情報処理学会論文誌　コンピューティングシステム　Vol. 4　No. 4　65–75 (Oct. 2011)

グラフ分割を用いた大規模2部グラフのデータストリーム処理

雁瀬　優[†1]　上野晃司[†1]　鈴村豊太郎[†1,†2]

近年，グラフ構造に対するマイニング技術が注目を集めている．従来の大規模グラフ処理の研究はデータを蓄積して実行するバッチ処理が中心となっているが，処理の中にはリアルタイムな処理が要求される分野も数多く存在し，データを蓄積せずに逐次に実行するデータストリーム処理による解決が求められている．本研究では大規模2部グラフに対するグラフ頂点間の関連性解析に対して焦点を合わせ，高速化手法を提案し，リアルタイム処理を実現した．本研究では，ソーシャルネットワークに代表されるグラフの持つヒューリスティック性の1つであるコミュニティ構造を利用して，いくつかのコミュニティにグラフを分割することで高速化を実現した．実データを用い4ノードで並列に実験した結果，グラフ分割数16においてグラフ分割を行わずに処理を行う場合と比較して線形以上の高速化を実現した．

Data Stream Processing for Large-scale Bipartite Graph Using Graph Partitioning

Masaru Ganse,[†1] Koji Ueno[†1]
and Toyotaro Suzumura[†1,†2]

In recent years, real-time data mining for large-scale time-evolving graphs is becoming a hot research topic. Most of the prior arts target relatively static graphs and also process them in store-and-process batch processing model. In this paper we propose a method of applying on-the-fly and real-time stream computing model to such dynamic graph analysis. To process large-scale graph streams on a cluster of nodes in a real-time and scalable fashion, we propose a method of dividing graph streams into several sub-graph streams by using the notion of "community structure" typically appeared in social networks and processing a set of divided streams with multiple compute nodes in parallel. Our experimental results demonstrate that our method achieves up to more linear times speedup with 16 partitions against no partitioning.

1. 背　　景

近年，スーパコンピュータの評価指標としてGraph500[1)]が策定されるなど，大規模グラフの処理が注目を集めている．これまでの大規模グラフ処理の研究は，データをある程度蓄積してから処理を実行するバッチ処理が中心であったが，大規模グラフ処理が必要とされる分野の中には，株式市場やマーケットサイトのリコメンデーションシステムなど，リアルタイムな処理が要求される分野も多数存在している．そういったリアルタイム処理を可能にするにはデータを蓄積することなく逐次に実行するデータストリーム処理を行う必要がある．

しかし，大規模グラフの処理にかかる計算時間は膨大で，単に流れてくる情報に対し，逐次実行をしただけではリアルタイム性を保持しつつデータストリーム処理を行うことができない．また，バッチ処理ではデータを蓄積してから1度だけ実行すればよいのに対し，データストリーム処理はエッジが更新されるたびにグラフ処理を行わなければならないため，エッジ情報の増加が計算時間の増加につながってしまう．

ソーシャルネットワークに代表される大規模グラフの持つ性質の1つとして，コミュニティ構造[2)]という特徴が存在している．コミュニティ構造とはエッジが密な集合どうしが疎なエッジでつながっている構造であり，分割して計算してもある程度の処理精度を保つことが知られている．本研究ではその性質を利用した高速解析手法を提案し，実装と評価を行った．

評価対象としては，コミュニティ性を持つソーシャルネットワークの1つ，匿名掲示板を選択し，Random Walk with Restart法（RWR法）[3),4)]による頂点間の関連性の測定を行った．その他のリアルタイム性が要求される2部グラフの関連性解析には，次のような例があげられる．

- 株式市場：2つの頂点群にユーザと株式を持ち，株式の売買をエッジとして扱う．関連性の解析を行うことで似た性質を持つ銘柄や，株式の不正売買を監視することができる．
- 市場調査：2つの頂点群に顧客と商品を持ち，商品の売買をエッジとして扱う．関連性解析を行うことで似た商品や，ユーザの嗜好を判断することができる．
- P2Pシステム：2つの頂点群にユーザとファイルを持ち，データのダウンロードやアッ

†1 東京工業大学
Tokyo Institute of Technology
†2 IBM 東京基礎研究所
IBM Research-Tokyo

プロードをエッジとして扱う．関連性を解析することでユーザ間の類似性を判断し，最も必要なファイルを持つユーザを特定することができる．

いずれの場合も，必要な解析は頂点間の関連性に集約されるため，今回の実装の結果を応用することができる．

以降の章については，2 章でデータストリーム処理について述べ，3 章では問題定義について述べる．4 章では提案手法について述べ，5 章で実装，6 章で評価，7 章で議論，8 章で関連研究，9 章でまとめと今後の展望について述べる．

2. データストリーム処理と System S

本論文では，グラフ構造に対してデータストリーム処理を実現するために，IBM Research の提供するデータストリーム処理系 System S を利用している．この章では，データストリーム処理に対する補足説明を行い，System S の実装について説明する．

2.1 データストリーム処理

データストリーム処理[5)–9)]とは，始点・終点という概念のない情報の列をストリームと呼び，このストリームを蓄積することなく逐次処理していくという新しい計算パラダイムである．バッチ処理と呼ばれる計算対象をすべてストレージに蓄積してから計算する従来の手法と違い，リアルタイムの応答が要求される場合や，時系列で前後するわずかなデータのみを参照すればよい計算や，全データの蓄積が物理的に困難な処理に適している．このような手法は音声や動画のストリーミングなど一部の処理では利用されていたが，データストリーム処理はこれを抽象・汎用化し，幅広い処理に対して適用できるよう洗練された処理系としてまとめられている点が従来とは異なっている．このような処理系を DSMS/DSPS（Data Stream Management/Processing System）と呼ぶが，その一例として MIT の Borealis[5)] や IBM Research の System S[6)–8)] などが存在し，ここ数年活発な研究がなされている．多くの DSMS がシングルノードでの実行を前提としているが，Borealis と System S は分散環境上で実行可能である．

2.2 System S と SPADE

System S[6)–8)] は，データフロー図から直感的に処理を記述できる SPADE[6)] という言語と，自動性能最適化機構を持つ SPADE コンパイラ，処理基盤である SPC[7)] によって構成される．SPADE は高級な宣言的言語で，処理対象であるストリームと，処理を行うオペレータの関係をデータフローとして記述するだけでデータストリーム処理を定義でき，ノード間やプロセス間の通信や，デーモンの立ち上げなどを意識することなくプログラミングが可能である．広範な処理に適用可能な汎用の組み込みオペレータを持つため，単純な処理ならば組み込みオペレータにパラメータを設定するだけで実装できる．汎用オペレータだけでは不十分な場合は，C++や Java を用いたユーザ定義の独自のオペレータや関数の作成もサポートされている．SPADE では，コンパイルや最適化を段階的に行うことで高度な最適化を施す．SODA[8)] では実行中のノード割当ての変更のような動的な最適化もサポートしており，処理全体の高速化が図られている．

