Pub/Sub基盤におけるユーザエージェントの移動による システム最適化に関する一考察

A consideration on performance optimization of Pub/Sub infrastructure using User Agent migration

檜垣　貴良†　　秋山　豊和†

† 京都産業大学コンピュータ理工学部　〒 603-8555 京都府京都市北区上賀茂本山
E-mail: †{g1044921,akiyama}@cse.kyoto-su.ac.jp

あらまし　PIAX（P2P Interactive Agent eXtensions）は、P2P 構造化オーバーレイネットワークとエージェント機構を組み込んだオープンソースのフレームワークであり，Pub/Sub メッセージングモデルをサポートしている．本研究では，PIAX をエンドユーザとネットワークを調停するためのミドルウェアとして利用し，Software Defined Network (SDN) との連携によりネットワークの最適化が可能な Pub/Sub 基盤の構築を目指している．本稿ではまず，SDN を用いたシステムの事前評価として，SDN を用いていない PIAX の配信を最適化する方法として，エンドユーザの Pub/Sub 基盤へのランデブポイントであるユーザエージェントの移動を利用することを想定し，PIAX 上でのユーザエージェント移動性能の計測を実施する．計測結果からユーザエージェント移動による最適化の可能性について考察する．

PIAX (P2P Interactive Agent eXtensions) is an open source framework that incorporates the agent mechanism and structured P2P overlay network. and it supports the Publish / Subscribe (Pub / Sub) messaging model. In this reserch, we are aiming to construct a Pub/Sub infrastructure that can optimize network through the cooperation with Software Defined Network (SDN) by utilizing PIAX as a middleware for arbtrating the network and end-users. In this paper, as a pre-evaluation of the SDN aware Pub/Sub system, we evaluate the agent migration performance of PIAX because we assume that the agent migration can be used for the optimization of message delivery in our Pub/Sub environment. From the evaluation results, we will discuss the applicability of the agent migration to the delivery optimization.

キーワード　Pub/Sub，オーバーレイネットワーク，エージェント移動　　Overay netowork，Agent migration

1. はじめに

近年では，SNS，ソーシャルゲーム，メッセージングサービスなどのサービス普及に伴い，エンドユーザ間での情報の発信，取得，交換が盛んになっている．このようなサービスでは，Publisher/Subscriber（Pub/Sub）メッセージングモデル (出版・購読型モデル) と呼ばれる通信形態をサービスのバックエンドで利用している [1]．Pub/Sub メッセージングモデルとは，メッセージの送信者が特定の受信者を想定せずにメッセージを送る非同期メッセージングモデルである．Pub/Sub メッセージングモデルには，コンテンツベースおよびトピックベースのものが存在するが，本研究では，トピックベースの Pub/Sub を対象とする．具体的なトピックとしては，例えば，Twitter ではフォローしているユーザ，チャットシステムでは，チャットルームなどがあげられるが，本稿ではトピックの定義方法については議論しない．以下では，Pub/Sub メッセージングモデルに基づく通信を提供するネットワークを Pub/Sub ネットワークと呼ぶ．

本研究では，Pub/Sub ネットワークを単一のデータセンタ内で利用するのではなく，広域ネットワークにまたがるネットワークサービスとして利用することを考える．これにより，多くのサービスのバックエンドに流れるトラフィックを Pub/Sub ネットワークというミドルウェアレイヤで抽象化し，共通の基準で最適化することを目指す．以下ではまず 2 節で本研究で用いる Pub/Sub ネットワークミドルウェアについて述べ，本研究での Pub/Sub ネットワークを利用したアプリケーションの構成方法を紹介する．3 節では，提案する構成におけるアプリケーション最適化のアプローチについて述べる．4 節では提案構成の基礎的な項目に関する計測結果について述べ，5 節で本稿のまとめとする．

