Vol.2011-ARC-197 No.22
Vol.2011-HPC-132 No.22
2011/11/29

Wikipediaの編集履歴を用いた大規模2部グラフのデータストリーム処理

竹野　創平[†1]　　上野　晃司[†1]
雁瀬　優[†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，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の併用が求められる．本研究では，バッチ処理を定期実行しつつデータストリーム処理を行うシステムを実装し，大規模2部グラフとしてWikipediaの編集履歴を用いて評価した．大規模グラフをリアルタイムで処理するために，ソーシャルネットワークに代表されるグラフの持つ性質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して処理を行った．結果，4ノード48コアを用いて，頂点数が207,329と1,847,166，エッジ数が22,034,825の2部グラフを処理し，グラフ分割数に対してデータストリーム処理で2乗，バッチ処理で3乗の高速化を30分割において実現した．その結果，実データの到着レートの約32倍の速度でデータストリーム処理することができた．

Data Stream Processing for Large-Scale Bipartite Graph with Wikipedia Edit History

Souhei Takeno,[†1] Koji Ueno,[†1] Masaru Ganse[†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 real-time stream computing model to such dynamic graph analysis with the combination of batch and stream processing model. 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 ap- peared in social networks and processing a set of divided streams with multiple compute nodes in parallel. We apply our system to the edit history of Wikipedia, which can be represented as large-scale bipartite graph. Our experimental results demonstrate that our system achieves up to square speedup on stream processing and cubic speedup on batch processing at 30 devision using 4 nodes including 48 cores with the proposed method on the bipartite graph, which has 207,329 vertexes and 1,847,166 vertexes and 22,034,825 edges. As a result, our system process the data stream at real data rate of Wikipedia edit history.

1. はじめに

近年，スーパーコンピュータの性能指標として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．これまでのグラフ処理の研究は，データをある程度蓄積したのち処理を行う，バッチ処理が中心であった．一方，大規模グラフ処理が必要とされる分野の中には，株式市場やリコメンデーションシステムなど，リアルタイム性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行うには，データを蓄積するまで待つ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．

大規模グラフの処理にかかる計算時間は膨大である．これに対してデータストリーム処理では，差分更新を行うなどして逐次実行を行うが，それだけではリアルタイムでの処理を行うのに十分ではない．また，グラフをデータストリーム処理する場合，エッジが更新される度に処理をおこなう必要がある．大規模なグラフであれば，全体としてエッジの更新頻度が高くなるので，時間あたりに行わなければならない計算も多くなる．

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一つとして，コミュニティ構造[2)]というものがある．コミュニティ構造とはエッジが密な集合同士が疎なエッジで繋がっている構造であり，分割して独立に計算してもある程度の精度で処理できることが知られている．この性質を利用し，2部グラフを対象としたデータストリーム処理の高速化を行い，評価したものとして雁瀬らの論文[3)]がある．雁瀬らの論文では，匿名掲示板を2部グラフとして扱い，Random Walk with Restart法[4),5)]により頂点間の関連性を解析している．しかし，評価対象となるデータが大規模なグラフではないという点で十分ではない．

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

Vol.2011-ARC-197 No.22
Vol.2011-HPC-132 No.22
2011/11/29

また，バッチ処理システムとの併用が前提となっているが，データストリーム処理システムのみの実装・評価であった．従って，大規模な 2 部グラフに対しての適用と，バッチ処理システムと併用した場合の評価を行う必要がある．そこで本研究では，雁瀬らの実装を元に，バッチ処理システムの実装を行い，Wikipedia の編集履歴を用いて大規模な 2 部グラフに対する評価を行った．

匿名掲示板や Wikipedia の編集以外で，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，次のようなものが挙げられる．

- 株式市場:2 つの頂点群にユーザーと株式を持ち，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，ユーザー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ーとファイルを持ち，データのダウンロードやアッ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ー間の類似性を判断し，最も必要なファイルを持つユーザー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．