SPADE では多様な組み込みオペレータのほかにユーザ定義オペレータ UDOP（User Defined Operator）を持ち，複雑なデータストリーム処理に対応することが可能となっている．UDOP は実際の処理以外の通信やストリームの管理部分が書かれたスケルトンコードを自動生成するため，ユーザは処理部分のみを C++や Java で記述すればよい．これにより，汎用オペレータでは表現できない複雑な処理や操作を実現でき，UDOP 自体も他のオペレータと同様にモジュール化されるため高度な柔軟性，再利用性の恩恵を受けることができる．SPADE のオペレータの詳細については論文 9) に詳細が記載されているため省略するが，SPADE は簡易な記述と高い柔軟性を兼ね備えており効率的なシステム開発が可能となっている．

3. 問題定義

本研究では Random Walk with Restart 法（RWR）[3),4)] を用いたグラフ頂点間の関連性のリアルタイム解析を目的として，グラフのヒューリスティック性の 1 つであるコミュニティ構造[2)] を用いた高速化手法を提案している．本章では，この高速化手法が適用できる前提条件と前提条件を満たす例について述べる．

3.1 前提条件

対象とするグラフはコミュニティ構造を持った 2 部グラフとする．コミュニティ構造を持ったグラフを用いる理由は，詳しくは後述の提案手法で説明するが，高速化手法を精度を保ちつつ実行するために必要なためである．2 部グラフを用いる理由としては，2 部グラフはグラフの構造的特徴として有用であるという点，2 部グラフの片方の頂点のみを分割の対象とすることで，エッジ構造を損なわずに分割を行うことを可能としている点（5 章参照）があげられる．データストリーム処理を行う都合上，グラフは時系列に従って一定の変化がみられることが望ましい．

次に，本研究でのデータストリーム処理は，単独で長時間にわたって解析するのではなく，あくまでバッチ処理とバッチ処理の間の今まで処理を行っていなかった期間の補完とし

て扱う．データストリーム処理はすべての面でバッチ処理より優れている手法ではなく，逐次処理が不可能という点を除けば，バッチ処理を行った方が計算資源を長時間 1 つの処理に使用することができるため，計算量が高く精度の高いアルゴリズムを使用できる．そのため，データストリーム処理を単独に処理を行うよりも，バッチ処理と並行して行った方が処理全体として効率が良い．

3.2　前提条件を満たす 2 部グラフの例

前提条件を満たす実データのグラフの例としては匿名掲示板 “2 ちゃんねる” があげられる．匿名掲示板には 1 つのテーマの集合体 “板”（例：ニュース板）とテーマの中のトピック “スレッド”（例：事件 A について），書き込みである “レスポンス” が存在し，ユーザは ID によって一定の期間（通常 1 日）自己の同一性が保証される．匿名掲示板の特徴としては，時系列データが特有に持つプライバシの問題の非考慮性，1 つのスレッドが終了した場合次のスレッド（例：事件 A について 2）にユーザが移動することに起因する時系列上のスレッド間の関係の明確性，トピックの種類の多様性に起因するトピックの種類ごとのコミュニティ構造の所持の 3 点があげられる．

匿名掲示板のデータは，2 つの頂点群としてユーザ ID とスレッドを，エッジとして “レスポンス” を持つ 2 部グラフとして扱うことができる．時系列上での変化が激しいグラフなので時系列上でのグラフの関連性の変化を比較する際には適している．“2 ちゃんねる” は匿名掲示板の例としてしばしば用いられており，“2 ちゃんねる” を評価対象として用いている論文の例として松村らによる論文 10) があげられる．実際に分割を行わずにスレッド間の関係性を測定した場合，同じトピックで立てられたスレッド間の関連性は他のスレッドと比較して高くなった．本論文では同じトピックで立てられたスレッド間の関連性が高いという匿名掲示板の性質を利用することで，グラフ分割による関連性の精度の変化を測定している．

4. 提 案 手 法

4.1　グラフのコミュニティ構造とグラフ分割

1 章でも説明したが，ソーシャルネットワークに代表される大規模グラフの持つ性質の 1 つとして，コミュニティ構造[2)] という特徴が存在している．コミュニティ構造とはエッジが密な集合どうしが疎なエッジでつながっている構造であり，分割して計算してもある程度の処理精度を保つことが知られている．本研究ではその性質を利用した高速解析手法を提案し，実装と評価を行った．本手法ではグラフ処理要求の頻度とグラフのサイズから，必要とされる処理速度を達成するグラフ分割数を適切に選び，各グラフに並列分散処理させることによって，リアルタイム処理を実現する．

4.2　グラフのコミュニティ構造と既存研究

コミュニティ構造が認められるグラフの例として，インターネット，疫学，論文の引用と共著などが存在する．グラフのヒューリスティック性としてコミュニティ構造を利用した論文[2),4),11)–14)] は多数存在しており，本論文と同様にコミュニティ構造を持ったグラフを分割した際の 2 部グラフの関連性解析の精度に関する論文 14)（8 章参照）も存在する．この研究は本論文と同様にグラフ分割のアルゴリズムに METIS [15),16)] を用いている．ただし，この研究はあくまでグラフ分割における 2 部グラフ関連性解析をバッチ的に実行したものであり，データストリーム処理を行った場合での時系列上での精度，性能を測る必要がある．

4.3　グラフ分割と高速化

この節では，グラフ分割による高速化について言及する．関連性解析に限らずグラフ解析の計算量，計算時間は要素数に依存している．たとえば，要素数の二乗の計算量がかかる計算が存在したとして，扱う要素が半分になったとすれば必要な計算時間は 4 分の 1 となる．

これを式で表すと次のようになる．要素数 n のグラフに対して計算量 $O(n^t)$ が必要な演算があった場合，必要な計算時間は $c * n^t$（ただし，c は固定値で，計算環境によって異なる）この演算を，グラフ分割数 k によって分割して行うと，必要な計算時間は $c * (n/k)^t$ となる．この式は容易に式変形でき，$c * n^t * (1/k)^t$ となる．つまり，グラフ分割を行うことで k^t の高速化が可能となる．今回の実装ではグラフ解析に $O(n^2)$ の計算量がかかるため，グラフ分割数により k^2 の高速化を得ることができる．また，扱う要素が同じサブグラフに属していなければ，分散して処理を実行することが可能となるため，さらなる高速化が得られる．

