

インクリメンタル大規模グラフストリーム処理系 の実装と評価

雁 瀬 優[†] 西 井 俊 介[†] 鈴 村 豊 太 郎^{†,††}

近年、時系列上で成長するようなグラフデータに対してリアルタイムに解析を行う研究が活発に行われている。そういった研究の多くはグラフを静的なものとしてみなし、解析をバッチ的に行ってしまっているが、本研究では、時系列上を流れるグラフストリームデータに対して動的なグラフ解析を行うことができる手法を提案した。既存の GIM-V モデルと比較して、1 ノード 4 コアの計算機を 4 台使用したクラスタ上で (計 16 コア)、SCALE16 (頂点数 65,536, 辺数 8,364,525) の対数正規分布グラフに対して PageRank アルゴリズムを実験し、48% の高速化を実現した。同様に、SCALE18 (頂点数 262,144, 辺数 8,388,608) の Kronecker グラフに対して Random Walk with Restart アルゴリズムを実験し 3023% の高速化を、SCALE18 の Kronecker グラフに対して最短経路探索アルゴリズムを実験し 501% の高速化を実現した。

A System for Incremental Large-Scale Graph Stream Processing

MASARU GANSE,[†] SHUNSUKE NISHII[†] and TOYOTARO SUZUMURA^{†,††}

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 incremental graph stream computing model to such dynamic graph analysis. Our performance evaluation demonstrates that our method achieves up to 48% speedup on PageRank with SCALE16 Log-normal Graph (vertexes=65,536, edges=8,364,525) with 4 nodes, 3023% speedup on Random walk with Restart with SCALE18 Kronecker Graph (vertexes=262,144, edges=8,388,608) with 4 nodes and 501% speedup on Single Source Shortest paths with SCALE18 Kronecker Graph with 4 nodes against original GIM-V.

1. 研究背景

近年、大規模データ処理の需要は高まりつつある。大規模データとして注目を集めているデータ構造の 1 例としてグラフストリームが挙げられる。グラフストリームとは、Web グラフや道路交通ネットワーク、たんぱく質の構造など時系列上で変化を生じるグラフを流れ (ストリーム) として扱う構造を指す。グラフストリームは他のストリームデータと同様に時々刻々とデータが流れてきて構造が変化するという特徴を持っており、そういったデータに対して効率よく処理を行うためには、従来のデータ蓄積後に処理を行うバッチ処理だけではなく、流れてくるデータに対して逐次に

処理を行えるデータストリーム処理系による処理が望まれている。そこで本稿では、データストリーム処理系を用いてグラフストリームの処理を行うことを提案する。そのための計算モデルとして、既存の大規模グラフ処理系 PEGASUS¹⁾ の GIM-V モデル (Generalized Iterative Matrix-Vector multiplication) に汎用的な動的グラフ処理メソッドを組み込んだグラフ処理系 Incremental GIM-V モデル (IGIM-V モデル) を提案する。

2. 関連研究

本稿では関連研究として既存の大規模グラフ処理系、動的グラフ処理メソッド、データストリーム処理系について解説する。

2.1 大規模グラフ処理系

大規模グラフ処理を行うシステムに関する研究として、Google の Pregel²⁾ やカーネギーメロン大学の

[†] 東京工業大学

Tokyo Institute of Technology

^{††} IBM 東京基礎研究所

IBM Research-Tokyo

PEGASUS¹⁾などが挙げられる。これらはバッチ処理に基づくグラフ処理システムであり、大規模なデータを汎用的に効率よく処理するために複数の計算ノードを用いて並列分散処理を行っている。Pregelでは、頂点ごとの処理とメッセージパッシングにより反復計算を行い、最終的な解析結果を得る。様々なグラフ処理を実装できるように、C++ APIの形でインターフェースを提供している。計算性能としては、Google社内の300コアからなるクラスタ上で、10億頂点、1271億辺からなるグラフに対するシングルソースの最短路問題を、700~800秒で解くことができる。PEGASUSはMap Reduceの処理系であるHadoop³⁾を用いたオープンソースのグラフ処理システムである。PEGASUSでは、GIM-V (Generalized Iterative Matrix-Vector multiplication) という、行列とベクトルの乗算(べき乗法)を一般化したものとして抽出し、その構造に基づいて計算の最適化を行っている。計算性能としては、Yahoo!のM45 Hadoopクラスタを含む90台の計算機を用いて、5.9万頂点、2.82億辺からなるグラフに対するPageRankの計算を50~100秒で解くことができる。Pregel, PEGASUSともに、計算時間はデータの大きさ(辺の数)や計算機の台数の逆数に線形にスケールする。

