ソーシャルネットワーキングサービスの統合手法

北田 駿也† 井上 潮‡

†東京電機大学大学院工学研究科 〒120-8551 東京都足立区千住旭町 5 番

E-mail: †11kmc12@ms.dendai.ac.jp, ‡inoue@c.dendai.ac.jp

あらまし 現在、多種多様な SNS が存在しており、各サービスの中では、発言を投稿しあうといったコミュニケーションが行われている。しかし、これらのコミュニケーションは各サービス内に孤立してしまい、他のサービスとのコミュニケーションが取りにくいという問題がある。例えば、あるサービス内で行われた発言は、他のサービスを利用している人からは見えにくく、またその発言に対して他のサービスからコメントをしたとしても、それは気づかれにくい。本論文では、複数 SNS からユーザーの発言を収集し、これを P2P ネットワークによって共有することによって異なるサービス間のコミュニケーションを統合する手法を提案する。

キーワード SNS, P2P, コミュニケーション統合

1. はじめに

ソーシャルネットワーキングサービス（以下 SNS）は、人と人とのつながり、コミュニケーションをサポートするサービスである。各サービスの中では、すでに友人同士の人とだけでなく、自分の趣味の合う人や、友人の友人といったところから新しいつながりを作ることや、ユーザー同士で自分の発言や写真などを見せあい、それらに対してコメントするといったコミュニケーションが可能である。

現在、SNS には Twitter、Facebook、Google+、Mixi など多種多様なサービスが存在しており、また新しい SNS も日々増え続けている。

しかし、これらのコミュニケーションは各サービス内に孤立してしまい、他のサービスとのコミュニケーションが取りにくいという問題がある。例えば、あるサービス内で行われた発言は、他のサービスを利用している人からは購読などができず、その発言は見えにくい。また、その発言に対して他のサービスから発言の URL を参照するなどしてコメントをしたとしても、それは気づかれにくい。

本研究では、異なる SNS 上の発言をピアツーピア（以下、P2P）ネットワーク上で共有することにより、異なるサービス間のコミュニケーション統合を目指す。

2. 関連研究

今田ら[1]は、Web 上の異なるサービス内で行われているコミュニケーションを統合するために、一つのサーバーに各サービスのユーザーの発言を集める手法を提案している。この手法は、ユーザーの発言を各サービスが公開しているフィード（ブログなどのコンテンツを配信用に加工した文書のこと）を利用して収集し、発言の URL と発言が参照している URL から、発言の参照関係を構築し、ユーザーに見せるというものである。これにより、ユーザーは登録されたフィード同士のコミュニケーションが可能になる。しかし、このようなサーバーはスケーラブルでないという問題がある。大量の発言をサポートするには、そのシステムはスケーラブルでなければならない。

スケーラブルな SNS として、P2P を用いた分散型 SNS の研究がある。Cuckoo[2]は、Twitter クライアントを組み込んだ Twitter と互換性のある分散マイクロブログサービスである。Aiello ら[3]は、ユーザーID 管理システムを組み込み、ユーザーのプライバシー懸念をなくした分散型 SNS のプラットフォームを提案している。しかし、これらは各サービスのコミュニケーションの孤立という問題については考慮していない。

3. 提案手法

異なる SNS 間のコミュニケーションを統合するための提案システムを図 1 に示す。まず各ユーザーは、既存の SNS から自身の発言を、API を利用して収集する。次に収集した発言を、P2P ネットワーク上で共有するために分散ハッシュテーブル（以下、DHT）を利用してユーザー同士で共有できるようにし、コミュニケーションを可能にする。これにより、異なる SNS に所属するユーザー同士のコミュニケーションが可能になる。

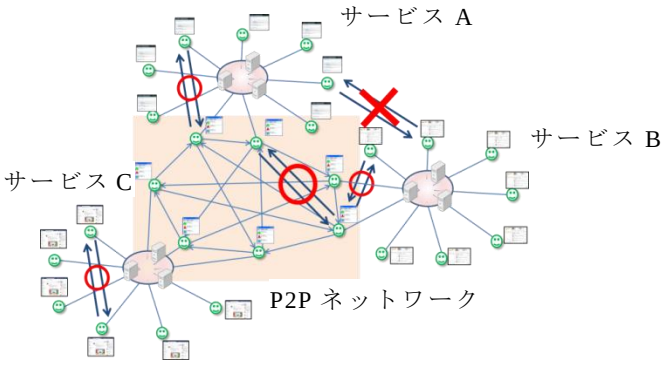

図 1 提案システムの構成

3.1. API による発言の収集

API とは、Web サービスが他のアプリケーション向けにサービスの機能を利用できるように提供しているインターフェイスのことである。一般的な SNS は、サービスを外部から利用できるように API を公開しており、これを利用することで、ユーザーが各 SNS 上で行った発言を収集する。