5. 実　　装

本論文では，2 部グラフの関連性解析などのグラフ処理をグラフ分割によりリアルタイムで行うことができるシステムを実装した．本章では，実装の詳細について述べる．

5.1　システムの要件

グラフ分割において精度を保つには，より正確にコミュニティ構造を検出する必要がある．3 章で述べたとおり，本システムはグラフのデータストリーム処理をバッチ処理と並行して行うことを前提としている．過去の情報をもとにバッチ処理で分割を計算しておくことで，正確なコミュニティ構造の検出が可能である．そこで，過去の情報をもとに行ったバッチ処

理の結果を読み込み，ストリーム処理に反映させることができるようにする必要がある．

しかし，バッチ処理の結果だけでは，過去の情報にある頂点のことしか知ることができないため，ストリーム処理中に来た新しい頂点をどのようにサブグラフに割り当てるかが問題となる．ストリーム処理中に来た新規頂点の，動的な追加を処理する必要がある．

また，ストリーム処理の過程で関連性が変化していくため，当然コミュニティ構造が変化する可能性がある．コミュニティ構造が変化した場合，サブグラフをコミュニティ構造に合わせて再構成しなければ精度が下がっていくことになる．よって，コミュニティ構造の変化を検出し，サブグラフを再構成する必要がある（このサブグラフの再構成は，頂点の移動と見ることができるため以下，マイグレーションと呼ぶ）．

5.2　システムの全体像

実装には，データストリーム処理系には System S を，Split オペレータ内（**図 1** (4) 参照）でグラフ分割を行うコンポーネントとして METIS [15] を，グラフ処理 UDOP 内（図 1 (5) 参照）で 2 部グラフの関連性を求めるコンポーネントとして FSU [3]（Fast-Single-Update）を用いた．System S の詳細と実装に関しては 2 章を，METIS，FSU のアルゴリズムに関してはそれぞれ 5.4，5.5 節を参照していただきたい．

このシステムのデータフローは図 1 のようになっている．

(1)　まず，過去の情報をもとに行ったバッチ処理の結果として METIS の分割結果を与える．METIS は分割元のグラフを受け取り，処理の結果として分割結果（各頂点の属するグラフ ID）を返す．METIS によって取得した分割結果を用いて，Split オペレータが頂点，辺のデータを各グラフ処理 UDOP に振り分ける．

ここまではバッチ処理で行うことであり，次に実際のストリーム上での処理の流れについて解説する．

(2)　まず，Source オペレータが取得してきたエッジストリームを Split オペレータによって各グラフ処理 UDOP に分割する．エッジは 2 つの端点と重みで表現される．

(3)　Split オペレータは，新規頂点の動的な追加や，コミュニティ構造の変化の検出，マイグレーションに必要なデータの送信なども行う．

(4)　各グラフ処理 UDOP 内では分割された要素をもとに FSU が行われる．

(5)　FSU の処理の結果として，グラフ更新結果が Sink オペレータにその結果が送られる．

グラフの分割数は SPADE コンパイル時に任意に決定することができる．

処理結果は，各グラフ処理 UDOP が保持し，このデータはストリーム処理によりいつでも最新のデータを処理した結果になっている．本論文では精度評価のためエッジ処理ごとに結果を出力する実装としたが，実運用ではクライアントからの要求に応じて，データを取り出すことになる．

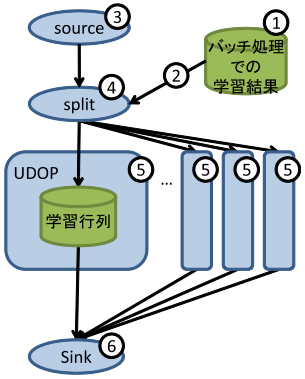

図 1　実装のフロー図
Fig. 1　Flow of the proposed approach.

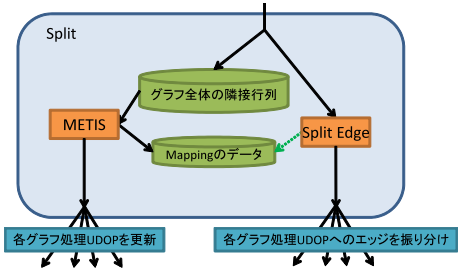

図 2　Split オペレータの実装フロー
Fig. 2　Flow of the Split operator.

5.3　Split オペレータ

この章では，前節で紹介した Split オペレータの内部実装について説明する．データフローは**図 2** のようになっている．処理の中核をなすのはグラフの動的変更を司る METIS 処理，エッジの割当てを司る Split Edge 処理である．Source オペレータから来たエッジ情報は，まずグラフ全体の隣接行列に反映される．Split Edge 処理によって，頂点の分割情報である Mapping のデータから各グラフ処理 UDOP へエッジの振り分けを行っている．以下，Split オペレータの各処理について説明する．

5.3.1　頂点とサブグラフの Mapping

Mapping は頂点 ID とその頂点の属するサブグラフ ID の対応が書かれたデータである．3 章で述べたとおり，本システムは 2 部グラフの片方の頂点群のみを分割の対象としているため，Mapping には分割対象の頂点群とサブグラフの対応が書かれている．エッジは分割対象側の頂点の ID を見て，各グラフ処理 UDOP に振り分けられる．バッチ処理により精度の高い分割が計算されると，そのデータを読み取り Mapping を更新する．実装では，分割情報が記述されたファイルを読み込み Mapping のデータを更新している．また，Split オペレータ内で新規頂点の動的な追加やサブグラフ再構成にともない頂点とサブグラフの対応を変更する場合も更新される．Mapping のデータは，頂点 ID をインデックスとしてサ

ブグラフ ID が引けるような配列として実装されている．入力データのエッジストリームは頂点 ID が 0 から順番に振られるように前処理されているため，この配列は頂点の個数と同じ長さになっている．

5.3.2　新規頂点の動的な追加

入力エッジの 2 つの端点のうち分割対象側頂点が Mapping のデータにない場合，新規頂点となる．ストリーム処理中に来た新規頂点をどのようにサブグラフに割り当てるかが問題となるが，これには次のような方法を用いた．新しい頂点を含むエッジは，どのサブグラフに属するかを決定できるだけの十分な情報が来るまで蓄積する．その間，エッジ情報はグラフ全体の隣接行列には反映するが，グラフ処理 UDOP には振り分けない．この蓄積の回数はグラフの特徴によって様々であるため，ヒューリスティックなパラメータによって決定する．蓄積されたエッジ情報を METIS により解析し，適したサブグラフに割り当て，Mapping のデータを更新し，蓄積されたエッジ情報をグラフ処理 UDOP に渡す．これにより新規頂点の追加に対応する．