2. Pub/Sub ネットワークミドルウェア

Web サービスのバックエンドとして利用される Pub/Sub ネットワークのミドルウェアとして，オープンソース製品である Redis（Re-Mote Dictionary Server）[2] のような Key Value Store（KVS）を拡張したミドルウェアが利用されることが多い．しかし，広域に分散した拠点間で Pub/Sub ネットワークを効率的に動作させるには，より多数のノード間の連携を考慮したミドルウェアが必要となる．本研究では，P2P 構造化オーバーレイネットワークとエージェント機構を組み込んだオープンソースフレームワークである PIAX (P2P Interactive Agent eXtensions) を Pub/Sub ネットワークのミドルウェアとして利用する．以下では PIAX の特徴と，本研究で想定する PIAX を用いた Web アプリケーションの構成について述べる．

2.1 P2P 構造化オーバレイネットワーク

PIAX では，複数の構造化オーバレイネットワークを切り替えて利用する機能を提供している．構造化オーバレイネットワークでは対象となるコンテンツの特性や検索方法に適したネットワークを選択することで効率的な検索やネットワークの維持管理が実現できる．本研究では，Pub/Sub ネットワークならびに後述するエージェントの検索を目的とするため，Multi-key Skip Graph（MKSG）[3] を構造化オーバレイとして利用する．

2.2 エージェント機構

エージェントとは，構築されたオーバーレイネットワーク上で，エンドユーザやデバイスに代わって処理を実行するインスタンスを指す．エージェントには属性があり，“topic” などの任意の名称の属性とその値を追加できる．図 1 は，PIAX におけるピア，エージェント，エージェント属性の関係を示したものである．

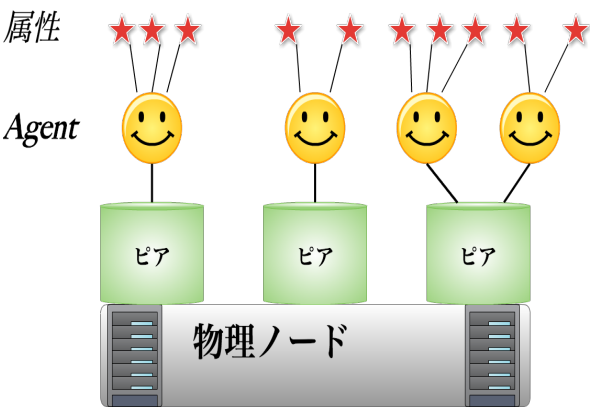

図 1　PIAX ネットワークの構成図

本研究では，PIAX が提供しているエージェントを継承する形で，エンドユーザやセンシングデバイス等の要求をオーバレイネットワーク上で代行するユーザエージェントを実装した．実装したユーザエージェントを Publisher および Subscriber として動作させる事で Pub/Sub ネットワーク機能を実現する．ユーザエージェントはオーバレイネットワーク上に属性値を登録することで，属性値を用いて互いに検索し，互いのメソッドを呼び出すことができる．属性の検索時には Query が指定でき，Query によって任意のユーザエージェント群を柔軟に選び出すことができる．メソッド呼び出しとして，同期，非同期，要求/応答，片方向などの呼び出し形式が提供されている．エージェントは複製や移動にも対応しており，エンドユーザやデバイスからの要求に応じて，複製ならびに移動させることが可能である．ユーザエージェントを Subscriber として動作する際には，ユーザエージェントが購読したいトピックを属性として追加し，オーバレイネットワークに登録する．Publisher として動作する際には，ユーザエージェントが配信したいトピックを Query で指定し，それらのユーザエージェントに対してメッセージを送信する．MKSG では，最小限のエージェントのみを経由してメッセージを配信することができるため，アプリケーション層おいては効率的にメッセージを配信できる．