2.2 動的グラフ処理メソッド

グラフ処理のうち、特にPageRankを高速に計算する手法として、Desikanら⁴⁾、Yamadaら⁵⁾の提唱するIncremental PageRank, Kamvarら⁶⁾のAdaptive PageRankなどがある。Desikanらの提唱するIncremental PageRankは、成長し続けるグラフ構造に対して、ある時点のPageRankの計算を、それよりも古い時点でのPageRankの計算結果を利用して高速に解く手法である。Yamadaらの提唱するIncremental PageRankはDesikanらの提唱するIncremental Pagerankとは異なり、Web Pageの効率的な収集戦略としてクローラの処理実行時間以下で処理を終えるためにPageRankの反復計算の回数を制限するという手法をとっている。KambarらのAdaptive PageRankは、PageRankのべき乗法による計算の際に、頂点ごとに収束の速さが異なることを利用して、収束した頂点から順に計算対象から除外していくことで計算を速くする手法である。本研究では、グラフストリームに対する処理の高速化のために、これらのPageRankの高速計算手法を利用して汎用的で逐次的な処理を実現している。

2.3 データストリーム処理系とバッチ処理系
大規模なグラフ処理系が注目を集めている一方で、

データストリーム処理系もまた注目を集めている。既存のデータストリーム処理系としては、Yahoo S4⁷⁾やBorealis⁸⁾、TelegraphCQ⁹⁾などがあるが、商業利用を目的としたデータストリーム処理系も登場しており、その代表的な例として挙げられるのがIBM System S¹⁰⁾である。IBM System Sで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。

既存のバッチ的な解析手法も情報量の爆発的な増加に対して一定の貢献をしている。バッチ的な処理系としてはYahoo社で開発されたオープンソースの処理系Hadoop³⁾が有名である。Hadoopは大規模分散処理を目的としたプラットフォームでありHadoopユーザはMapReduceと呼ばれるプログラミングモデルを使用することで、簡単に並列分散アプリケーションを作ることができる。

2.4 汎用グラフ処理モデル GIM-V

GIM-V (Generalized Iterative Matrix-Vector multiplication) とは、行列とベクトルの乗算を一般化したものである。 $n \times n$ 行列 M と n 次元ベクトル v の乗算を行い、ベクトル v' を得る計算 $v' = M \times v$ を、要素ごとの式で表わすと、 $v'_i = \sum_{j=1}^n M_{ij} \times v_j$ となる。この計算は「乗算」「総和」「代入」の3つの演算からなる。これらを一般化し、3つの演算をそれぞれ関数 `combine2`, `combineAll`, `assign` とおくと、式は以下の通りとなる。

$$v'_i = \text{assign}(vi, \text{combineAll}(\text{combine2}(M_{i1}, v_1), \dots, \text{combine2}(M_{in}, v_n)))$$

ここで、行列の要素の型 (T_M)、ベクトルの要素の型 (T_v)、`combine2` の戻り値の型 (T_c) は、実数型 (float, double) やブーリアン型 (bool) などのプリミティブだけでなく、配列や構造体などを取ることも可能である。それぞれの関数の写像を書くと、

$$\text{combine2} : T_M \times T_v \rightarrow T_c \quad (1)$$

$$\text{combineAll} : T_c \text{ の配列} \rightarrow T_v \quad (2)$$

$$\text{assign} : T_v \times T_v \rightarrow T_v \quad (3)$$

となる。グラフ処理としては、行列 M をグラフの隣接行列(各辺の値)、ベクトル v を各頂点の値として扱い、GIM-Vの計算を反復的に行うことで、様々なグラフ処理を行うことができる。なお、 M_{ij} の値は、 j から i への辺の重みを表わしている。PEGASUSでは、GIM-Vモデルの計算をMapReduce(Hadoop)上で実行している。このMapReduce計算は2ステージのMapReduceからなる。まず、ステージ1のMapReduceで、グラフデータの入力(辺、頂点)をMapジョブから行い、Reduceジョブでは j をキーとして `combine2()` を

実行して、その出力をステージ2のMapReduceの入力に与える。ステージ2では、MapジョブはそのままReduceジョブにデータを投入し、Reduceジョブでは i をキーとしてcombineAll(), assign()を実行、収束した場合はそこで全計算を終了し、再度反復する場合はステージ2のMapReduceの出力をステージ1のMapReduceの入力に与える。なお、PEGASUS¹⁾では、GIM-Vを用いて、PageRank, Random Walk with Restart, 直径推定, 連結成分推定のアルゴリズムを実装している。