2. データストリーム処理と System S

本研究では，大規模グラフに対してデータストリーム処理を行うために，IBM Research の提供するデータストリーム処理処理系 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ついて補足説明を行い，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[6)–10)] とは，データのストリームを蓄積することなく逐次に処理していく，新しい計算パラダイムである．ここで言うストリームとは，始点・終点という概念のない情報の列であり，時々刻々と流れてくる時系列データなどである．バッチ処理と呼ばれる，計算対象を全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, 時系列で前後する僅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．その一例として MIT の Borealis や IBM Research の System S[7)–9)] 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 SPADE

System S[7)–9)] は，データフロー図から直感的に処理フローを記述できる SPADE[7)] 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[8)] によって構成される．SPADE は，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記述していく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必要がない，高級な宣言的言語である．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 を用いたユーザー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[9)] では実行中のノード割り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．

3. 問題定義

本研究では，コミュニティ構造を持った二部グラフを対象として，グラフ分割を行い Random Walk with Restart 法[4),5)] を用いて頂点間の関連性解析を行う，データストリーム処理システムを構築・評価する．この章では，データストリーム処理システムの位置づけを述べ，また用いている手法が適用できる前提条件を述べる．

3.1 データストリーム処理の位置づけ

処理の効率と精度を高めるという観点から，本研究のデータストリーム処理は，バッチ処理と併用される前提とした．データストリーム処理は全ての面でバッチ処理より優れているわけではない．リアルタイムで応答する必要があるので，精度の高い結果を出すことができないというデメリットも存在する．言い換えれば，バッチ処理ではその制約がないため，計算量は多いが精度の高いアルゴリズムなどを使用することによって，精度の高い結果を出すことができる．従って，バッチ処理で精度の高い結果を得て，データストリーム処理でリアルタイム性への要求に対応することで．処理の効率と精度を同時に実現できる．

3.2 対象となるグラフ

対象とするグラフは，コミュニティ構造をもった 2 部グラフとする．コミュニティ構造を

Vol.2011-ARC-197 No.22
Vol.2011-HPC-132 No.22
2011/11/29

もったグラフを対象とする理由は，詳しくは次章で述べるが，精度を保ちつつ高速化手法を実行するのに必要なためである．2 部グラフを対象とする理由は，グラフの構造が有用である点，2 つの頂点群のうち片方の頂点群のみを分割する（5 章参照）だけなら，エッジ情報を損なわない点が挙げられる．データストリーム処理を行う都合上，時系列上で一定以上の変化が現れるグラフが望ましい．

3.3 対象となる 2 部グラフの例

条件を満たす 2 部グラフの例として，Wikipedia における編集者と記事の関係が挙げられる．Wikipedia は，Wiki によって誰でも編集が可能な，オンラインのフリーな百科事典である．それぞれの記事（以下，ページ）に対して，一人以上の編集者（以下，ユーザー）が存在し，Wikipedia の編集履歴はこの関係を記録している．

Wikipedia の編集履歴は，ページとユーザーを二つの頂点群とし，書き込み回数をページ-ユーザー間のエッジの重みとする 2 部グラフとして扱える．非常に大規模な 2 部グラフなので，システムの適用可能サイズを評価するのに適している．また，WWW のネットワークの一部であるため，コミュニティ構造などの性質を持っていると考えられる．

4. 解析アルゴリズムと高速化手法

この章では，2 部グラフを解析するアルゴリズムについて触れたのち，高速化手法について説明する．

4.1 用いる解析アルゴリズム

本研究では，グラフの解析アルゴリズムに Random Walk with Restart 法[5] を用いる．各頂点間の関連性がわかるため，レコメンデーションシステムなどへ幅広く応用が可能である．ただし実装では，バッチ処理に BB_LIN[4]，データストリーム処理に Fast-Single-Update[4] というアルゴリズムを用いる．BB_LIN は，Random Walk with Restart 法を 2 部グラフに適用したアルゴリズムであり，2 部グラフのうち小さいほうの頂点群の要素数（L とする）の 3 乗の計算量である．Fast-Single-Update は，BB_LIN などで得られた関連性を近似的に差分更新するアルゴリズムであり，L の 2 乗の計算量である．

4.2 グラフのコミュニティ構造とグラフ分割

第 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一つとして，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同士が疎なエッジで繋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．本研究ではこの性質を利用して 2 部グラフのうち片方の頂点群を分割し，高速化を行った．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．