2.3 Web アプリケーションの構成方法

本研究では，PIAX を Web アプリケーションのバックエンドとして利用する．PIAX は Java で実装されており，Redis のようにフロントエンドの Web サーバとは別のサーバとして構築する方法も考えられるが，本研究では，ノード上のインスタンス数を少なくするため，単一の JavaVM 上に PIAX のピアと Web アプリケーションサーバを立ち上げて，相互に連携させることとした．なお，Web アプリケーションサーバとしては Jetty [4] のバージョン 9.1 を用いている．図 2 に想定する構成の概要図を示す．エンドユーザはブラウザから HTTP や WebSocket [5] 等のプロトコルを用いてピア上で動作する Jetty のインスタンスにアクセスする．Jetty のインスタンスはユーザエージェントの API を用いてオーバレイネットワーク上のユーザエージェントにアクセスする．本研究では，エンドユーザに対応するユーザエージェントをオーバレイネットワーク上に配置し，Publisher/Subscriber として動作させる．ブラウザへの配信は，サーバ側からの PUSH 通信をサポートする WebSocket を用いて実現する．

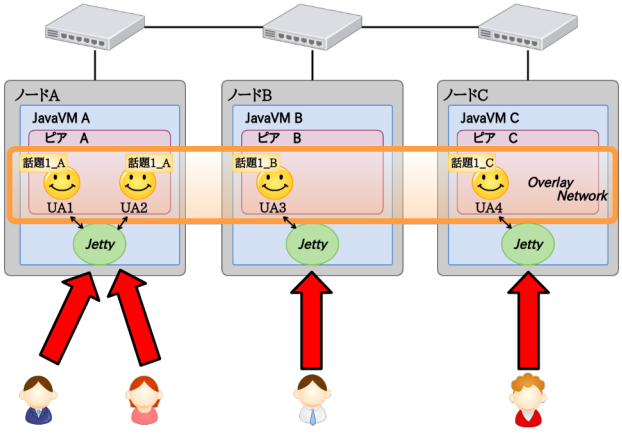

図 2　想定する構成概要図

3. 提案構成における最適化のアプローチ

本節では，提案する Web アプリケーションの構成方法において，サービスの応答時間やネットワークならびにサーバ負荷を最適化するアプローチについて述べる．

3.1 ランデブサーバの選択とエージェント移動による応答時間の改善

本研究が対象とする Web アプリケーションは，広域なネットワーク上で提供されるサービスであり，Web サーバは異なるデータセンタ上に分散して配置されることを想定してる．そのためサービス提供者は，ユーザやデバイスが他のユーザやデバイスとメッセージの送受信を行うために接続するサーバ（ランデブサーバ）を決定する仕組みを提供する必要がある．一般的な方法として，Contents Delivery Network（CDN）のように，ネットワークごとに最寄りのサーバの IP アドレスを DNS で提供する方法が挙げられる．また，ランデブサーバを選択するサービスを提供し，すべてのユーザにランデブサーバ選択サービスへアクセスさせることで適切なランデブサーバを選択する方法も考えられる．さらに，データセンタ内で負荷分散を行う場合は，負荷分散装置などを間に挟むことで適切なサーバが選択できる．今回想定している Web アプリケーションも，同様な方法によりランデブサーバを決定する．ランデブサーバ選択サービスや負荷分散装置を利用する場合，JSESSIONID のようなセッション ID を用いることで，ユーザに常に自身のユーザエージェントが待機するサーバを選択させることで，ユーザエージェントの状態復帰を短い応答時間で実現できる．しかし，ユーザがネットワークを移動することにより，ユーザエージェントが待機するサーバへの接続が必ずしも最適ではないケースが生じる．具体例を図 3 に示す．この例では，東京のネットワークにいるユーザが東京のデータセンタのサーバにアクセスしてユーザエージェントを生成し，その後京都のネットワークに移動した場合を示している．ユーザエージェントを生成した際のネットワークにしばらく留まる場合は，セッション ID 等を用いて同じサーバにアクセスさせることで，再接続時の応答時間を短縮できる．一方，京都のネットワークに移動した場合，同じ東京のサーバにアクセスさせると，サーバまでの通信遅延により応答時間が悪化する．ここで，移動先の京都のデータセンタのサーバにアクセスさせることでピアまでの通信遅延を短縮することができるが，ユーザエージェントの状態復帰ができなくなる．このような場合にユーザエージェントの状態復帰をする方法として，