3.2. DHT による発言の管理

DHT とは、P2P ネットワーク上のマシンに対してデータを保存・取得するためのアルゴリズムの総称である。これはデータを検索する際、アルゴリズムによって検索メッセージの転送先を選ぶ方法をあらかじめ構造的に決めておき、「検索キーに対応する端末」を確実に分かるようにし、その端末に対して、データを保存・取得できるようにしている。構造的な DHT は、他の非構造な P2P ネットワークに比べ、ネットワーク負荷が上がらず、高速な検索が可能となる。そのため、本研究では DHT を扱い、ユーザー同士の発言データ共有やデータ管理を行なう。

3.3. ユーザーのフォローと友人関係の構築

あるユーザーの発言を見たい場合に、毎回検索するのは手間となるので、ユーザーをフォローすることで、そのユーザーの発言を自動で取得できる。また、互いのユーザーがフォローし合った状態を友人同士として扱っている。

3.4. ユーザー同士のコミュニケーション

本システムでのコミュニケーション手法は、主に発言への返信である。ユーザーA がユーザーB の発言へ返信をする場合、返信メッセージを DHT に格納する。そして、ユーザーB は DHT にアクセスすることで自分の発言への返信を知ることができる。

4. 提案手法の実装方法

4.1. システム構成

システム構成を図 2 に示す。

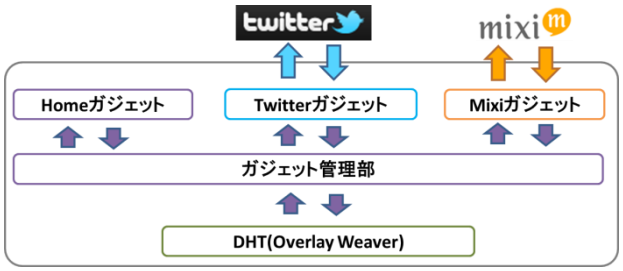

図 2 システム構成

(1) 既存 SNS からの発言の収集

既存 SNS からの発言の収集は、各 SNS の API をサポートしているガジェットによって行う。まず、ユーザーは、ガジェットを用いて利用している SNS に本システムを認可する。そして本システムは、ガジェットを通してユーザーのコンテンツにアクセスできるようになり、ユーザーの発言を収集する。

発言の収集は、現在 Twitter、Mixi から可能である。

(2) データの管理方法

データ（ユーザーのプロフィール、ユーザーの発言、フォローメッセージ、発言への返信など）の管理は DHT によって管理する。

はじめに DHT の仕組みを説明する。DHT は、表 1 のような検索キーとそれに対応する値の表（ハッシュテーブル）を、P2P を用いて各端末に分散させたものである。

検索キーはユーザーの ID のハッシュ値とし、ユーザーのプロフィールや発言のデータをこれに対応させ、ユーザーのデータを検索できる。また、データはオリジナルデータとレプリカデータを分散配置し、ユーザーが離脱してもデータが損なわれないようにする。

本システムの DHT は、Overlay Weaver[4]（オーバレイ構築ツールキット）を使用している。また、DHT のアルゴリズムとして Chord[5]を使用している。

ここで、Chord のアルゴリズムについて簡単に説明する。まず、ネットワークに参加するノードは、SHA1 のハッシュ値の値域を満たす一意なノード ID が割り当てられる（IP アドレスのハッシュ値が用いられる）。あるデータの所持をその検索キーのハッシュ値に近いノードが担当する。そして Chord では、図 3 のようにノードはそのノード ID 順に時計回りに配置され、検索も時計回りに、各ノードが所持している経路表をたどって行われる。また経路表は、図 3 の太い矢印のように自分のノード ID から 2^{i-1}離れたノードの IP アドレスを所持している。

