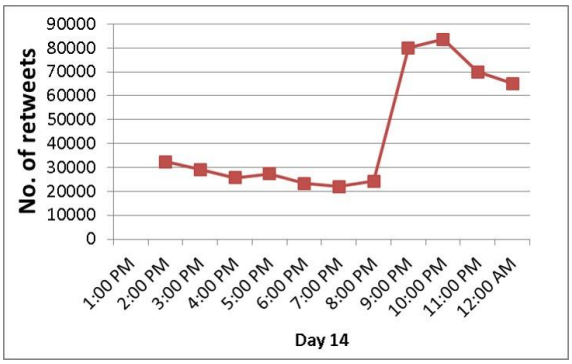

図 4　ツイートのバースト
Figure 4　Bursting of tweets

図 5 は各時間帯においての，Query 2, 3 の平均応答時間とストリームサーバーの CPU 使用率をを示している．開票時間後に徐々に応答時間は長くなり，22 時台には約 3 秒もかかっている　（詳細な実験設定については次章に述べる）．CPU 使用率も 90%にまで到達し，リソースが限界近くまで達していることがわかる．本システムは，分析ユーザーからインタラクティブに問合せ処理が発行されることを想定している．したがって，問合せ応答時間が数秒を要するということは性能上で問題である．

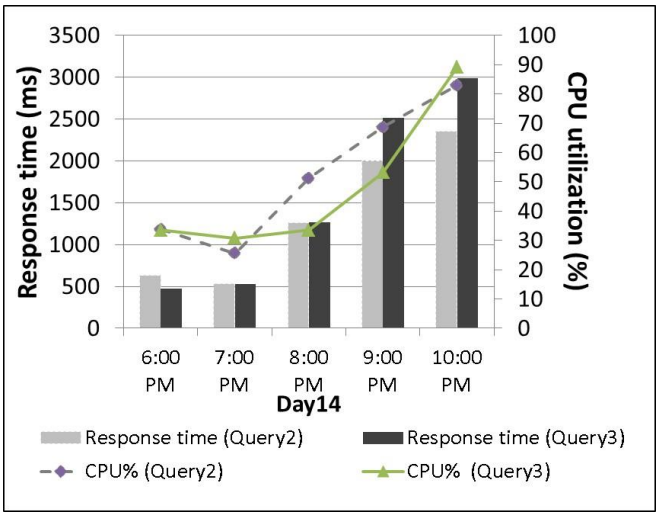

図 5　各クエリの問合せ応答時間
Figure 5　Response time of each query

4.2. データバースト時のキャパシティコントロール手法の提案

解決策として，システムを分散することが考えられるが，データが分散することになり，分散問合せ処理を実行しなくてはならなず，問合せ実行時間に影響を及ぼす可能性があり，トレードオフとなる．本研究では，他の手段として，分析においてあまり重要ではな

いデータをフィルタリングすることにより，性能劣化を回避する手法を提案する．

単純な解決策としては，送信されてくるツイートをランダムにフィルタリングして，処理可能な量のみを扱うようにするという方法が考えられる．しかしながらこの場合，各ツイートのリツイートがランダムフィルタリングされ，図 1 で示したような拡散ネットワークの経路が分断されてしまう．本システムが分析対象とするのは，拡散データであるため，このようにデータが欠損してしまうことは望ましくない．

ツイートには，発信したユーザーの ID，フォロー/フォロワーのユーザー数など，様々な情報が含まれている．そこで，ツイートに付随しているそれらの情報を用いて，ツイートに重要度を表す重み値を付与する．拡散データを分析するにあたり，拡散数が大きいツイートほどインパクトの大きいツイートであり，分析対象とする価値が高いと考えられるため，そのようなツイートの重要度が高くなるようにしたい．したがって，拡散数が大きくなりそうなツイートとそのリツイートのデータはできるだけフィルタリングせずに残すようにする．

以下にツイートの重要度を計算する式を表す．

$$Weight(t) = \begin{cases} getUserInfo(t_{RTID}), & if\ \ t\ \ is\ \ retweet \\ getUserInfo(t_{ID}), & otherwise \end{cases}$$

*Weight(t)*はツイート，もしくはリツイート *t* の重要度を示す重み値である．*getUserInfo* メソッドは，*t* に付随する情報をインプットとして，重み値を返す．本論文では，ユーザーのフォロワーの数を返す．これは，フォロワー数が多いユーザーのツイートは RT される頻度が高いという仮定のもとに設定した．送信されてきたツイートがツイートの場合はそれを発信したユーザーのフォロワー数を，リツイートの場合は，そのオリジナルツイートを発信したユーザーのフォロワー数を参照している．この重み値がある閾値を下回った場合，フィルタリングされる．