4.3 グラフのコミュニティ構造に関する既存研究

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),5),12)–15)] は多数存在しており，本研究と同様にコミュニティ構造をも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[3),15)]（8 章参照）も存在する．本研究と同様，グラフ分割のアルゴリズムに METIS[16),17)] を用いている．ただし，Sun らの論文[15)] 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る．また，雁瀬らの論文[3)] はグラフ分割において 2 部グラフ関連性解析をデータストリーム処理で行ったものであるが，いくつか改善すべき点がある．本論文でも述べたのと同様に，雁瀬らの論文でもデータストリーム処理はバッチ処理と併せて実行される前提となっているが，データストリーム処理のみの実装・評価となっている．また，大規模なグラフに対する高速化手法として提案しているが，適用しているグラフが十分大規模とはいえないものであった．そこで，本研究では雁瀬らの実装を改善・拡張することで，2 部グラフ関連性解析のバッチ処理機構を追加，さらに大規模な 2 部グラフに対して適用し，評価を行った．

4.4 高 速 化

ここでは，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．例えば，要素数の 2 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 $O(n^2)$ または $O(n^3)$ の計算量がかかるため，グラフ分割数により k^2 または k^3 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，更なる高速化が得られる．

Vol.2011-ARC-197 No.22
Vol.2011-HPC-132 No.22
2011/11/29

5. 実　　装

この章では，本研究で実装したシステムの詳細について述べる．

5.1 システム要件

3 章で述べたように，大規模 2 部グラフのデータストリーム処理はそれ単独ではなく，バッチ処理と併せて行う．精度を保つために，通常時はデータストリーム処理で差分更新を行い，定期的にバッチ処理を実行して正確な関連性を計算する必要がある．

精度の良い結果を得るためには，コミュニティ構造を正確に抽出することも重要である．データストリーム処理の過程で関連性が変化することで，コミュニティ構造自体が変化する可能性も当然あり，そのままでは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，グラフの再分割を行う必要がある．本実装では，グラフの再分割は，バッチ処理の実行時に前処理として行っている．

グラフを分割してリアルタイム処理する場合，分割時点では存在しない頂点が出現する可能性も当然ある．この新規頂点は分割先が決まっていないため，データストリーム処理の中で分割先を動的に決定する仕組みが必要である．

5.2 システムの全体像

実装では，データストリーム処理系として System S を，Split オペレータ（**図 1** 参照）においてグラフ分割を行うコンポーネントとしてグラフ分割ライブラリ METIS[16] を，SubGraph オペレータ（図 1 参照）において 2 部グラフの関連性を求めるアルゴリズムとして BB_LIN と FSU（Fast-Single-Update）[4] を，行列演算ライブラリとして LAPACK，ATLAS を用いた．System S の実装に関しては 2 章を，METIS，BB_LIN 及び FSU に関してはそれぞれ 5.4 章，5.5 章を参照して頂きたい．

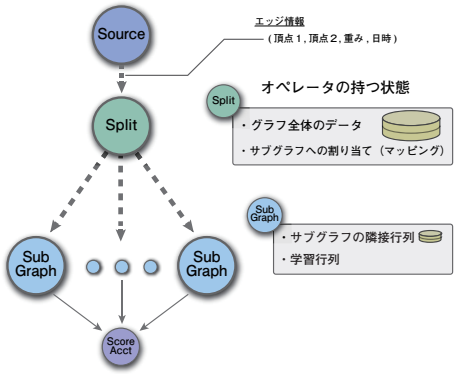

図 1 実装したシステムのデータフロー図
Fig. 1 Data flow graph of this system

このシステムのデータフロー図は図 1 のようになっている．Split，SubGraph，ScoreAcct はいずれもユーザー定義オペレータ（2.2 章参照）として実装している．

本実装ではバッチ処理は System S 上で実装しており，Split オペレータと SubGraph オペレータにおいて行っている．まずは，定期的に実行されるバッチ処理時の動作について述べる．