（1） 京都のサーバから東京のサーバのユーザエージェントにアクセスして状態を取得する

（2） 東京のサーバから京都のサーバにユーザエージェントを移動させる

という方法が考えられる．ユーザの移動後，しばらくアクセスが続く場合は (2) の方が応答時間を短縮できると考えられる．このように広域ネットワークにおいてはユーザエージェントの移動がネットワーク最適化において重要な役割を果たす．

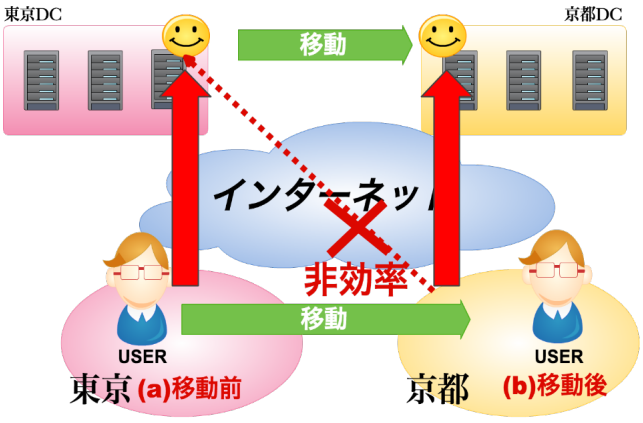

図 3 ユーザエージェント移動が必要な例

3.2 サーバおよびネットワーク負荷の最適化

前節で述べたように，ユーザエージェント移動は，広域ネットワークでの応答時間の改善に活用できる．一方，データセンタ内でのサーバ負荷分散やサーバ間通信の最適化においても，ユーザエージェント移動が活用できる可能性がある．提案する構成での最適化について述べる前に，2 節で述べた Redis のようなオープンソースの KVS を Pub/Sub 基盤として活用した場合について述べる．本節では利用が簡単な単一サーバで実現する Pub/Sub 基盤の場合を例にして提案する構成と比較する．なお，Cassandra や HedWig 等の分散 Pub/Sub 基盤については関連研究の節で述べる．図 4 は Redis を利用しているデータセンタ内の Web サーバと Redis の接続構成の例を示したものである．各 Web サーバはクライアントからの Publish/Subscribe 要求を Redis に転送する．そのため，クライアントと Web サーバ間と同様の通信がバックエンド側でも発生する．この場合，クライアントからの Pub/Sub 要求が，増加するとバックエンドで動作している Pub/Sub 基盤のネットワークトラフィックも増加する．一方，提案する構成では，バックエンドのネットワーク負荷を最適化することが可能である．図 5 は PIAX を利用した際の，データセンタ内サーバの負荷分散の様子を示したものである．この場合，クライアントからの Pub/Sub 要求が増加しても，同じトピックを購読するエージェントを移動によって集約することで，バックエンドのネットワークトラフィックが低減できる．しかし，提案構成でトピックごとにユーザエージェントをクラスタリングし，クライアントのリクエストをリダイレクトすることで，バックエンドトラフィックを最適化した場合，逆にサーバ負荷が集中する可能性がある．そのため，サーバ負荷とネットワーク負荷のバランスを考えてユーザエージェントの再配置を行う必要がある．また，それ以前にユーザエージェント移動の際のトラフィックやサーバ負荷も無視できない．そこで，本稿ではまず基礎的な調査として，エージェント移動にかかる負荷を計測し，ユーザエージェント移動をベースとした Web アプリケーション構築手法の可能性について検