この閾値をあげて，多くのツイートをフィルタリングする程，分析結果の精度をさげてしまう可能性が大きくなる．これは，キャパシティのコントロールと分析結果の精度のトレードオフとなる．

5. 実験

Twitter のデータを用いて，バースト時のキャパシティコントロールの手法の効果を評価する．

5.1. 実験シナリオ

4 章で提案した，データバースト時のコントロール手法についての評価を行う．以下の 2 手法にて実験結果を比較する．

(手法 a)　Random Filtering

入力ツイートをある特定のレートでフィルタリングを行うナイーブな手法である．本実験では，手法 b の閾値 2000 の時のフィルタリング結果と同程度の削減率になるように，フィルタリングレートを 40%に設定する．これにより，手法 b の閾値 2000 の時の実験結果と比較可能な状態となる．

(手法 b) Weight filtering (提案手法)

重み値としてユーザーのフォロワー数を用いる．閾値は 1000 または 2000 に設定する．すなわち，ツイートの発信ユーザー（RT の場合はオリジナルツイートの発信ユーザー）のフォロワー数が閾値よりも小さい場合，そのツイートはフィルタリングされる．

実験データは，2014 年 12 月 の衆議院議員選挙の投票日に，政党名をメッセージに含んでいた日本語ツイートを用いる．該当日のツイート数は図 4 に示す通りである．実験に用いるマシンは 2 x CPU Xeon X5670 (2.93GHz, 6 cores) with RAM 32 GB, OS は Red Hat Linux 5.5 を使用した.システムは Java (IBM J9 VM JRE 1.7.0) で実装した． データベースは，H2 v1.4.184 をインメモリモードで使用し，RETWEET テーブルと ORIGIN_TWEET テーブルの，オリジナルツイートの ID (TweetID, RTID)と，RETWEET テーブルの，オリジナルツイートを発信したユーザー(Src)にインデックスを張っている．

5.2. バースト時の問合せ性能評価

RETWEET テーブルと ORIGIN_TWEET テーブルのレコード数を表 1 に示す．これらのテーブルには，現在 RT され続けているアクティブなツイートのデータが格納されている[9]．リツイートされなくなった古いデータはインメモリデータベースからは削除される．どちらのテーブルも 22 時台のレコード数が最も多い．これは投票開票の 20 時以降にツイート数が急増し，それらのリツイートが依然活発にリツイートされており，蓄積されていった結果である．手法 b の閾値 1000 と 2000 では，RETWEET テーブルの約 70%と 60%までそれぞれ総レコード数を削減している．手法 a は，約 60%まで削減している．

一方，ORIGIN_TWEET テーブルでは，手法 b の閾値 1000 と 2000 において，それぞれ約 50%と 30%まで総レコード数を削減している．手法 a は約 75%の削減となっており，手法 b よりも削減数が少ない．これは，手法 b は，リツイートのオリジナルツイートを発信したユーザーの重み値が閾値以下のリツイートはすべてフィルタリングされるため，オリジナルツイートの総

数自体が減るためである．手法 a は，ランダムにリツイートをフィルタリングするため，オリジナルツイートの総数自体は手法 b ほど削減しなかったと考えられる．

表 1. 各テーブルの総レコード数

Table 1. Total number of records in each table

RETWEET	20:00	21:00	22:00
original	146,745	205,965	252,876
method (a)	88,049 (60%)	123,581 (60%)	151,727 (60%)
method (b) user_1000	110,368 (75%)	149,830 (73%)	181,752 (72%)
method (b) user_2000	91,763 (63%)	123,164 (60%)	149,043 (59%)

ORIGIN_TWEET	20:00	21:00	22:00
original	28,542	36,747	43,699
method (a)	21,032 (74%)	27,193 (74%)	32,380 (74%)
method (b) user_1000	14,530 (51%)	17,957 (49%)	20,928 (48%)
method (b) user_2000	9,949 (35%)	12,060 (33%)	13,927 (32%)

問合せ性能評価として，3.2 節で紹介した問合せパターンである，Query 2 と Query 3 の SQL を用いて応答時間を測定した．100 ユーザーが同時にこれらのクエリを発行してくると想定して測定する．WHERE 句で指定するツイートの ID は，我々の実験データの中で，“自民党”というキーワードが含まれるツイートの ID のリストをセットする．これは，選挙に関するリツイートのデータを対象にして，自民党に関するツイートの中で，どのツイートが最もリツイートされていたのか (Query 2), どのユーザーが最もリツイートされたか (Query 3) のランキング結果を取得することになる．