（1） Split オペレータには，その時点でのグラフ全体のデータが保持されている．これを元にグラフをいくつかのサブグラフに分割する．その結果の分割群側の各頂点のサブグラフに対する割り当て（マッピング）が，Split オペレータに保持される．その後，Split オペレータはこのマッピングに従って，各 SubGraph オペレータに対応するサブグラフのデータを送信する．

（2） SubGraph オペレータは，Split オペレータから送られてきたサブグラフのデータを隣接行列として保持する．その後，そのデータを元に BB_LIN を実行し頂点間の関連性を算出する．この結果を学習行列と呼び，各 SubGraph が密行列として保持している．

以上がバッチ処理で行うことである．続いてデータストリーム処理での動作について，オペレータごとに順を追って述べる．

（1） Source オペレータは，エッジ情報をシステムの外部から受け取り，Split オペレータに送信する．

（2） Split オペレータは，送られてきたエッジ情報をグラフ全体のデータに反映し，保持しているマッピングに従って，適切な SubGraph にエッジ情報を送信する．

（3） SubGraph オペレータは，送られてきたエッジ情報を FSU で処理し，学習行列を更新する．

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．また，バッチ処理はシグナルを送ることで任意のタイミングで実行できるが，実験では一定数のエッジを処理するごとに行っている．

解析結果は SubGraph オペレータが学習行列として保持しており，このデータはデータストリーム処理により常に最新のデータを処理した結果となっている．結果の出力は要求に応じて行うようになっており，ScoreAcct オペレータが結果を集約して行う．

5.3 Split オペレータ

この節では，Split オペレータの内部実装について説明する．バッチ処理時の分割と，データストリーム処理時のエッジの振り分けの二つの役割がある．データストリーム処理時の基本的な役割は単純で，Source オペレータから受け取ったエッジ情報を保持してるグラフ全

Vol.2011-ARC-197 No.22
Vol.2011-HPC-132 No.22
2011/11/29

体のデータに反映させた後，保持しているマッピングデータに基づいて適切な SubGraph に送信するというものである．このマッピングを決定する，分割の方法について述べる．

5.3.1 グラフの分割

4.1 章でも述べたように，BB_LIN と FSU の計算量は，2 部グラフの小さい方の頂点群の要素数に依存する．従って，小さい方の頂点群が分割対象頂点群となる．そこで，まず Random Walk with Restart 法を簡易的に用いて（以下，簡易 RWR 法），2 部グラフのデータから分割対象頂点群同士の関連性のグラフを作成する．簡易 RWR 法では疎行列と疎行列の積を計算すればよい．これによりできた分割対象頂点群のグラフを METIS[16),17)] に入力し分割する．分割結果はマッピングデータとして保持される．このマッピングデータは，頂点の ID をインデックスとして，そこから対応するサブグラフの ID が引けるような配列として実装されている．その後，マッピングを元に各 SubGraph オペレータにサブグラフのデータを送信する．マッピングデータの更新が準備できるまでの処理は，バックグラウンドで実行されるため，データストリーム処理を長期に止めることはない．

疎行列表現[18)] 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，データの追加が高速な独自のデータ構造を使っている．このデータ構造は，2 部グラフの一方の頂点群の頂点それぞれに対して配列を与え，各頂点を端点とするエッジを格納する．このため，配列を与えた側の頂点群に属する頂点の，隣接ベクトルは高速で取り出せる．エッジ情報更新の際は対応する配列の末尾にデータを追加する．このためエッジの追加が高速で行える一方，配列内でエッジ情報の重複が発生するため，適宜ソートを実行してこれを解消してやる必要がある．また，サブグラフへのデータの送信と，METIS への入力の際は，メモリの使用量が少ない Compressed Sparse Row 形式（CSR 形式）[18)] に変換している．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．