疎行列表現[17] には，実行する処理によって最適なデータ構造が異なるが，グラフ全体の隣接行列はインクリメンタルな要素の追加が高速なハッシュマップを利用した Dictionary of Keys 形式（DOK 形式）で保持している．DOK 形式とは，要素の行や列の番号をキーとして要素の値を引く辞書を利用した疎行列の表現形式である．METIS へ入力させる場合，メモリ使用量の小さい Compressed Sparse Row 形式（CSR 形式）[17] に変換する必要があるが，この変換は全要素を 2 回走査するだけで可能である．1 回目の走査で必要スペースを計算してメモリを確保し，2 回目の走査でデータをコピーする．

5.3.3　コミュニティ構造の変化への対応

ストリーム処理中のコミュニティ構造の変化に対応するため，定期的にグラフ全体を METIS により解析する．METIS による解析結果から，新しいサブグラフと現在のサブグラフの差分に対して，マイグレーションを行う．マイグレーションを行う際は Split を停止させ，Mapping データの更新と各 UDOP オペレータの更新を行ってから，Split を再開するという手順をとっている．各 UDOP オペレータへは，差分の更新情報をストリームで送信する．送信する情報は削除する頂点のリストと追加する頂点からなるサブグラフである．サブグラフの送信には，疎行列の表現形式としてデータ量の小さい CSR 形式を利用した．エッジはグラフ全体の隣接行列から必要な情報を取り出して，ストリームで送信している．

5.3.4　METIS ライブラリによる分割

新規頂点のサブグラフへの割当てや，サブグラフの再構成では，METIS [15),16)] による解析結果を使用する．METIS の出力は，入力グラフの全頂点の分割結果である．METIS は 2 部グラフの全頂点を分割するが，本システムでは 2 部グラフの片方の頂点しか分割しないため，METIS の分割結果の，本システム分割対象側頂点の分割データのみを使用し，分割対象でない頂点の分割は無視する．

```
入力は，サブグラフマッピング m，METIS の分割結果 p
出力は，サブグラフと METIS の分割結果におけるグラフ ID のマッピング subm
cm は n × n の行列 {n は分割数 }
 1.for i = 1 to n
 2. for j = 1 to n
 3.  cm(i,j) = 0
 4.for i = 1 to 頂点数
 5. cm(m(i),p(i)) = cm(m(i),p(i)) + 1
scm は各要素ごとに x,y,c の 3 つの属性を持つ配列である
 6.for i = 1 to n
 7. for j = 1 to n
 8.  scm(i + j × n) の (x,y,c) = (i,j,cm(i,j))
 9.scm を属性 c の大きい順に並べ替える
10.for i = 1 to n × n
11. if scm(i).x と scm(i).y はどちらも対応が決定していない
12.  subm(scm(i).x) = scm(i).y
13.  サブグラフ scm(i).x とグラフ ID scm(i).y を対応決定済みにする
```

図 3　METIS の分割結果におけるグラフ ID とサブグラフのマッピング決定アルゴリズム
Fig. 3　Algorithm for mapping subgraphs to graph ids given by METIS.

METIS はグラフが少し変化すると，分割結果のグラフ ID がランダムに変わってしまうので，METIS 実行後，METIS による分割結果におけるグラフ ID と，現在のサブグラフの対応を計算しなければならない．METIS の分割結果と現在のサブグラフとで，類似度を計算して対応を決定するが，そのアルゴリズムを**図 3** に示す．このアルゴリズムは，各頂点の METIS による分割結果と，現在のサブグラフマッピングを入力とし，METIS の分割結果におけるグラフ ID と現在のサブグラフのグラフ ID との対応を出力する．アルゴリズム中の 4〜5 を計算すると，cm の i 行 j 列は，“現在のサブグラフ i にある頂点のうち METIS の分割結果におけるグラフ ID が j の頂点” の数になる．6〜8 は，9 の並べ替えを行うために，行列の値と，行・列の番号をセットにしたデータに変換している．9 で，頂点数の大きい順に並べ替え，10〜13 で頂点数の大きい順に対応を選ぶ．これにより，類似度の大きいサブグラフどうしが対応付けられる．

新規頂点のサブグラフへの割当てでは，METIS に新規頂点を含む 2 部グラフ全体を入力

させ，分割を計算し，新規頂点の割り当てられたサブグラフに，頂点を割り当てる．

5.3.5　3 つの Split 処理

Split オペレータは，以下の Static-Split 処理，SemiDynamic-Split 処理，Dynamic-Split 処理の 3 つの処理に対応している．

- Static-Split 処理は分割対象の頂点群側の新規頂点の追加を行わずに，バッチ処理の時点で与えられた Mapping をもとに計算を行う手法である．この手法では前処理段階でバッチ的に取得した Mapping のデータを動的に変更せずに，Split 処理を行っている．後述する予備実験でも用いられているこの手法は，分割対象の頂点群に新規頂点の追加がなく，コミュニティ構造の変化が少ない場合に用いる処理である．この手法ではデータストリーム処理中には METIS は実行しないため，計算量が少ない．
- SemiDynamic-Split 処理は，新規頂点の動的な追加は行うが，サブグラフの再構成は行わない処理である．分割対象の頂点側の新規頂点追加があるが，コミュニティ構造の変化が少ない場合に用いる処理である．サブグラフの再構成は処理時間コストが大きいため，必要がない場合は再構成しないほうが望ましい．
- Dynamic-Split 処理は，新規頂点の追加に加えて，定期的に METIS を起動し，サブグラフの再構成も行う処理である．この処理は，時系列で変化の大きいグラフにも対応できる．

5.4　グラフ分割ライブラリ METIS

グラフ分割ライブラリ METIS[15] ではエッジカットを最小にしつつグラフを同じ大きさの k 個のサブグラフに分割するアルゴリズム Multi Level Recursive Bisection 法（MLRB）[16] をグラフの縮小復元を用いて効率的に行っている．MLRB 法の計算量を下げるために METIS では分割を 3 フェーズに分け，第 1 フェーズではグラフを縮小し頂点数を下げ，第 2 フェーズで頂点数を下げたグラフに対し MLRB を行い，第 3 フェーズで分割の結果を補正しつつ縮小したグラフを元に戻している．第 1 フェーズのグラフの縮小に関しては，縮小したグラフに対して分割を行っても，元のグラフに対して分割を行う場合と差異が出ないように縮小を行うためのアルゴリズムをいくつかあげているが，今回はそのアルゴリズムに関しての説明は割愛する．METIS のアルゴリズム，MLRB 法の詳細に関しては参考文献 15)，16) を参照していただきたい．グラフの縮小と復元を行うことによって，MLRB 法に対して計算量を下げており，METIS の計算量はエッジ数 E に対して MLRB 法の計算量 $O(|E|logk)$ に対して $O(|E|)$ に削減している．また，METIS ライブラリには，MPI を使った分散処理が可能な ParMETIS ライブラリがあり，大規模なグラフに対応できる．