100 ユーザーは，20, 21, 22 時台のデータを対象に Query 2 と Query 3 の SQL をそれぞれ 10 回発行し，平均応答時間を計算する．図 6 は各時間帯での Query 2 の応答時間の中央値を示している．全手法において，22 時台の応答時間が最も長い．これは，テーブルのレコード数が 22 時台が最も多いためであると考えられる．手法 b の閾値 1000 では，オリジナルデータの結果と比較して約 50%まで削減されているが，依然として 22 時台は 1 秒以上の応答時間になっている．閾値 2000 の場合，全時間帯において応答時間は 1 秒以下に削減できている．手法 a では，データの削減率は手法 b の閾値 2000 と同じだったものの，応答時間は長く，22 時台では約 2 秒の応答時間になってしまっている．

問合せ応答時間は，Query 2, Query 3 の WHERE 句で指定されるオリジナルツイートの ID の数に影響をうけると考えられる．表 4.2 で示したように，手法 a では，ORIGIN_TWEET テーブルの総レコード数は手法 b より削減できていないために，より長い応答時間の結果となったと思われる．

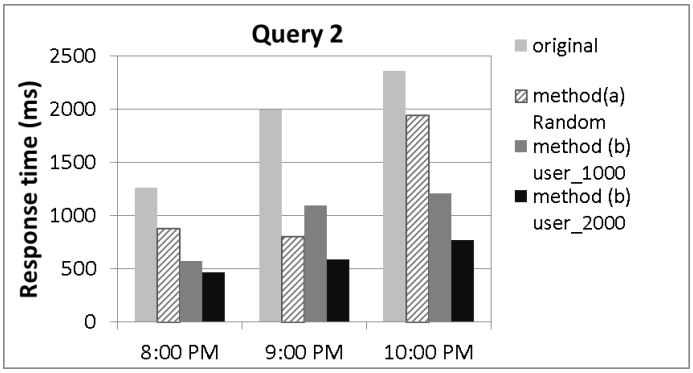

図 6　Query 2 の応答時間

Figure 6　Response time of Query 2

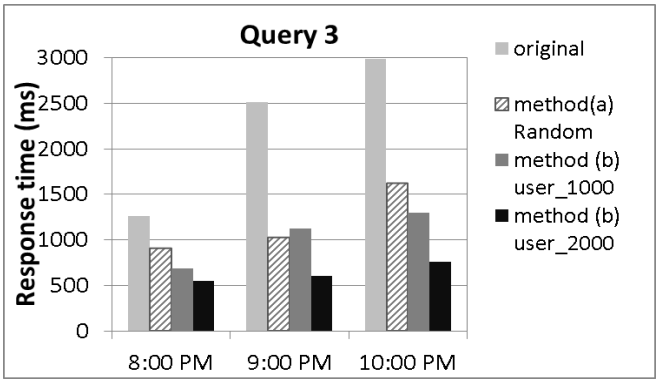

図 7　Query 3 の応答時間

Figure 7　Response time of Query 3

Query 3 の結果を，図 7 に示す．全体的な傾向は，Query 2 の結果と同じである．手法 b の閾値 2000 の結果だけが，全時間帯において応答時間を 1 秒以下に維持できていた．図 8 はそれぞれの Query を実行したときの平均 CPU 使用率を示している，オリジナルデータでの実行時は，最高で 80%以上の CPU を消費してい

る．フィルタリング適用後は，全手法において，50%前後の使用率にまで下げることができている．これにより，フィルタリングにより，データバースト時のリソース消費を抑えることができていることがわかる．

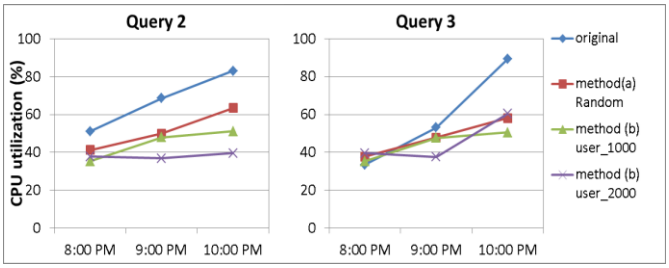

図 8　各 Query 実行時の CPU 使用率
Figure 8　CPU utilization when executing each query

5.3. 問合せ結果の精度の評価

フィルタリングの手法を適用することにより，データ数が削減されて問合せ性能が改善されたが，一方で，フィルタリングしたことにより，問合せ結果がオリジナルのデータを用いた時と異なってしまう可能性があるというトレードオフがある．そこで，フィルタリング手法を適用したときの Query 1, 2, 3 の結果が，オリジナルのデータを用いたときの結果と比較して，どの程度結果を維持できていたかを検証する．