表 1 データの管理

検索キー	値
hash (yamada)	プロフィールデータ
	発言データ
hash (tanaka)	プロフィールデータ
	発言データ
hash (suzuki)	プロフィールデータ
	発言データ

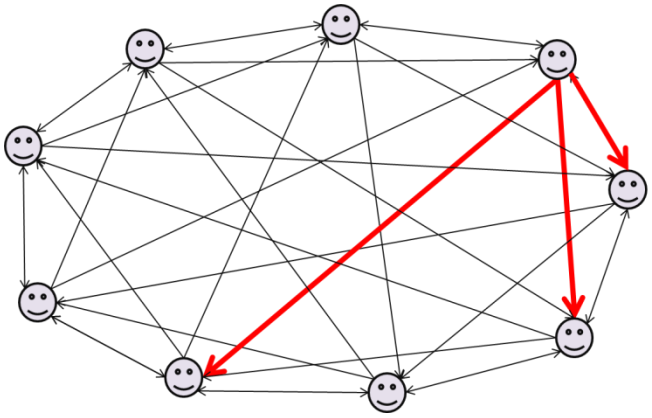

図 3　Chord のアルゴリズム

(3) 収集した発言の共有

収集した発言は、DHT 上に自分の ID を検索キーとして、保存する(表 2)。これにより、他者が自分の ID を検索した際にこの発言を取得することができる。

表 2 発言データの格納

検索キー	値
hash (yamada)	発言データ

(4) ユーザーのフォローと友人関係の構築

友人関係は、ユーザー同士がフォローすることで構築される。例として、ユーザーyamada がユーザーtanaka の友人になりたい場合を考える。

ユーザーyamada は DHT 上に検索キーを tanaka としてフォローメッセージを格納する(表 3)。そして、ユーザーtanaka は、自身の ID を検索することで、自分がフォローされたことを知ることができる。

同様にユーザーtanaka がユーザーyamada をフォローすることで、両者は友人となる。また、フォローしたユーザーの発言は自動で DHT 上から検索し、収集するようになり、タイムラインとして自分の発言とフォローした人の発言を時系列順で表示する。

表 3 フォローメッセージの送信

検索キー	値
hash (tanaka)	フォローメッセージ

(5) ユーザー同士のコミュニケーション

本システムでのコミュニケーション手法は、発言への返信である。表 4 のように DHT 上に返信相手の ID をキーとして返信メッセージを格納する。そして、返信相手は自身の ID を検索することで、自分に返信があったことを知ることができる。

表 4 返信メッセージの送信

検索キー	値
hash (tanaka)	返信メッセージ

4.2. システムの実行画面

動作画面例を図 4 に示す。本システムは、画面上にガジェットに対応したカラムを配置する。図 4 では、左から Home ガジェット、Twitter ガジェット、Mixi ガジェットが配置されている。Home ガジェットでは DHT 上に対して発言の投稿やユーザーの検索、ユーザーのフォローなどができる。Twitter ガジェットでは、Twitter のコンテンツへアクセスし、発言の投稿や、収集を行える。Mixi ガジェットでは、Mixi のコンテンツへアクセスし、発言の投稿や、収集を行える。

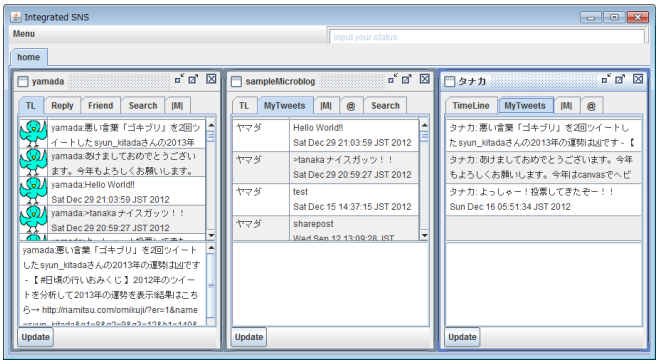

図 4 システムの実行画面

4.3. 考察

ここで、システムの動作検証をする。

yamada と tanaka が相互にフォローして友人同士になっており、yamada は Twitter を利用しており、tanaka は Mixi を利用しているとする。