5.5　グラフ処理 UDOP

グラフ処理 UDOP では，Split オペレータで決定された各サブグラフに対してグラフ処理を行っている．グラフ処理 UDOP 内ではグラフ処理を行うために関連性解析アルゴリズム Fast-Single-Update（FSU）[3] を用いている．

FSU は，グラフのランダムウォーク法 Random Walk with Restart（RWR 法）を用いた 2 部グラフの関連性解析アルゴリズム BB_LIN[4] を差分更新可能とすることによって，毎回すべての要素を計算し直している BB_LIN と比較して計算量を減らす手法である．計算量を比較すると，BB_LIN が 2 部グラフの 2 つの頂点群のうち要素が少ない頂点集合（L と定義する）の要素数の 3 乗の計算量がかかるのに対し，FSU では計算量は L の 2 乗となり，計算量を劇的に減らすことに成功している．これは大規模 2 部グラフのバッチ処理の実行時間を低減させるという点では非常に大きな意味を持つが，L が増加すると各処理の計算量が 2 乗に増加するという欠点がある．このアルゴリズムを逐次に実行することによってデータストリーム処理を実行しようとするとき，ストリーム上を流れるエッジ情報の到着頻度内に処理を終えることができないとリアルタイム性を保持できなくなってしまうため問題となる．そのため，本論文ではグラフのコミュニティ構造を利用したグラフ分割による高速化を用いてこの問題を解決している．

マイグレーション実行時は，サブグラフからの頂点の削除が発生する．FSU[3] には頂点の削除方法が定義されていないが，頂点の削除は行列から値を削除することで対応した．つまり，頂点の削除後，行列には削除されなかった頂点のデータだけをそのまま残している．

グラフ処理 UDOP で保持するサブグラフは，行方向や列方向の連続アクセスやインクリメンタルな要素の追加が高速な木構造を利用した DOK 形式を使用している．ハッシュマップを利用しないのは，FSU では 1 つの行（または列）のデータを取り出す操作が必要だからである．もし，ハッシュマップを使うと，1 つの行（または列）のデータを取り出すのに全要素を列挙しなければならなくなるため計算量が大きくなる．

FSU では L×L の密行列を学習行列として使用する．この行列の i 行 j 列は，i に対応する頂点と j に対応する頂点の関連性を表す．Split オペレータのグラフ全体の隣接行列における頂点 ID と，学習行列上の位置の対応は，Split オペレータの Mapping データと同じように配列によるマッピングで保持している．

6. 評　　価

この章では前章での実装に対する評価を行う．

6.1 実験環境

測定にはノードを 5 台使用した．環境は全ノード共通で，CPU は AMD Phenom 9850（2.5 GHz，4 コア），メモリは 8 GB，OS は CentOS 5.4，ソフトウェアは，InfoSphere Streams 1.2.0（System S），gcc 4.1.2，METIS 4.0，行列演算には，ublas（boost 1.33.1）を使用した．ネットワーク環境はそれぞれ 1 Gb Ethernet で接続する．実装のフロー図での Source と Split を行うノードに 1 台（4 コア），実装フロー図での UDOP（今回は FSU）を行うノードには 4 台（16 コア）割り当てた．実験での物理コアと UDOP オペレータとのマッピングはラウンドロビン法を用いている．gcc のコンパイルはすべて最適化オプション “-O3” で行った．

6.2 対象とするデータ

3 章で紹介した匿名掲示板 “2 ちゃんねる” を対象とした．匿名掲示板のデータは，2 つの頂点群としてユーザ ID とスレッドを，エッジとして “レスポンス” を持つ 2 部グラフとして扱うことができる．今回用いた 2 部グラフは “2 ちゃんねる” 内の “板”，“ニュース速報版” 内の 2010/12/14 0:00:00 から 2010/12/15 23:59:13 までのデータを用いた．グラフの規模としては，ユーザ ID 数 25,532 個，スレッド数 492 個，書き込み数 85,656 回となり，書き込みの頻度としては約 2 秒に 1 回となる．このグラフを大規模 2 部グラフとして扱うには書き込み頻度，グラフ要素数ともに少ないが時系列上での変化が激しいグラフなので時系列上での精度を比較する際には適している．

6.3 予備実験

匿名掲示板のデータに対して実適応を行う前に，予備実験としてすべてのエッジを既知として，事前にグラフ構造の分割をオフラインで行っておき，ストリーム処理の際には Mapping を変更しないで処理する Static-Split 処理を用いて実験を行った．この実験では，つねに最適なグラフ割当てが特定できる場合のパフォーマンスについて検証している．たとえば，株式市場ではユーザの動的変化は激しいが株式の変動は稀である．このような場合，この実装を用いれば同様の高速化を得ることができる．

図 4 ではグラフ分割後の計算時間を，分割なしでの計算時間で割ることで，計算量高速化の比率をグラフとして表している．処理が分割した各グラフに，均一に振り分けられた場合，各グラフは並列に処理を実行できるため，1 処理にかかる平均計算時間を減少させることができる．グラフ分割による高速化比率は図 4 のように変化し，グラフ分割数 16 においての計算時間は 4 ノード 16 コアを用いて約 206 倍の高速化を実現した．

次に精度について検証する．2 ちゃんねる上には，すべての頂点（スレッド）と関連性がまったくない頂点（スレッド）もしばしば存在する．そこで，精度評価としてグラフ分割なしでの処理の結果，関連性解析の学習結果のスコアが平均以上である頂点に対して，グラフ分割を行った場合に最も重要度の高い要素が一致した割合を比較した．つまり，全スレッドで平均以上に関連性のあるスレッドのみを評価の対象とし，グラフ分割を行ったときに最も関連性が高いと判断される要素に変化が生じるかを実験した．エッジ数が少ないうちは，解析する対象頂点数が少ないため 30,000 エッジほど測定しないと正確な精度が出ておらず，全体を通して分割数 2，分割数 4 ではまったく同じ結果となった．結果としては**図 5** のように変化し，分割数 16 においても 7 割近くの精度を保った．

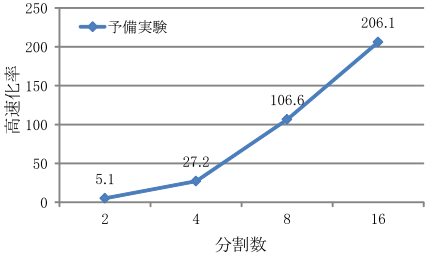

図 4　グラフ分割による計算量高速化比率
Fig. 4　Speedup with graph partitions.