図 9 は，各クエリでの，オリジナルデータの結果のカバー率を表している．100%の場合は，オリジナルデータの結果をそのまま再現できたということを意味する．Query 1 は，指定したツイートの ID のリツイートデータをすべて取得する問合せである．すなわち，指定したツイートの拡散ネットワークを生成するための問合せとなる．ツイートの ID には，Query 2 の結果である総リツイート数トップ 100 件の ID を指定する．Query 2, 3 は，それぞれ総リツイート数，ユーザー数のトップ 100 件の結果を返すランキング問合せである．したがって，オリジナルのトップ 100 の結果のうち，何件をトップ 100 以内に維持できていたかの割合を計算している，

手法 a では，Query 2 と Query 3 の結果において，約 90%の結果を維持できている．これは，手法 a はランダムに均一にデータを削減するために，もともとリツイート数の多いツイートやユーザーは，相対的に多く残ったためと考えられる．一方，手法 b は，手法 a よりもやや低い結果となっている．この要因の一つとして，フォロワー数が少ないユーザーが発信したツイートでも，フォロワー数の多いユーザーにリツイートされたことにより多くのユーザーがそのツイートを閲覧することになり，ツイートが広く拡散した場合があると考えられる．

例えば，Query 3 のユーザーランキングのオリジナルデータでの 12 位のユーザーは，フォロワー数が 610 であった．このユーザーは手法 b では閾値 1000，2000 のいずれの時もフィルタリングされる対象となるため，手法 b 適用後の Query 3 のランキング結果には現れない．しかしながら，このユーザーは，フォロワー数が 30,498 のユーザーにリツイートされている．この結果により，普段以上にリツイート数が増えて，12 位にランキングしたのではないかと考えられる．

手法 b のカバー率は，このようにリツイートしたユーザーの属性も考慮すると，より改善できる可能性がある．現時点では，オリジナルツイートを発信したユーザーの属性しか考慮していない．

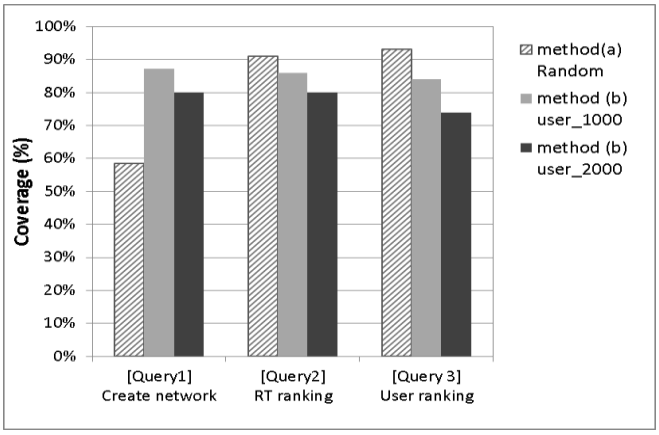

図 9　オリジナルの結果のカバー率
Figure 9　Coverage of the original results

Query 1 のカバー率は，WHERE 句で指定した ID の拡散ネットワークのエッジ総数のうち，何%を維持できたかを意味する．各ツイートの拡散ネットワークは，2.1 節で示したように，{リツイートされたユーザー，リツイートしたユーザー}を 1 エッジとするデータ構造となっている．手法 a は約 40%のエッジを失った結果となっている．一方で手法 b は閾値 1000，2000 でそれぞれ約 90%, 80%のカバー率を維持している．

本実験において，バースト時のキャパシティコントロールのため，ユーザーの属性情報を用いてデータ数を制御することによる効果を示した．

6. 関連研究

ソーシャルネットワーク上の情報拡散の分析に関する研究は大きく三種類に分類される[12]．一つ目は，流行のトピックを検出する研究である．例えば，TwitterMonitor[3]はキーワードのバーストを発見して，それらのキーワードをグループ化することにより，トレンドのトピックを抽出する．二つ目は，情報拡散の

モデル化である．例えば，リツイートで広がる時間経過の拡散の分布は，主に対数正規分布に従うとされている[13,14,15]．これらの知見に基づき，本研究においても，インメモリデータストアの，データ格納メンテナンスに適用されている[9]．

最後は，インフルエンサーの発見に関する研究である．インフルエンサーの発見には，情報拡散ネットワークの情報を用いて，ページランクを計算したり，ネットワークの中心にいる人物を発見するものがある[16]．