討する．

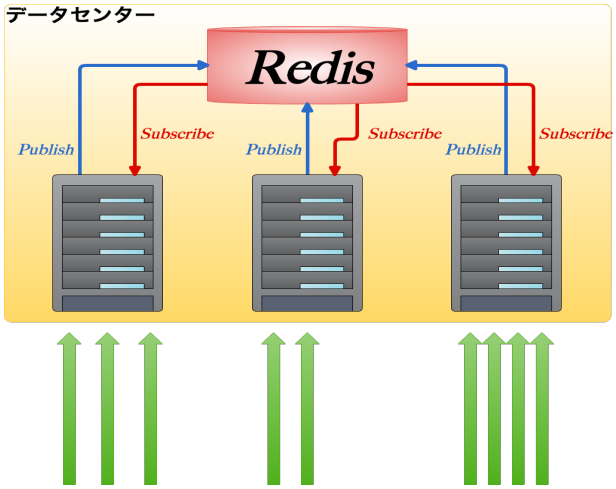

図 4　Redis とサーバとの関係

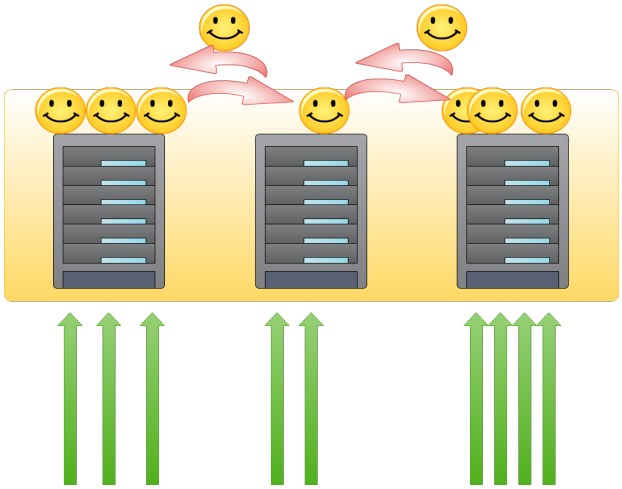

図 5　提案構成を用いたバックエンドトラフィックの最適化

4.　提案構成の基本性能調査

前節で述べたように，広く用いられている Pub/Sub 基盤を用いた Web アプリケーション構成に対して，提案構成を用いることでネットワーク負荷が低減できる可能性がある．しかし，提案構成において Web サービスを提供する上での基本的な動作の特性は把握できておらず，既存の環境に対する優位性が確認できているわけではない．そこで，以下ではユーザエージェント移動時の処理時間についての実験により調査する．実験では Web ブラウザからのアクセスにより，ユーザエージェントを移動させた場合のサーバ側での処理時間を計測する．ユーザエージェントを移動させるには，Web サーバは，クライアントからのアクセス要求が到着した際，Web サーバでは，何らかの認証手続きの後，ブラウザが提示するユーザ ID に基づき対応するユーザエージェントを検索する．なお，実験では認証手続きは省略する．PIAX では 2.2 節で述べたように，Query によるユーザエージェント属性の検索と同時にエージェントのメソッドを呼び出すことができるため，Web サーバと同じ VM 上に存在するピアの ID を取得し，遠隔のユーザエージェントの移動先として指定した上で，移動メソッドを呼び出す．これにより，ユーザエージェントはクライアントからアクセス要求を受けた Web サーバに移動する．この移動にかかる処理時間を以下のようにして計測した．

（1）　2 つのピアを立ち上げる．以下，peer1 と peer2 とする．
（2）　peer1 にユーザエージェントを新規生成する．
（3）　peer2 から peer1 上のユーザエージェントを探索し，移動させる．
（4）　Web ブラウザからのアクセス要求が Web サーバに到着してから，(3) の処理にかかる時間を計測する．

PIAX 上ではエージェントの属性はエージェントがもつ属性テーブルに格納される．よって，購読するトピック数が多ければ，エージェントインスタンスのサイズは大きくなり，ユーザエージェントの移動に時間がかかると考えられる．また，移動前後のオーバレイネットワークへの再登録処理も時間を要すると考えられる．そこでユーザエージェントが購読しているトピックサイズが，ユーザエージェントの移動にどれほど影響するのかを検証する実験を行った．さらに，ユーザエージェントが変数を保持した場合に，保持する変数のサイズでどの程度移動時間が変化するのも確認した．まず，この実験を行った実行環境について説明する．