5.3.2 新規頂点の動的な追加

データストリーム処理を行う場合，全節で述べたグラフの分割の際には存在しなかった，新たな頂点が現れる場合もある．これが分割対象頂点群に属する場合，その頂点をどの SubGraph に割り当てるかを決定する必要がある．これは，次のような方法で行った．まず，分割対照群の新規頂点を含むエッジは，その頂点についてのエッジが一定数になるまでバッファに蓄積する．これは，新規頂点がどのサブグラフに属するかを正しく決定するためである．このバッファのサイズはヒューリスティックなパラメータであるが，本実装では雁瀬らの実装を引き継いで 30 エッジとした．一定数エッジがバッファにたまった時点で，その新規頂点の配属先を決定する．決定方法は，その新規頂点をその時点で既にあるサブグラフに配属したとき，簡易 RWR 法で計算した関連性の欠損が最も小さくなるような配属先を計算するようにした．これは，バッチ処理で分割する際，簡易 RWR 法でユーザー同士の関連性のグラフを作成し METIS で分割していること，METIS はエッジカットが最小になるようにグラフを分割するアルゴリズムであること（5.3.3 章参照）の 2 点から整合性のある決定方法である．この方法は，新規頂点と一部の頂点との関連性のみを計算すればよい差分計算である．疎行列と疎ベクトルの積を計算するだけで済むため，新規頂点追加の度にグラフ全体を METIS にかけていた雁瀬らの実装よりも高速に新規頂点の配属先を決定することができる．

5.3.3 グラフ分割ライブラリ METIS

グラフ分割ライブラリ METIS[16)] 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)[17)] 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 16)，17) を参照して頂きたい．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．

5.4 SubGraph オペレータ

この節では，SubGraph オペレータの内部実装について説明する．SubGraph オペレータは，Split オペレータから渡されたエッジデータに対して解析処理を行う．この解析処理には，バッチ処理時は BB_LIN，データストリーム処理時は FSU を使用して行う．

BB_LIN[5)] は Random Walk with Restart 法を 2 部グラフに適用したアルゴリズムである．計算量は，2 部グラフの 2 つの頂点群のうち，小さい方の頂点群の要素数（L とする）の 3 乗である．計算量が非常に大きいため，バッチ処理時においてもそのまま計算するのは困難である．そこで，本研究では，データストリーム処理と同様に，バッチ処理で

Vol.2011-ARC-197 No.22
Vol.2011-HPC-132 No.22
2011/11/29

もグラフ分割を行う．BB_LIN の出力結果は小さい方の頂点群内での関連性であり，これを学習行列と呼ぶ．学習行列は L×L の密行列になっている．2 部グラフの任意の頂点間の関連性は，この学習行列と保持している隣接行列から高速で算出できるため，データストリーム処理では，この学習行列を最新に保っていればよい．そのためのアルゴリズムとして Fast-Single-Update（FSU）[4] があり，本研究でもこれを用いる．FSU は BB_LIN を近似的に差分更新することによって，全ての要素を計算する BB_LIN と比較して計算量を減らすアルゴリズムである．BB_LIN が L の 3 乗の計算量であるのに対し，FSU は L の 2 乗の計算量になっており，計算量を劇的に減らしている．しかし，これをそのまま大規模な 2 部グラフのデータストリーム処理に用いようとすると，ストリーム上を流れるエッジ情報の到着頻度内で処理を終えることができず，リアルタイム性を保持できなくなってしまう．そのため，本研究ではグラフを分割することによってこの問題を解決している．

グラフの再分割時には，学習行列，隣接行列などの保持データは一度破棄され，リセットされる．また，BB_LIN，FSU での行列演算には LAPACK，ATLAS を用いることで高速化を図っている．

6. 評　　価

この章では，前章で説明した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 4 台使用した．環境は全ノード共通で，CPU は 1 ノードに Intel Xeon X5670（2.93GHz，6 コア）が 2 ソケット，メモリは 8GB，OS は CentOS 5.4，ソフトウェアは InfoSphere Streams1.2.0（System S），gcc 4.1.2，METIS 4.0，行列演算には ATLAS 3.8.3, LAPACK 3.3.0 を使用した．ネットワーク環境はそれぞれ 1GB Ethernet で接続する．実装のフロー図での Source，Split に 1 台（12 コア），SubGraph：3 台（36 コア）を割り当てた．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 のコンパイルは全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 Wikipedia の編集履歴を用いた．Wikipedia では，編集者（以下ユーザー）の方が記事（以下ページ）よりも少ないため，ユーザー群を分割することになる．エッジは両端点としてページ ID とユーザー ID，重みとして書き込み回数（増分），到着時刻として書き込み日時を持つ．今回はこのデータから，2 通りのサイズのグラフを用いる．1 つは，2002/07/24 から 2004/12/16 までのデータを蓄積したグラフであり，ユーザー数は 4,995，ページ数は 138,109，エッジ数は 773,541 である．このグラフを「5000 ユーザー」と呼ぶ．もう 1 つは，2002/07/24 から 2011/03/11 までのデータを蓄積したグラフであり，ユーザー数は 207,329，ページ数は 1,847,166，エッジ数は 22,034,825 である．このグラフを「200000 ユーザー」と呼ぶ．書き込み頻度は，2011 年 3 月時点で 9.2 回／分となっている．2 つのサイズのグラフを用いるのは，分割なし，低分割数での精度を評価するためである．200000 ユーザーでは計算量が大きいため，分割数を大きくしなければ処理を行えない．