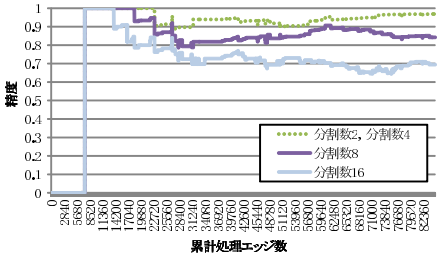

図 5　グラフ分割による精度変化
Fig. 5　Precision with graph partitions.

6.4 本実験

本実験として，どのエッジがどのサブグラフに属するのかの値を与えずに実験を行った．実験には，SemiDynamic-Split 処理，Dynamic-Split 処理を用いた．SemiDynamic-Split 処理で新規頂点の追加時に一時的に蓄積するエッジの数は予備実験の結果，統計をとると有意な分割指標が得られるまでは 30 回程度のエッジ情報が必要であったことから 30 個とした．Dynamic-Split 処理では入力エッジ 10,000 ごとに METIS の再計算を行っている．METIS の再計算を行うパラメータは数が大きすぎると METIS による再計算が行われないため，精度が**図 6** のように低下し，数が少なすぎると METIS による処理がオーバヘッドとなり**図 7** のように実行時間が増加してしまう．今回は 3 例（100，10,000，50,000）について測定し，それぞれの精度の高低と計算時間の増減の関係を図 6，図 7 のように測定し，METIS による再計算を行う入力エッジのタイミングとして計算の精度の高さと計算時間の少なさが両立している 10,000 に決定した．精度の高低と計算時間の増減には本実験と同じデータを使用し，すべてのエッジを処理したときの精度と計算時間を比較している．この実

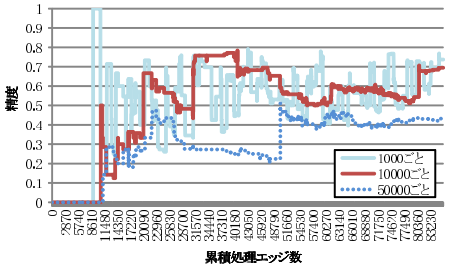

図 6　入力エッジによる精度変化
Fig. 6　Precision by input edges parameter.

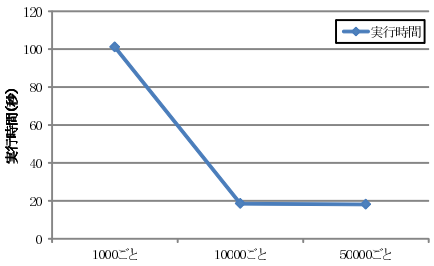

図 7　入力エッジによる実行時間
Fig. 7　Computation time by input edges parameter.

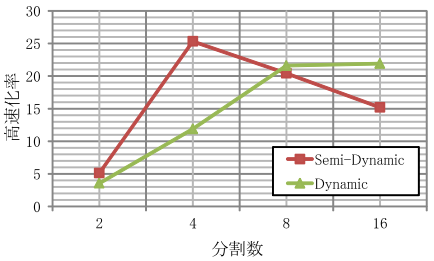

図 8　グラフ分割による計算量高速化比率
Fig. 8　Speedup with graph partitions.

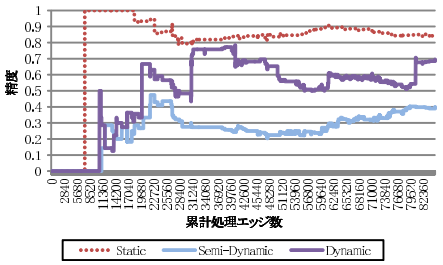

図 9　グラフ分割による精度変化
Fig. 9　Precision with graph partitions.

験は予備実験での実装と異なり，実際の 2 ちゃんねるのように動的に新規頂点が追加されるアプリケーションに対しても実適用することが可能である．

図 8 ではグラフ分割後の計算時間を，分割なしでの計算時間で割ることで，計算量高速化の比率をグラフとして表している．グラフ分割により，高速化比率は図 8 のように変化し，SemiDynamic-Split 処理においてはグラフ分割数 16 においての計算時間は約 15.2 倍の高速化を，Dynamic-Split 処理ではグラフ分割数 16 において約 21.9 倍の高速化を実現した．

SemiDynamic-Split 処理を使用した実装では，頂点の追加タイミングでストリームを止めてしまうため，速度低下がみられた．この実装の特徴としては，METIS による分割では少数のエッジ追加がサブグラフの大部分を変更してしまうため，大幅な精度の低下がみられた．少数のエッジ追加がサブグラフの大部分を変更しない代替のアルゴリズムを用いれば改善される可能性がある．一方で，Dynamic-Split 処理は入力エッジを基準にマイグレーションを行っているため SemiDynamic-Split 処理ほどの速度低下はみられなかった．いずれにしても，METIS によるマイグレーションがボトルネックとなり速度低下が発生しているが，Dynamic-Split 処理を使用した実装ではグラフ分割数 16 において線形以上の高速化を保つことに成功している．この実装の特徴としては，SemiDynamic-Split 処理と同様に，METIS による分割で少数のエッジ追加がサブグラフの大部分を変更してしまうため，計算速度の低下がみられた．少数のエッジ追加がサブグラフの大部分を変更しない代替のアルゴリズムを用いれば改善される可能性がある．

両処理では，計算時間が分割数を増やすことにより下がってしまっているが，その原因について補足する．METIS と Split Edge の処理はマルチスレッドで計算を行っており，その結果から最も類似しているサブグラフを割り当てる．その際の演算手順としては

(1)　新規頂点追加命令 or マイグレーション命令
(2)　ストリームを止める
(3)　METIS 処理の起動
(4)　割り当てるサブグラフの決定
(5)　Mapping の更新
(6)　ストリームを流す

となっているが，METIS 処理がオーバヘッドとなってしまっていて，M ETIS での処理にかかる時間以上の高速化をグラフ分割で得ることができない．その問題を解消するためにはフローの順番を 1，3，4，2，5，6 とし，METIS によるストリームの停止を最小限にする必要がある．しかし，ストリームの停止を最小限にした実装の場合，実験結果がスレッド処理特有の性質である非同期性により，処理時間によって結果の再現性がない．実験では結果を高速に得るため，すべてのエッジ情報を時系列と同タイミングに 2 日分のデータを 2 日かけて実験という処理はしておらず，すべてのエッジ情報を時系列の順番にエッジ間の猶予なく即時に処理している．その結果，分割数によっては，METIS による処理が終わるまでにほぼすべてのエッジ情報がグラフ処理を実行してしまうため，実験結果が本来の実処理上での結果と大きく異なってしまう．実験で精度を測るためには今回用いた実装の形式は必要な措置であるが，実処理ではストリームの停止を最小限にした実装を用いればこのオーバヘッドは実際にはかかることはない．