4.1　本実験での実行環境について

今回の実験時の実行環境を以下に記す．

機種名	iMac
プロセッサ名	Intel Core 2 Duo
プロセッサ速度	3.06GHz
メモリ	8GB
java version	1.7.0_40
java option	-Xmx3072m -Xms3072m
プロファイラ	VisualVM 1.3.6

表 1　実 験 環 境

本実験では，応答時間の計測を VisualVM [8] を利用している．VisualVM は，JVM で実行中の Java アプリケーションに関する詳細情報を表示するための，ビジュアルインタフェースを提供するツールである．VisualVM をプロファイラとして得た計測結果とコード内に埋め込んだタイマとの誤差は求めている計測結果に比べ，非常に小さいものであったため，本研究では，この誤差を無視し，プロファイラによって計測された結果を採用する．(図 6 参照)

本実験ではユーザエージェント移動前に，ユーザエージェントに指定した個数，0 個，100 個，250 個，500 個のトピックを Subscribe する．本実験ではユーザエージェント移動時間計測に焦点をおくため，ガベージコレクション（GC）による処理時間の遅延を防ぐ必要がある．実験開始時，JVM のヒープ最大サイズおよび初期化サイズを 1GB に設定した．しかし，250

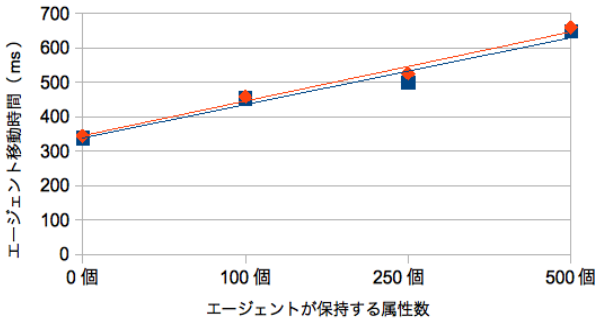

図 6　プロファイル結果値とコード内タイマの差異

個のトピックをユーザエージェントに Subscribe させると GC アクティビティ値が変動し，CPU 使用率の 50%以上を占めてしまう場合が発生し，ユーザエージェント移動時間に大きく影響した．同様に，2GB では，トピック数を 500 個にした際に遅延が大きくなった．そのため，以下では 3GB を指定し，GC による影響がない環境で実験を行った．

4.2　エージェント属性数の変化による移動時間への影響

提案する Web アプリケーションをユーザが利用している上で，購読するトピックを増やしていく場合を想定する．この節の実験では，ユーザエージェントの属性数によってどの程度移動時間へ影響しているのかを確認する．ユーザエージェントが保持する属性の数を変更し，計測を行った．属性数は 0 個，100 個，250 個，500 個の 4 段階に変化させて計測した．結果を図 7 に示す．この結果から属性数が大きいユーザエージェントを移動させると，ユーザエージェントが保持する属性数に伴って移動時間に遅延が発生する事が確認できた．

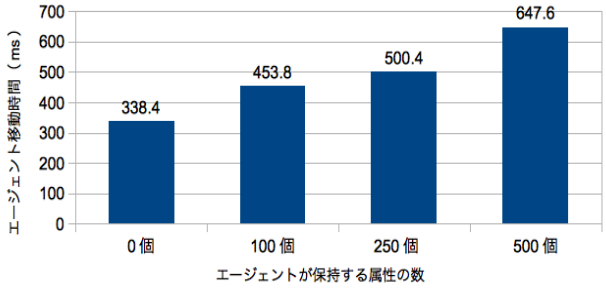

図 7　ユーザエージェント移動における属性数別平均応答時間

ユーザエージェントの属性数に伴う移動時間の遅延原因として，TCP コネクション確立並びに，ユーザエージェントのシリアライズ化および転送コストが挙げられる．シリアライズ化とは，オブジェクトの状態を 1 バイトずつ読み書きできるバイト配列状態のデータ構造に変換する事である．ユーザエージェントの保持する属性によりシリアライズ化されたイメージサイズは変化するため，大きな属性数を保持することは，ユーザエージェント移動時において，移動時間の遅延を引き起こす一つの原因となり得る．そこで，ユーザエージェントが保持する属性数の変化による移動時間を計測する際，移動するユーザエージェントをシリアライズ化したバイト配列のサイズも計測した．その結果を図 8 に記す．属性数の増加に伴って，シリアライズ化されたイメージサイズも増加していることが確認できる．