この時点で、yamada のタイムライン（TL）には、自分の twitter から収集した発言と tanaka の mixi から収集した発言が時系列順に表示されている（図 5)。そして、tanaka の発言の中で気になった発言に対して返信メッセージを送信できる(図 6)。メッセージを送ると、tanaka は自身の Reply 画面にその返信メッセージが表示される（図 7)。同様に tanaka が yamada の twitter から収集した発言に対して返信メッセージを送ることもできる。

このように、本システムでは異なる SNS 同士でのコミュニケーションが可能になることが確認できた。

図 5 yamada のタイムライン

図 6 yamada の返信画面

図 7 tanaka の返信メッセージリスト

5. 評価

提案手法のスケーラビリティについて評価した。これは、既存 SNS の大量の発言をサポートするには、そのシステムはスケーラブルでなければならないためである。スケーラビリティを評価するため、応答速度と通信量をシミュレーションにより測定した。シミュレーション実験には、OverlayWeaver に付随する分散環境エミュレータを利用している。また、本提案手法では P2P を使った手法をとっているが、比較検証のため P2P を使わず一つのサーバーに各サービスの発言を集めて同等の機能を持ったシステムを構築し比較した。こちらの手法を以降では比較手法と呼ぶことにする。

5.1. 評価環境

実験環境を表 5 に示す。また、分散環境エミュレータのシステム構成を図 8 に示す。実験では、事前にシナリオファイルを用意して行い、シナリオ解析部でこれを解析し、それに従って、本システムのインスタンスを起動させ操作する。通信制御部はインスタンス間のメッセージの送受信の中継を担っている。また、本システムのインスタンスでは、送受信した通信を計測してその結果を出力できるようにしており、実験時に各インスタンスの計測結果を出力し、実験後にこの出力結果を解析することで評価データを得る。

次に、シミュレート時に Twitter、Mixi に対して大量のリクエストを送ることは迷惑となるため、その代わりとなるクローンサーバーを作成した。また、クローンサーバーの API は、実際の API から認証機能等のシミュレーションでは不必要な機能を省略した簡易な API である。

表 5 実験環境

OS	Windows7 Professional 64bit
CPU	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
メモリ	16GB
使用言語	Java 7

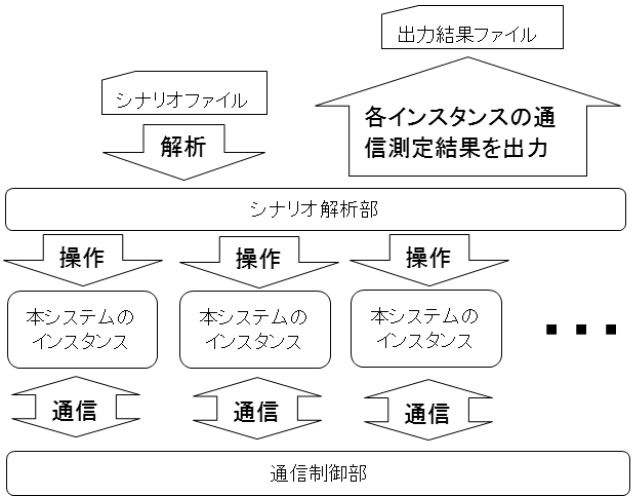

図 8 分散環境エミュレータのシステムの構成図

5.2. 通信速度の評価実験

(1) 評価方法

本システムにおいて、ユーザー数を 10 から 1000 まで変化させていった場合の応答ホップ数を計測する。処理コマンドは、発言の投稿と、発言を取得する更新処理をそれぞれ全体で 5000 回ずつ行い、その際のホップ数を計測する。ここでホップ数とは、P2P ネットワーク上で端末を検索する際に経由した端末数であり、これが多ければ応答速度も遅くなる。また、ユーザー一人あたりの平均フォロー数を全ユーザー数の 1 割として実験を行なっている。また、比較手法については、クライアント・サーバーの形を取っているのでそのホップ数は常に 1 であるので実験は行わない。