Wikipedia の編集履歴を用いる理由としては，グラフが大規模であること，コミュニティ構造を持っているであること，解析結果が有用であることの 3 点が挙げられる．例えば，Wikipedia のあるページに最も関連する 3 つのページをリアルタイムに表示でき，ユーザーに知らせることができたとする．すると，ある事件が起こった際，その事件の記事の編集者に，関連するページの編集を提案することで，周辺記事の編集を促進できると考えられる．

6.3 精度の評価方法

精度の評価方法は，実運用を想定して，あるページに最も関連する上位 10 個のページの正しさを測ることとする．比較対象としては，Split オペレータ内において Random Walk with Restart 法を反復法で実行した結果を用いる．ページのサンプル数は実験コストの関係から 25 とし，その精度の平均を用いる．サンプルの選定方法はランダムである．

6.4 実 験 結 果

6.4.1 5000 ユーザーにおける実験結果

5000 ユーザーにおいては，250000 エッジをデータストリーム処理しその精度を測定した．

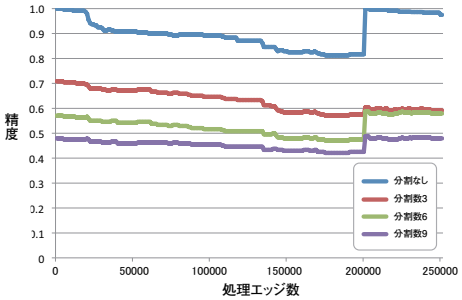

図 2 処理エッジ数に対する精度変化（5000 ユーザー）
Fig. 2 Speedup stream processing with graph partitions

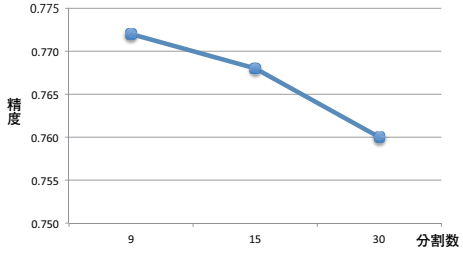

図 3 グラフ分割数に対する精度変化（200000 ユーザー）
Fig. 3 Shortening BB_LIN processing-time with graph partitions

Vol.2011-ARC-197 No.22
Vol.2011-HPC-132 No.22
2011/11/29

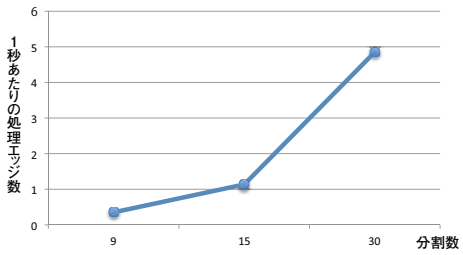

図 4 グラフ分割によるストリーム処理の高速化（200000 ユーザー）
Fig. 4 Speedup stream processing with graph partitions

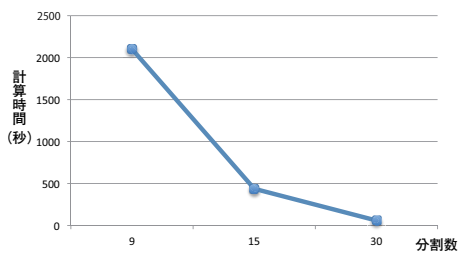

図 5 グラフ分割による BB_LIN の計算時間の短縮（200000 ユーザー）
Fig. 5 Shortening BB_LIN processing-time with graph partitions