3. インクリメンタルグラフ処理モデル Incremental GIM-V の提案

本節では、データストリーム処理によるグラフ処理のための計算モデルである Incremental GIM-V (IGIM-V モデル) について説明する。IGIM-V モデルの大筋の枠組みは基本となる GIM-V モデル¹⁾に加え、変化頂点のみの更新を行い変化しない頂点に対しては計算処理から除外することで計算時間の高速化を図っている。なお、IGIM-V で使用されている関数である combine2, combineAll, assign については既存の GIM-V と同様の処理を行っている。

3.1 Incremental GIM-V のアルゴリズム設計

IGIM-V モデルは主に3つの方法によってグラフ処理の高速化を行っている。第1の手法として計算頂点の収束速度の違いを利用した計算領域の逐次削減、第2にグラフの到達可能領域を利用した計算領域の限定、第3に収束の定義の変更が挙げられる。以下にその詳細を述べる。

3.1.1 Kamvar らの Adaptive PageRank⁶⁾ 手法の利用

Kamvar らの Adaptive PageRank⁶⁾ 手法は既存のアプリケーションに対する高速化手法である。この高速化手法は Web などの偏りが多いグラフに対して処理を行う際に頂点の収束速度の偏りを利用して、既存のアプリケーションの計算を高速化することにより、この計算高速化から得られる結果自体は GIM-V とほぼ変わらない。Adaptive に PageRank の計算を行う際に、収束されたとして除外された頂点に対して、閾値以下の変化が発生していても再計算を行う頂点としては扱われないため、通常の PageRank と比較するとごくわずかにスコアが変化してしまうという問題点が存在するが、ランク順位に影響を及ぼすほどではない。また、同様に閾値に近い範囲で収束を繰り返すアルゴリズムに対しては収束回数が増えるという現象が確認されている（実験の章を参照）。

3.1.2 Desikan らの提唱する Incremental PageRank⁴⁾ による高速化手法の利用

Incremental PageRank の高速化も Web などの偏りが多いグラフに対して行われるヒューリスティック的な手法であるが、この場合、有向グラフの到達不可領域を利用して最後まで計算が行われない領域を事前に計算しておくことで計算にかかる時間を削減している。Desikan らの提唱する Incremental PageRank の手法には2つの問題点が存在する。それは、事前に到達不可領域を計算しておくというアルゴリズムを用いているため事前処理にかかるコストが高いという点と、アルゴリズムに有向グラフの到達不可領域を利用しているため、無向グラフに対して適用することができない点である。

そこで、提案手法では Desikan らの提唱する Incremental PageRank に独自の手法を適用することでそれら2つの問題を解決している。Desikan らの提唱する Incremental PageRank では事前に到達不可領域を計算する際に、値の直接変化した頂点から iterative に計算を行っているが、グラフ処理のアルゴリズムで再計算が必要となる領域は値の直接変化した頂点から iterative に拡大していくため、事前に到達不可領域を計算する必要はなく、各 iteration の計算のついでに到達可能領域を計算すれば、通信や計算の無駄なく計算を行うことができる。また、Incremental PageRank では到達不可領域を厳密に定義していたが、提案手法を用いれば計算が不要な領域を定義することができる。

例えば、最大ホップ数が極大であるグラフに対して PageRank を計算する場合、ホップを繰り返すに従って変化の伝播が閾値以下になり、全ての到達可能頂点の計算を行う前に PageRank の計算が終了してしまうという可能性は十分考えられる。その場合、計算可能領域であったが、計算に使用しなかった領域が発生する。そういった頂点群は計算が不要な領域として扱うべきであり、提案手法では計算が必要な領域をアプリケーションの計算と同時に進めているため算出することができる。計算が不要な領域は実際に計算に用いられたか用いられなかったかのみで算出されるため、有向グラフのみならず、無向グラフに対しても適用することができる。図1にその概要を示す。

3.1.3 収束の定義の変更を利用した高速化手法の利用

Yamada らの提唱する Incremental PageRank⁵⁾ では、PageRank の計算の反復回数を制限することで制限された時間内での計算性能を算出している。実際の