(2) 実験結果

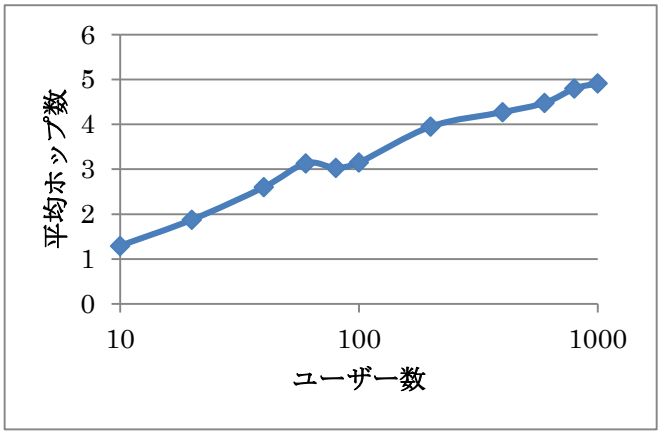

図 9　ユーザー数と平均ホップ数の関係

(3) 考察

図 9 より、比較手法の応答ホップ数が 1 なのに対して、提案手法のホップ数は、ユーザー数が 10 人の場合は 1.29 と大差ないが、1000 人となると 4.9 ホップとなり、比較手法に比べて 5 倍近く遅くなってしまうことがわかる。

本提案手法では、指定したユーザーの発言を DHT の Chord を利用して保存・取得を行なっている。しかし、Chord ではデータの探索の際に最大で、$\log_2 N$（N はノード数）のホップ数を必要とし、ユーザー数の増加にあわせてホップ数は増える特性を持っている。これより、ユーザー数 1000 人であれば、最大で 10　$(\log_2 1000)$ ホップ必要である。また、最小では自身がデータを持っている場合があり 0 ホップなので、その平均値は 5 となるはずである。今回の実験では、ユーザー数 1000 人の場合に、平均ホップ数が 4.9 となり、この値は妥当だといえる。

また、仮にユーザー数が 2 億人（2012 年 12 月 18 日時点での Twitter のアクティブユーザー数）だとすると最大で、27.5 ホップかかることになる。

以上のことより、多くのユーザーをサポートし、リアルタイムな応答性を求めるためには、Chord 以外のアルゴリズムについても検討し、検索時のホップ数を少なくする必要があると考えられる。

5.3. 通信量の評価実験

(1) 評価方法

2 つの手法の送受信通信回数について比較する。まず、ユーザー数を増やしていき、その際発言の投稿と、発言を取得するリクエストをそれぞれ全体で 5000 回ずつ行い、通信回数を計測し比較する（図 10）。次に、ユーザー数を 1000 人に固定して発言の投稿と発言を取得するリクエスト数を増やしていった場合の通信回数を計測し比較する（図 11）。また、ユーザー一人あたりの平均フォロー数は全ユーザー数の 1 割として実験を行なっている。