次に精度について検証する．**図 9** では，グラフの分割数を 8 とし Static-Split 処理，SemiDynamic-Split 処理，Dynamic-Split 処理について精度を比較している．SemiDynamic-Split 処理を使用した実装では精度の低下が著しいが，Dynamic-Split 処理では分割なしで

の結果と比較しておおむね 6 割程度の精度を保っている．METIS は逐次処理，マイグレーションなどを考慮していないアルゴリズムであるため，改良すれば予備実験と同等の高速化と精度を両立させることができる余地がある．実際に実験の結果として，Dynamic-Split 処理についてはマイグレーション実行直後の精度は，予備実験と同様の値まで上昇している．

7. 議　　論

実験から得られた知見としては，データストリーム処理は単体で使うと時間経過とともに精度の低下がみられるため，バッチ処理とバッチ処理の間の従来処理を行っていない期間の補完を行うための方法として用いた方が効果が高い．そういった観点から考察すると，グラフ分割による精度低下はグラフ全体の構造が時間とともに急激に急激に大きく変化しなければ次のバッチ処理によって補正されるため起こらない．一方で，グラフ全体の構造が急激に変化するようなグラフも当然存在している．本章では時間経過による関連性の変化を考慮する大規模 2 部グラフ処理に対する議論を行う．

7.1　時間経過による関連性の変化の例

時間経過による関連性の変化の例としては，周期性を持った時系列データがあげられる．時系列データが周期性を持つ場合，既存のバッチ処理系では検出できなかった関連性の変化が検出できる可能性がある．株式市場を例にとると，市場の開始時と終了時ではユーザがとりうる行動は変化する．今までのバッチ的な解析では 1 日を通しての関連性しか測定できなかったが，データストリーム処理を行うことで，関連性の変化を時々刻々測定することができる．

7.2　時間経過による関連性の変化への考慮

時間経過による関連性の変化を考慮するうえで問題なのは，解析の方法である．時系列とともに関連性が急激に変化しなければ，これまでのバッチ的な解析を逐次に行うだけでも解析は可能であるが，急激な時系列の変化が起こるグラフに対してバッチ的な解析を行っても，過去のデータが現在のデータに対し干渉するため問題となる．よって，時間経過による関連性の変化を考慮するには，データストリーム処理に適したアルゴリズムを考案する必要があるだろう．時間経過による関連性の変化を考慮する例としてはスライディングウィンドウ[18]という手法がある．この手法は過去すべてのデータを処理対象にせず，保持する期間，保持する上限を指定して直近のデータのみに処理を行うという手法である．この方法を用いれば過去のデータが現在のデータに対し干渉するという問題点は解消できるが，反面，過去のデータを利用することができなくなってしまう．データの古さに応じてデータの持つ情報の重要度を下げることで対応するアルゴリズム[18]も存在するが，これらの研究はいまだ発展途上である．

次に，グラフ全体の構造が急激に変化するようなグラフに対しての本システムの拡張性について述べる．グラフ全体の関連性が大きく変わった場合サブグラフ間でマイグレーションが必要となる．現状の実装では，入力エッジごとにマイグレーションを行っているが，グラフ全体の構造が急激に変化するようなグラフに対しては，グラフ全体の関連性が大きく変わったかを判定しマイグレーションを行う必要がある．現状の仕様では外部のシステムにマイグレーションの実行タイミングについての判断基準を委ねることができるため，タイミングを任意に決定することが可能である．

8. 関 連 研 究

8.1　既存の大規模グラフ処理系

既存の大規模グラフ処理系としては Pregel[19]，PEGASUS[20] が有名であるが，いずれもグラフデータをすべて蓄積してから処理を行うバッチ処理系であり，データストリーム処理を行っているグラフ処理系は存在していない．両者の処理系は，ともにすべてのグラフ領域に対して計算を行っているため逐次処理を考慮しておらず，その点で本研究とは異なっている．

Pregel ではメッセージパッシングモデルに基づいたグラフ分析モデルを提唱している．Pregel では，各頂点が他の頂点から伝えられた情報をもとにローカルな計算を行い，計算結果の情報を伝え合うことで計算を反復し，すべての頂点が終了条件を満たしたときに処理を終了する．すべての計算において最適な計算を行うことはできないが，汎用的に用いることのできるグラフ処理系である．

PEGASUS では GIM-V（Generalized Iterative Matrix Vector Multiplication）モデルという計算モデルを提唱し，MapReduce を用いてグラフの解析を行っている．GIM-V モデルとはすべての計算を行列とベクトルの積に抽象化し，反復して計算を行うことで処理を行うモデルである．GIM-V モデルに適応するグラフ処理しか行えないが，GIM-V モデルに適応するグラフ処理に対しては Pregel よりも最適化された計算を行うことができる．GIM-V モデルが処理可能としている処理の例としては PageRank，Random Walk with Restart，グラフ直径問題，グラフ連結成分問題が存在している．

8.2　関連性解析の高速化

関連性解析とはネットワークの構造から頂点間の関連性を解析する手法である．PageRank

法（PR 法）や Randam Walk with Restart 法（RWR 法）などが代表的である．逐次に PR を行う方法，逐次に RWR 法を行うアルゴリズムとしては論文 3)，21)，22) のような例がある．

論文 21)，22) で紹介されているのはインクリメンタル PR 法である．Desikan らによる研究[21] では，有向グラフを用いて PR 法を計算する際，エッジの追加が十分に小さければ再計算に必要な領域は少なくて済むという性質を利用して，PR 法の高速な逐次実行を可能としている．この手法は，精度を下げることなく PR 法を計算することができるが，並列分散処理を行うことはできず，データレートに応じて高速化の度合いを変更することができない．山田らによる研究[22] では，PR を計算する際に，反復して計算する回数を減らすことでリアルタイムに PR の結果を取得している．この手法では，データレートに応じて高速化の度合いを変化させることができるが並列分散処理を行うことはできない．

Tong らの論文[3] では Fast-Single-Update 法（FSU 法）による逐次 RWR 法が紹介されている．この方法では，エッジ追加の結果，解析結果に変更が起きる箇所のみを再計算することで計算量を減らしている．本研究ではグラフ解析部分にこの FSU 法のアルゴリズムを用いている．

Sun らによる論文[14] では，グラフ分割を用いた関連性解析の高速化を行っている．この論文ではあくまでバッチ処理の高速化としてグラフ分割を行っているが，本研究ではそれをデータストリーム処理に適用している点で異なる．

9. まとめと今後の展望

実データを用い 4 ノード 16 コアで並列に処理した結果として，グラフ分割数 16 において，推定までの処理回数を増やすことなく，分割前と比較して処理時間を線形以上に減らすことに成功した．グラフ要素の分割による計算量の低減に加え，サブグラフが並列に処理を実行することができるため線形以上に処理を高速化することが可能である．