バッチ処理の実行間隔は 200000 エッジごととしたので，処理開始時と 200000 エッジ処理した時点でバッチ処理が実行される．この実験を，分割なし，3，6，9 のように分割数を変えて行った．その結果が**図 2** である．

図 2 を見ると，確かに分割数を増やすに従って精度が低下していき，またデータストリーム処理を続けるに従って精度が低下している．また，200000 エッジにてバッチ処理を行っているので，精度が処理開始時の水準にまで回復することも確認できる．分割数 3 において精度が十分に回復していない点について，考えられる原因として，コミュニティ構造の正確な抽出とグラフ分割が上手くいかなかったことがある．

6.4.2 200000 ユーザーにおける実験結果

200000 ユーザーにおいては，10000 エッジをデータストリーム処理し，その精度と性能を測定した．ただし，10000 エッジでは精度の比較元の関連性出力に変化がなかった．さらに予備実験として同グラフで 150000 エッジを受け取り比較元の変化を測った結果，変化率は 1%程度であった．このことから，精度の測定はバッチ処理時点でのみ行った．その結果が**図 3**，**図 4**，**図 5** である．

図 3 を見ると，分割数を増やすに従って精度が低下していることがわかる．しかし，5000 ユーザーの結果に比べ，精度低下があまり起こっていないことがわかる．これは，コミュニティ構造抽出の精度の違いだと考えられる．5000 ユーザーにおいてはエッジデータの蓄積が十分でなく，従ってグラフの分割精度があまり高くならなかったと考えられる．

図 5 はバッチ処理の中心処理である BB_LIN の計算時間を測定した結果である．この結果は，分割数に対してほぼ 3 乗で反比例しており，BB_LIN の計算量からの予想と一致する．図 4 はデータストリーム処理のスループットを測定した結果である．この結果は，分割数に対して 2 乗をやや上回って比例しており，FSU の計算量からの予想と一致する．計算量の削減以上に高速化ができるのは，分割によって並列処理を行えるようになるためである．以上のことから，バッチ処理，データストリーム処理ともに大規模グラフに対するグラフ分割による高速化に成功していると言える．その結果，Wikipedia の平均書き込み頻度の約 32 倍のデータレートである，毎秒 4.9 エッジを処理することができた．

7. 議　　論

実験結果から，データストリーム処理を大規模なグラフに適用できること，データストリーム処理を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に行うと効果的であるということを示すことができた．バッチ処理の実行間隔は対象となるグラフのサイズなどに依存するが，この間に急激なグラフの変化が起こると精度の大幅な低下が起こってしまう．変化が急激なグラフを処理する場合は，頻繁にバッチ処理を実行する必要がある．予備実験で示したように，Wikipedia の編集履歴はグラフのサイズは大きいが変化は小さいグラフであった．

現在のシステムは，あくまでバッチ処理の結果を随時差分更新していくものであり，その時々におけるトレンドなどを強調して捉えるような仕組みは持っていない．そのため，過去のデータの蓄積が膨大な場合，ある時点で局所的に急激な変化が起こっていても，出力には反映されにくいという問題がある．特に．頻繁にグラフの構造が変化するようなグラフの場合，過去の変化が現在の変化に干渉してしまい，この問題は顕著になる．そこで，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への考慮

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[19] と言う手法がある．この手法は過去全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．この方法を用いることで，過去の蓄積により現在の変化が出力に現れにくくなるという問題は解決することができる．また，この手法を導入するメリットは別にもある．ソーシャルネットワークに代表される大規模なグラフには，多くの頂点は僅かなエッジしか持たず，一部の頂点が多くのエッジを持つという性質がある．Wikipedia の編集履歴の場合，書き込み回数が 5 回以下のユーザーは全体の 60%いるが，これらのユーザーの全体の書き込みに占める割合は 1%にすぎない．これらのユーザーのうち多くはその時点では活動していないと想像できるが，現在の実装で