(2) 実験結果

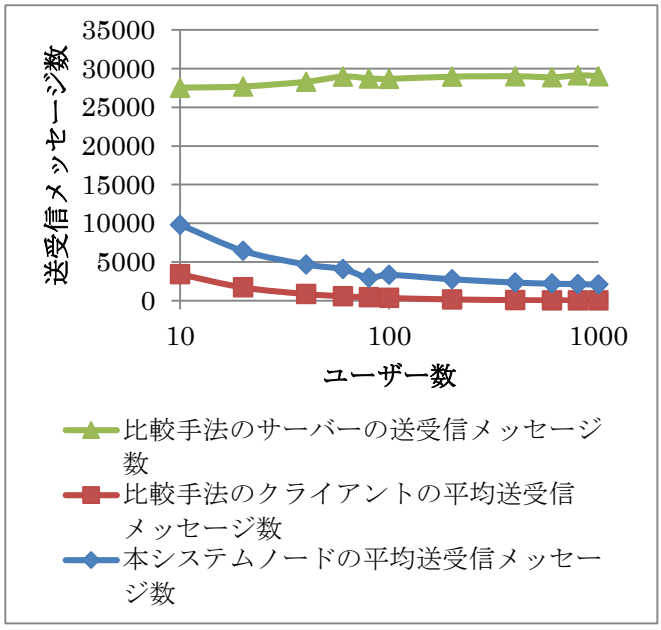

図 10　ユーザー数と通信回数の関係

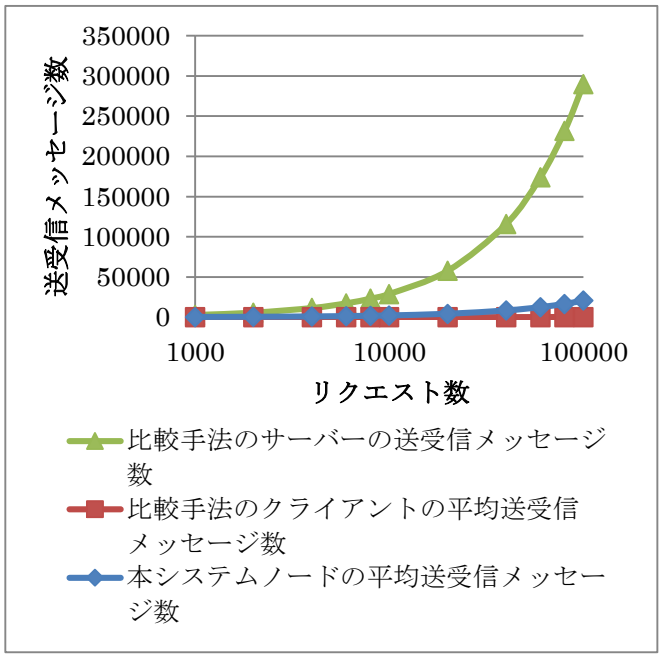

図 11 リクエスト数と通信回数の関係

5.3.1. 考察

まず、図 10 よりユーザー数と通信回数の関係についてであるが、こちらは発言の投稿と、発言を取得するリクエスト数がそれぞれ全体で 5000 回ずつと一定なので、本システムノードの通信回数は、ユーザー数

が増えれば通信量も分散され、一人あたりの負荷は少なくなっていくことがわかる。一方、比較手法では、クライアント側はユーザー数が増えれば一人あたりの通信量も少なくなるが、サーバーに関してはユーザー数が増えてもリクエストされる量は一定なので、通信回数も一定値となり、通信の負荷が集中することがわかる。

次に、図 11 よりリクエスト数と通信回数の関係についてであるが、本システムノードの通信量は、リクエスト数が増えても、通信量は分散されるためゆるやかに上昇していく。一方、比較手法では、クライアント側は、リクエスト数が増えても通信量はほとんど上昇しないが、サーバー側は、リクエスト数が 10000 を超えたあたりから急激に上昇している。このことからもサーバーに通信の負荷が集中していることがわかる。

5.4. スケーラビリティの評価実験

(1) 評価方法

本システムが、何人のユーザー数まで利用可能かを評価する。

まず、ユーザー数に比例してどの程度のリクエストが発生するか想定する。Twitter のアクティブユーザー数が 2 億人、1 秒間あたりのツイート数が最高で 3 万 3388(2013 年 1 月 1 日時点)とあり、1 秒間にユーザー数の 0.016694%が投稿している。また、タイムラインの更新処理も同程度以上あると考えられる。そこで、今回は 1 秒間にユーザー数の 0.04%の投稿・更新の処理が発生するものとし評価する。

また、1 端末が 1 秒間に処理できるメッセージ数を 1 とし、この数値を下回った場合に、利用可能であると判断して評価する。

実験方法は、ユーザー数を徐々に増やしていき、発言の投稿・取得のリクエスト処理をさせる。そして、その時の 1 端末あたりの処理メッセージ数から、1 秒間にユーザー数の 0.04%の処理が発生した場合に 1 端末が処理しているメッセージ数を求める。その際の算出式を式(1)に記す。また、ユーザー一人あたりの平均フォロー数を全ユーザー数の 1 割としており、またその上限を 500 人（2012 年 4 月 21 日時点での Twitter の平均フォロー数が 325 人のためこれを上回る値を設定した）として実験を行なっている。

$$y = \frac{x}{\frac{r}{n \times \frac{0.04}{100}}} \qquad (1)$$

ユーザー数を n、リクエスト数を r、1 端末あたりの処理数を x、ユーザー数の 0.04%のリクエストがあった場合の 1 端末あたりの処理数を y とする（図 12）。