今回の実装ではいったんストリーム処理を止めないとグラフを何分割するのかの決定が行えないため急激なデータレートの変動を考慮していない設計となっている．そのため，ストリーム処理中に動的にグラフ分散数を決定するシステムの構築が必要である．また，今回の実装ではグラフ解析として関連性解析処理中心の実装となっているため，他のアプリケーションにも適応できる汎用的な大規模グラフのデータストリーム処理系への拡張などが必要である．その際には，METIS の代替として少数のエッジ追加がサブグラフの大部分を変更しないグラフ分割アルゴリズムの導入が望まれる．

参 考 文 献

1) Graph500, Graph 500 Steering Committee (online), available from ⟨http://www.graph500.org/⟩ (accessed 2010-12-15).
2) Aggarwal, C.C. and Yu, P.S.: Online Analysis of Community Evolution in Data Streams, *Proc. SIAM International Data Mining Conference* (*SDM 2005*) (2005).
3) Tong, H., Papadimitriou, S., Yu, P.S. and Faloutsos, C.: *Proximity Tracking on Time-Evolving Bipartite Graphs* (2008).
4) Tong, H., Faloutsos, C. and Pan, J.-Y.: Fast Random Walk with Restart and Its Applications, *ICDM '06: Proc. 6th International Conference on Data Mining*, pp.613–622, Washington, DC, USA, IEEE Computer Society (2006).
5) Abadi, D.J., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J.H., Lindner, W., Maskey, A.S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S.: The Design of the Borealis Stream Processing Engine, *2nd Biennial Conference on Innovative Data Systems Research* (*CIDR'05*), pp.277–289 (2005).
6) Gedik, B., Andrade, H., Wu, K.L., Yu, P.S. and Doo, M.: SPADE: the system s declarative stream processing engine, *Proc. 2008 ACM SIGMOD international conference on Management of data*, pp.1123–1134, New York, NY, USA, ACM (2008).
7) Amini, L., Andrade, H., Bhagwan, R., Eskesen, F., King, R., Selo, P., Park, Y. and Venkatramani, C.: SPC: A distributed, scalable platform for data mining, *Proc. 4th international workshop on Data mining standards, services and platforms*, pp.27–37, New York, NY, USA, ACM (2006).
8) Wolf, J., Bansal, N., Hildrum, K., Parekh, S., Rajan, D., Wagle, R., Wu, K.-L. and Fleischer, L.: SODA: An Optimizing Scheduler for Large-Scale Stream-Based Distributed Computer Systems, *Middleware 2008*, Issarny, V. and Schantz, R. (Eds.), Lecture Notes in Computer Science, Vol.5346, pp.306–325, Springer Berlin/Heidelberg (2008).
9) 松浦紘也，雁瀬　優，鈴村豊太郎：データストリーム処理系 System S と Hadoop の統合実行環境，第 22 回コンピュータシステム・シンポジウム (2010).
10) 松村真宏，三浦麻子，芝内康文，大澤幸生，石塚　満：2 ちゃんねるが盛り上がるダイナミズム，情報処理学会論文誌 (2004).
11) Newman, M.E. and Girvan, M.: Finding and evaluating community structure in networks, *Physical review. E, Statistical, nonlinear, and soft matter physics*, Vol.69 (2004).
12) Flake, G., Lawrence, S. and Giles, C.L.: Efficient Identification of Web Communities, *6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, MA, pp.150–160 (2000).
13) Newman, M.E.J.: *Fast algolythm for detecting community structure in networks*

(2003).
14) Sun, J., Qu, H., Chakrabarti, D. and Faloutsos, C.: Neighborhood Formation and Anomaly Detection in Bipartite Graphs, *ICDM '05: Proc. 5th IEEE International Conference on Data Mining*, pp.418–425, Washington, DC, USA, IEEE Computer Society (2005).
15) Karypis, G. and Kumar, V.: Multilevel k-way Partitioning Scheme for Irregular Graphs, *Journal of Parallel and Distributed Computing*, Vol.48, pp.96–129 (1998).
16) Karypis, G. and Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs, *SIAM Journal on Scientific Computing*, Vol.20, pp.359–392 (1998).
17) Stoer, J. and Bulirsch, R.: *Introduction to Numerical Analysis*, 3rd edition, Springer, New York (2002).
18) Aggarwal, C.: *Data Streams: Models and Algorithms*, Advances in Database Systems (2007).
19) Malewicz, G., Austern, M.H., Bik, A.J.C., Dehnert, J.C., Horn, I., Leiser, N. and Czajkowski, G.: Pregel: A system for large-scale graph processing, *Proc. 2010 international conference on Management of data, SIGMOD '10*, pp.135–146, New York, NY, USA, ACM (2010).
20) Kang, U., Tsourakakis, C.E. and Faloutsos, C.: PEGASUS: A Peta-Scale Graph Mining System – Implementation and Observations (2009).
21) Desikan, P., Pathak, N., Srivastava, J. and Kumar, V.: Incremental page rank computation on evolving graphs, *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pp.1094–1095, New York, NY, USA, ACM (2005).
22) 山田雅信，高橋俊行，田浦健次朗，近山　隆：インクリメンタル PageRank による重要 Web ページの効率的な収集戦略，情報処理学会論文誌 (2004).

(平成 23 年 1 月 28 日受付)
(平成 23 年 6 月 14 日採録)

雁瀬　　優（学生会員）

1988 年生．2011 年東京工業大学工学部情報工学科卒業．同大学大学院情報理工学研究科計算工学専攻に在籍．2010 年より，ストリームコンピューティングの特性を利用した時系列予測や，複雑ネットワーク処理に関する研究に従事．

上野　晃司（学生会員）

1988 年生．2011 年東京工業大学工学部情報工学科卒業．同大学大学院情報理工学研究科計算工学専攻に在籍．2010 年より，ストリームコンピューティングにおける GPGPU を用いた性能最適化や，グラフ処理の並列化に関する研究に従事．

鈴村豊太郎（正会員）

1975 年生．2004 年東京工業大学大学院情報理工学研究科数理計算科学専攻博士課程修了．同年日本アイ・ビー・エム（株）入社．以来，同社東京基礎研究所にて，ストリームコンピューティング，大規模データ処理基盤，XML 処理や動的スクリプト言語処理等ソフトウェアシステムに関する性能最適化の研究に従事．現在，同研究所インフラストラクチャ・ソフトウェアグループ主任研究員．2009 年より東京工業大学大学院情報理工学研究科客員准教授．理学博士．電子情報通信学会，ACM，IEEE 各会員．