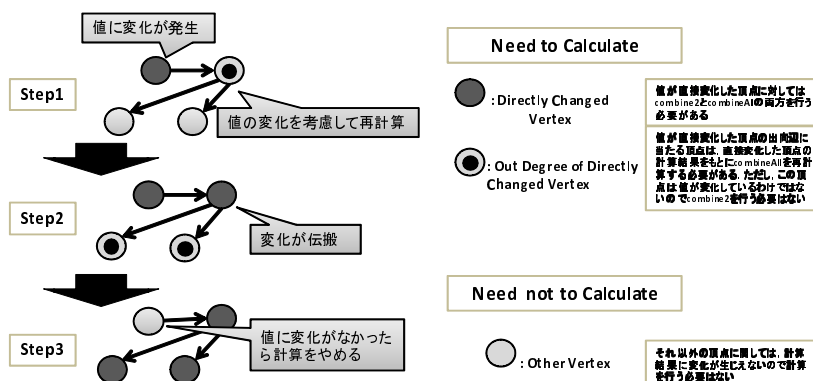


図 1 Incremental GIM-V 提案手法

インクリメンタルな処理が要求されるアプリケーションでは、計算を収束するまで行うことは少なく、制限時間内に処理を行い、結果を出力するケースが多いと思われる。提案手法では Yamada らの提唱する Incremental PageRank の手法を元に改良した設計を行っている。Yamada らの提唱する Incremental PageRank では反復回数を限定して計算を行っているが正確でない計算結果を最後まで使用してしまっている。つまり、Yamada らの提唱する Incremental PageRank をそのまま使用すると、超長期間の実行に対して精度を保つことが難しくなってしまうという問題点が存在する。そこで収束の制限がかかった頂点に対して再計算フラグを立てておき、次の計算機会の時に値の補正計算を行っている。アプリケーションの選定が難しいため、本研究ではこの手法に関して、実験、評価を行っていない。

いずれの方法も大規模グラフのヒューリスティック性を利用した高速化であるため、理論計算量自体は通常の PageRank 自体と変わらない。高速化の性能はグラフのつながりが疎であるほど得られる傾向にあるが、クラスタ性が高くてもコミュニティ構造を持つようなグラフに対しては、高速化を得ることができるので、本稿では高速化に依存する明確な変数は発見できていない。

3.2 Incremental GIM-V の実装設計

本節では、3.1 節のアルゴリズム設計を実装するための手法を提案する。アルゴリズム設計を効率的に実装するためには、計算領域の算出と計算そのものの算出を可能な限り共通な処理で実行する必要がある。無駄な通信を行わずに計算領域を算出するための手法としては Google 社のグラフ処理系 Pregel²⁾ が採用している Vertex State Machine があげられる。Vertex State Machine は変更があった頂点とその外方向の頂

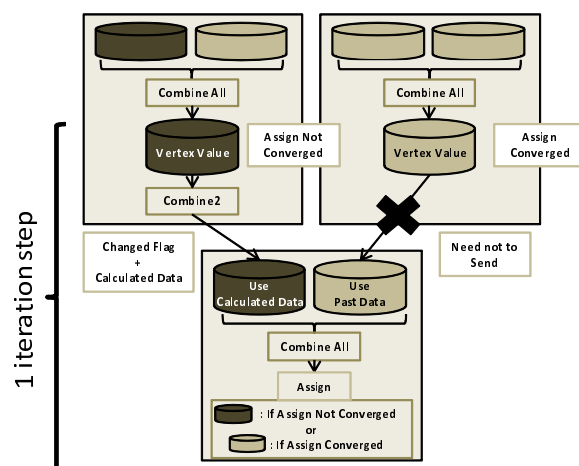


図 2 Incremental GIM-V 提案手法

点に対して処理要求を送り、処理要求が送られた頂点に対してのみ計算を行うという手法である。しかし、Pregel の Vertex State Machine は、各頂点の値の計算を行う際に以前の計算結果と送られてきた処理要求に含まれる送り元の頂点情報しか参照することができない。そのため、PageRank の様に計算にすべての in-edge である頂点の情報が必要な計算を行うことができないといった問題点が存在している。そこで、図 2 のように IGIM-V モデルでは各頂点内に前回のグラフ処理結果を格納する領域を確保することでその問題を解決している。この手法では in-edge に関する情報と out-edge に関する情報の格納が必要であり、out-edge に関する情報の格納のみが必要な既存の GIM-V と比較すると丁度 2 倍のメモリ量が必要になってしまうという問題点が存在しているが、計算領域を算出するための処理要求クエリは in-edge から送られる頂点の情報自体をトリガとすればよいため、既存の GIM-V モデルと比較して通信量が増えることはない。処理時間