Vol.2011-ARC-197 No.22
Vol.2011-HPC-132 No.22
2011/11/29

はこれらのユーザーも既存の頂点として考慮して計算を行っている．スライディングウィンドウを用いることで，一定期間以上現れないデータを削除することができるため，頂点数の削減から大幅な計算量の削減が見込める．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事で対応するアルゴリズム[19] も存在するが，これらの研究は未だ発展途上である．

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[20],PEGASUS[21] 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元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時に処理を終了する．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 4)，22)，23) のような例がある．

論文 22)，23) で紹介されているのはインクリメンタル PR 法である．Prasanna らによる研究[22] 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[23] では，PR を計算する際に，反復して計算する回数を減らす事でリアルタイムに 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[4] 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 法を用いている．

Sun らによる論文[15] 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も適用している点で異なる．

9. ま と め

頂点数が 207,329 と 1,847,166，エッジ数にして 22,034,825 という大規模な現実の 2 部グラフに対して，4 ノード 48 コアで並列に処理を行った．これにより，グラフ分割数に対して，バッチ処理時には 3 乗，データストリーム処理時には 2 乗の高速化を実現した．その結果，毎秒 4.9 エッジをデータストリーム処理することができ，Wikipedia のページ間の関連性をリアルタイムで解析できることを示した．また，バッチ処理とデータストリーム処理を併用した際の，精度の変化を示すことで，精度と性能のトレードオフを検証した．

今後の展望としては，別の大規模な実データに大しての適用がある．Wikipedia は規模は大きいが変化が小さいグラフであったので，変化が大きいと思われるグラフでの検証が望まれる．このようなグラフとして twitter データが挙げられる．今後このシステムを，twitter データに対して適用することで，変化が急激なグラフに対しての有効性を調べる．

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, *Proceedings of SIAM International Data Mining Conference (SDM 2005)* (2005).
3) 雁瀬優，上野晃司，鈴村豊太郎：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，情報処理学会論文誌 (ACS 論文誌第 35 号) (2011).

Vol.2011-ARC-197 No.22
Vol.2011-HPC-132 No.22
2011/11/29

4) Tong, H., Papadimitriou, S., Yu, P.S. and Faloutsos, C.: Proximity Tracking on Time-Evolving Bipartite Graphs (2008).
5) Tong, H., Faloutsos, C. and Pan, J.-y.: Fast Random Walk with Restart and Its Applications, *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, Washington, DC, USA, IEEE Computer Society, pp.613–622 (2006).
6) 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).
7) Gedik, B., Andrade, H., Wu, K.L., Yu, P.S. and Doo, M.: SPADE: the system s declarative stream processing engine, *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, New York, NY, USA, ACM, pp. 1123–1134 (2008).
8) 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, *Proceedings of the 4th international workshop on Data mining standards, services and platforms*, New York, NY, USA, ACM, pp.27–37 (2006).
9) 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, Springer Berlin / Heidelberg, pp. 306–325 (2008).
10) 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).
11) 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).
12) 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).
13) Flake, G., Lawrence, S. and Giles, C.L.: Efficient Identification of Web Communities, *Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Boston, MA, pp.150–160 (2000).
14) Newman, M. E.J.: Fast algolythm for detecting community structure in networks (2003).
15) Sun, J., Qu, H., Chakrabarti, D. and Faloutsos, C.: Neighborhood Formation and Anomaly Detection in Bipartite Graphs, *ICDM '05: Proceedings of the Fifth IEEE International Conference on Data Mining*, Washington, DC, USA, IEEE Computer Society, pp.418–425 (2005).
16) 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).
17) 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).
18) Stoer, J. and Bulirsch, R.: *Introduction to Numerical Analysis*, Springer, New York, 3 edition (2002).
19) Aggarwal, C.: *Data Streams: Models and Algorithms*, Advances in Database Systems (2007).
20) 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, *Proceedings of the 2010 international conference on Management of data*, SIGMOD '10, New York, NY, USA, ACM, pp.135–146 (2010).
21) Kang, U., Tsourakakis, C.E. and Faloutsos, C.: PEGASUS: A Peta-Scale Graph Mining System- Implementation and Observations (2009).
22) 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*, New York, NY, USA, ACM, pp.1094–1095 (2005).
23) 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).