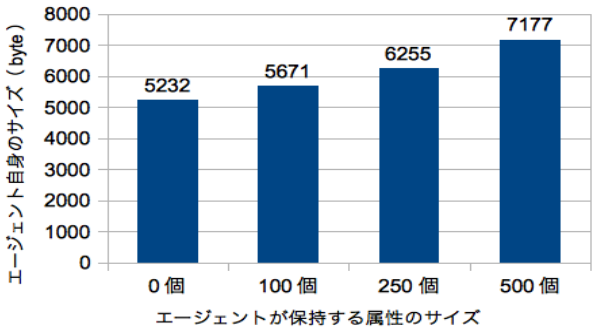

図 8　シリアライズ化されたユーザエージェントのサイズ（属性数別）

4.3　属性再登録処理によるオーバヘッド時間

PIAX では，各エージェントの保持する属性を追加する際，そのエージェントがいるピアおよびオーバレイネットワークにも同時に登録される．それは，ユーザエージェント移動時でも同様で，ユーザエージェントの保持する属性は，移動先ピアおよびオーバレイネットワークへも登録する．ここで，ピアおよびオーバレイネットワークへ属性を再追加する事に焦点を置き，ユーザエージェント移動について説明する．移動するユーザエージェントのコピーを移動先へ送信し，ピアおよびオーバレイネットワークへ属性の再登録処理を行う．その後，移動元ピアでオリジナルのユーザエージェントを削除する．その際，移動元のピアおよオーバレイネットワークの属性も削除する．以下では，再登録処理および削除によるオーバヘッド時間が属性数によってどの程度変化するのか確認した．

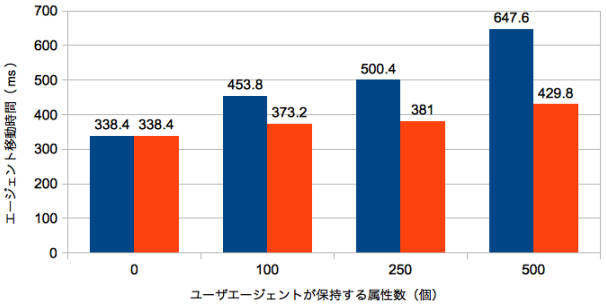

図 9　属性再登録処理によるオーバヘッド時間

属性数が 100 個，250 個，500 個の場合と同程度の大きさの byte 配列変数を保持させたユーザエージェントを移動させ，属性を持つユーザエージェントを移動した場合と移動時間を比較した結果を図 9 に示す．単純にユーザエージェントに状態を持たせる場合に比べ，属性再登録のオーバヘッドが大きいことが分かる．

4.4 考察と今後の課題

PIAX を Web アプリケーションのバックエンドとして利用する場合に，ローカル PC 上での Web ブラウザから Web サーバへのアクセス遅延が数十 ms 程度であるのに対して，エージェント移動の遅延が数百 ms 程度かかるため，無視できない大きさとなっている．3.1 節で述べたようなアクセスする場所が変更されるような場合であれば，移動はそれほど頻繁に発生しないと考えられるが，3.2 節で述べたような負荷分散において利用する場合，頻繁に移動を利用すると，サービス品質の低下につながる可能性がある．今後，エージェント移動ならびに Publish/Subribe にかかるサーバの CPU 負荷やネットワーク負荷についても計測し，具体的にエージェント移動がエージェント配置の最適化に適用可能なケースを明らかにし，既存の Pub/Sub 環境と比較していく必要がある．