(2) 実験結果

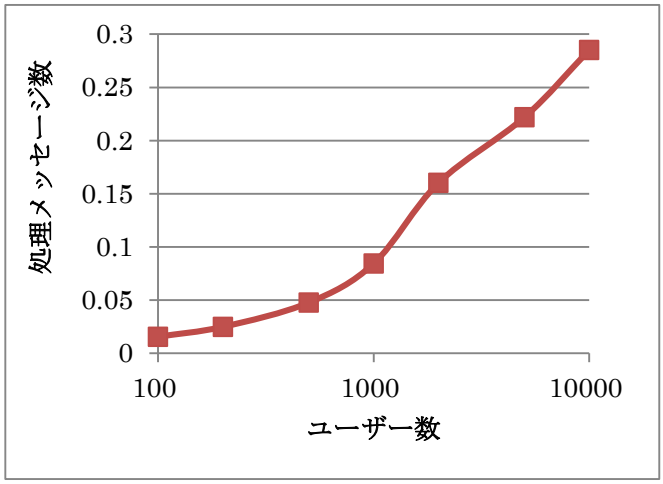

図 12　ユーザー数と処理量の関係

(3) 考察

ユーザー数が 10000 人までは本システムを利用可能であることが確認できた。現在、10000 人より多くのユーザー数についてはシミュレーションの時間がかかりすぎるため確認できていない。そのため、何人のユーザー数まで利用可能かどうかの正確な結果は得らなかった。

シミュレーションの時間は、ユーザー数が 100 人の場合は 11 分、1000 人では 54 分、5000 人では 15 時間、10000 人では 30 時間かかった。このため、スケーラビリティについて評価するためにはより多くのユーザー数を想定して実験すべきだと考えられるが、今回の評価方法ではスケーラビリティについて正しく評価できないことがわかった。また、これを評価するための実験手法を検討する必要があると考えられる。

6. まとめと今後の課題

複数 SNS からユーザーの発言を収集し、これを P2P ネットワークによって共有することによって複数 SNS のコミュニケーションを統合する手法を提案した。また、提案手法を実現するためのシステムを実装し、動作検証において Twitter と Mixi に接続し、ユーザーの発言を共有して異なる SNS 同士でのコミュニケーションが可能になることが確認した。さらに、OverlayWeaver に付随する分散環境エミュレータを用いてスケーラビリティについて評価した。その結果、通信速度に関しては、本手法はユーザー数が 10 人の場合はデータの保存・取得の際のホップ数が 1.3 と高速な応答が可能であるが、ユーザー数が 1000 人となるとホップ数 4.9 となり、リアルタイムな応答性は難しいとわかった。通信量に関しては、サーバーを使った場合に比べて通信量を分散させられるため、ユーザ

一数が増えれば負荷も少なくなることがわかった。また、スケーラビリティについては、ユーザー数が 10000 人までは利用可能であると確認できたが、それより多くのユーザー数については利用可能かどうか判断できなかった。

今後の課題は、データの保存・取得の際のホップ数を軽減し、リアルタイムな応答性を実現することである。そして、今回の実験ではどの程度のユーザー数まで利用可能かというスケーラビリティについての評価ができておらず、これを評価するための実験方法を検討する必要がある。また、本手法ではセキュリティについては考慮しておらず、ユーザーの発言はすべて公開されるように設計されている。しかし、ユーザーのニーズとしては発言を自分の許可したユーザーなどにしか見せたくないという需要もあるため、ユーザーの発言を守るセキュアな仕組みが必要となる。また、スパムユーザーなどの悪意あるユーザーに対しても対処していく必要がある。

参 考 文 献

[1] 今田智大, 矢吹太朗, 佐久田博司, "複数のソーシャルサービスを統合するコミュニケーションツールの開発", DEIM Forum 2011 A2-3.

[2] Tianyin Xu, Yang Chen, Jin Zhao, Xiaoming Fu , "Cuckoo: Towards Decentralized, Socio-Aware Online Microblogging Services and Data Measurements", HotPlanet’10 , June, 2010.

[3] Luca Maria Aiello, Giancarlo Ruffo, "Secure and Flexible Framework for Decentralized Social Network Services", IEEE PERCOM 2010, PP.594-599.

[4] 首藤一幸, 田中良夫, 関口智嗣, ”オーバレイ構築ツールキット Overlay Weaver”, 情報処理学会論文誌: コンピューティングシステム, Vol.47, No.SIG12 (ACS 15), PP.358-367.

[5] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications", Proc. ACM SIGCOMM