これらの関連研究は Twitter 上で情報拡散に関連する分析を行っているが，分析アルゴリズム等をメインにしており，問合せ処理の性能に関して言及しているものではない．我々は，これらのような分析に必要なデータを扱うミドルウェアのシステム構築と，データアクセスの最適化を目指している．

SONDY[17]は，ソーシャルネットワークのデータをインプットとして，トピック検出や，ページランクの計算などが実施できる．独自の分析も追加で組み込むことができる．SONDY はリアルタイムなデータ処理を対象にしておらず，本研究とはデータの扱いが異なる．リアルタイムな分析としては，Taxidou[18]らの Twitter の拡散分析がある．彼らの目的は拡散の広がりをリアルタイムに推定してネットワークを構築することである．

7. まとめと今後の課題

本論文にて，Twitter のようなリアルタイム性の高いソーシャルメディアの，情報拡散データをリアルタイムに分析するためのシステムのフレームワークを構築した．ソーシャルメディアにようなデータの特徴の一つにバースト性がある．あるトピックが瞬間的に大きく話題になると多くのユーザーが一斉にツイートを発信し，システムのリソースの限界に達して性能劣化を引き起こす可能性がある．

そこで，本論文では，各ツイートに付随する情報を用いて重要度を計算し，重要度が低いと判断されたツイートをフィルタリングして処理するデータ量をコントロールする手法を提案した．実際にバーストしたツイートデータを用いて，問合せ性能を評価し，効果を示した．

参 考 文 献

[1] 企業を襲う インターネットの“炎上” http://www.nhk.or.jp/ohayou/marugoto/2014/04/0416.html

[2] ソーシャルメディア運用・分析・監視ツール http://dmc-navi.sendenkaigi.com/keyword/view/3

[3] M. Mathioudakis and N. Koudas, “TwitterMonitor: trend detection over the twitter stream.” In Proceedings of the 2010 international conference on Management of data, pages 1155–1158. ACM, 2010.

[4] S. Asur, B. A. Huberman, G. Szabo, and C. Wang, “Trends in social media - persistence and decay.” In 5th International AAAI Conference on Weblogs and Social Media, 2011.

[5] Lee, C.-H, “Mining spatio-temporal information on microblogging streams using a density-based online clustering method”, Expert Syst. Appl., Vol. 39, No. 10, pp. 9623-9641, 2012.

[6] 那須川 哲哉,西山 莉紗, 金山 博,吉田 一星,大野 正樹, “一人称所有格を用いたプロフィール推定” 言語処理学会 第 19 回年次大会, 2013

[7] HBase http://hbase.apache.org/

[8] Neo4j http://www.neo4j.org/

[9] Miki Enoki, Issei Yoshida and Masato Ogushi, “Performance of System for Analyzing Diffusion of Social Media Messages in Real Time”, IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2015

[10] インメモリデータベース http://ja.wikipedia.org/wiki/%E3%82%A4%E3%83%B3%E3%83%A1%E3%83%A2%E3%83%AA%E3%83%87%E3%83%BC%E3%82%BF%E3%83%99%E3%83%BC%E3%82%B9

[11] インメモリー時代の DB 構築 http://coin.nikkeibp.co.jp/coin/sys_ranking10/img/sample3_2.pdf

[12] A. Guille, H. Hacid, C. Favre, and D. A. Zighed, “Information diffusion in online social networks: A survey,” SIGMOD Rec., vol. 42, no. 2,pp. 17–28, 2013.

[13] 松澤 有,セーヨー サンティ, 鳥海 不二夫, 陳 昱, “リツイート時系列の 3 パラメータ混合対数正規分布による分析”, 人工知能学会全国大会, 2013

[14] Galuba, W., Chakraborty, D., Aberer, K, Despotovic, Z., and Kellerer,W., “Outtweeting the Twitterers – Predicting Information Cascades in Microblogs.”, In 3rd Workshop on Online Social Networks (WOSN), 2010.

[15] Asur, S., Huberman, B. A., Szabo, G., and Wang, C., “Trends in social media: persistence and decay.” In Proceedings of the fifth International AAAI Conference on Weblogs and Social Media (ICWSM), 2011

[16] S. B. Seidman. Network structure and minimum degree. Social Networks, 5(3):269 –287, 1983.

[17] A. Guille, C. Favre, H. Hacid, and D. Zighed., “Sondy: An open source platform for social dynamics mining and analysis.” In SIGMOD ’13, (demonstration) 2013.

[18] Taxidou I, Fischer PM, “Online analysis of information diffusion in twitter.” In: WWW Companion, pp 1313–1318, 2014