5. 関 連 研 究

3 節においては簡単のため単一サーバを用いた Pub/Sub を用いた Web サービスの構築例を比較対象とした．しかし，実際にはオープンソース製品にも分散 Pub/Sub 機能を提供するものがある．例えば Apache Cassandra と Apache ZooKeeper [6] を併用することで，Pub/Sub 基盤を独自に構築している事例がある [7]．また，ZooKeeper を用いた分散ロギングフレームワーク BookKeeper をベースとして，構築された Apache Hedwig [9] のように分散 Pub/Sub 基盤として提供されているものもある．ここで，ZooKeeper は分散合意形成のためのフレームワークだが，基本的には合意形成用のサーバが互いに直接通信することを想定しており，サーバ数が増えた時にスケーラビリティについては評価が必要である．提案構成の有効性を示すためにはこれらの環境との性能比較が必要である．また，提案環境は Web サーバとエージェントが同じ JavaVM 上で動作することを想定しているため，Web アプリケーションの開発言語が限定されるなど，開発の容易性等の面でも制約をもつ．さらに，フロントエンドとバックエンドにサーバを分割するケースと比較すると，フロントエンドサーバにおけるメモリ使用量が増加するため，サーバ構成におけるコストについても比較が必要になる．以上のように，エージェントのインタフェースの構築方法も含め，今後さらに提案構成の有効性を検証していく必要がある．

6. ま と め

本稿では，PIAX をバックエンドとして利用する Web アプリケーションの構築方法を提案し，提案する構成におけるエージェント移動のコストについて性能計測を行った．性能計測の結果，エージェント移動のコストはエージェントのシリアライズ化ならびに転送コストならびに属性のオーバレイネットワークへの再登録コストであることが確認できた．エージェント自体の転送コストに対し，オーバレイネットワークの再登録コストが比較的大きくなることが確認できた．今後，エージェント移動ならびに Publish/Subribe にかかるサーバの CPU 負荷やネットワーク負荷を計測し，具体的にエージェント移動がエージェント配置の最適化に適用可能なケースを明らかにし，既存の Pub/Sub 環境と比較していく必要がある．

謝　　辞

本研究の一部は，戦略的情報通信研究開発推進事業（SCOPE）ならびに国立情報学研究所との共同研究の助成を受けて実施したものである．また本研究の実施にあたってご助言をいただいた BBR 吉田 幹氏，大阪大学石 芳正氏に感謝の意を表す．

文　　献

[1] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, “LIPSIN: line speed publish/subscribe inter- networking,” Proc. of the ACM SIGCOMM 2009 conference on Data comunication (SIGCOMM ’09). Oct. 2009, pp.195- 206, DOI=10.1145/1592568.1592592 http://doi.acm.org/10.1145/1592568.1592592.
[2] Redis, http://redis.io/, 2014 年 1 月参照.
[3] 小西佑治, 吉田 幹, 竹内 亨, 寺西裕一, 春本 要, 下條真司, 単一ノードに複数キーを保持可能とする Skip Graph 拡張, 情報処理学会論文誌, Vol. 49, No. 9, pp. 3223-3233, 2008 年 9 月.
[4] Jetty, http://www.eclipse.org/jetty/, 2014 年 1 月参照.
[5] IETF RFC6455, ”The WebSocket Protocol”,http://tools.ietf.org/html/rfc6455, 2014 年 1 月参照.
[6] Apache ZooKeeper, http://zookeeper.apache.org/, 2014 年 1 月参照.
[7] Eventbrite Engineering Blog, ”Replayable Pub/Sub Queues with Cassandra and ZooKeeper, ” https://engineering.eventbrite.com/replayable-pubsub-queues-with-cassandra-and-zookeeper/, 2014 年 1 月参照
[8] VisualVM, http://visualvm.java.net, 2014 年 1 月参照
[9] Apache HedWig, http://wiki.apache.org/hadoop/HedWig, 2014 年 1 月参照.