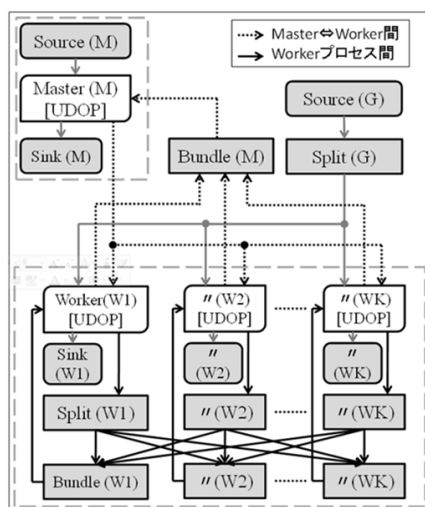


図 3 グラフ処理システムのデータフロー図

に関しても Combine2 を行い, CombineAll を行う前に配列に値を格納しておく処理を追加すればよいだけなので, 各フェイズを移行する際の Master - Worker 間の通信にかかるコストで十分隠蔽できる. 実装の各フェイズや Master, Worker については実装の章で述べる.

4. システムの実装

本システムは, IBM System S¹⁰⁾ 上で実装されている. 第 1 の理由としては既存の実装ではそもそもリアルタイムに処理を行うことができない. 既存の GIM-V モデルを実装している PEGASUS は Hadoop³⁾ を用いたオープンソースのグラフ処理システムであるが, Hadoop はファイルベースのシステムとして実装しているため, リアルタイムに処理を行うためにはファイル I/O がボトルネックになってしまう. そのため, メモリベースの実装をサポートしている処理系で実装を行う必要がある. また, 第 2 の理由としては, 本研究のようなリアルタイム性を要求される処理を行う際には, 同じくリアルタイム性を重視しているデータストリーム処理系を利用することで, 効率的に性能の高いシステムを実装することが可能であるという点があげられる. 既存のデータストリーム処理系には他にも商業用に開発されているシステムが存在しているが, いずれも SQL で実装を記述するシステムであり, グラフ処理のようなシステムを実装するためには, データフローで実装を記述できる IBM System S を使用したほうが効率的である.

4.1 データフロー図

図 3 にシステムのデータフロー図を示す. 図中の各頂点はオペレータ (プロセス) を表し, 矢印はデータの流れ (データストリーム) を表わす. 実装に用いたオペレータは, Source (データの入力), Sink (データの出力), Split (データの分割), Bundle (データストリームをまとめる) の 4 種類の組み込みオペレータと, Master (グラフ処理全体の管理), Worker (グラフ処理の並列実行) の 2 種類の UDOP (ユーザ定義オペレータ) である. 図中の各オペレータの動作について説明する. なお, 図中の K の値, Worker オペレータおよびそれに付随するオペレータ (Sink, Split, Bundle) の並列数を表わす. また, 各オペレータの処理はそれに対応するプロセスが 1 つずつ立ち上げられ, 並列実行される. これらのプロセスは複数の計算機間に渡って配置することが可能である.

- Source(G): システムへグラフデータの入力を行う. (辺の値 M_{ij} を設定する)
- Split(G): グラフデータを担当する Worker に渡す. (各 M_{ij} の値を j で分類する)
- Source(M): グラフ処理の開始命令を受け付ける.
- Master(M): グラフ処理全体の管理, 反復計算の同期の管理を行う.
- Sink(M): グラフ処理に要した時間などのログ情報を出力する.
- Worker($W_1 \sim W_K$): グラフ処理のメインとなる部分を実行する. $a (= 1, \dots, K)$ 番のオペレータは, $a = (j \bmod K + 1)$ を満たす頂点 j の情報 (v_j, M_{ij} for $\forall i$) を保持する.
- Sink($W_1 \sim W_K$): 各 Worker が担当する頂点に対するグラフ処理の計算結果を出力する.
- Split($W_1 \sim W_K$), Bundle($W_1 \sim W_K$): Worker 間のデータの流れを制御する. Worker(W_a) から Worker(W_b) へのデータの流れは, Split(W_a) と Bundle(W_b) を介する.
- Bundle(M): Worker から Master へのデータの流れを束ねる.

5. グラフ処理の流れ

グラフ処理の流れについて説明する. 既存の GIM-V モデルは MapReduce モデルを使用してグラフ処理を行っているが, 本システムは MapReduce モデルを厳密に再現しているわけではないのでグラフ処理の流れがオリジナルの GIM-V と異なっている. オリジナルの GIM-V の処理の流れは PEGASUS¹⁾ に詳細が書かれているのでそちらを参照していただきたい. ま

ず、グラフ処理の開始は Source(M) から Master へのデータの入力により行われる。処理全体の流れとしては「待機」「Combine2 計算」「変化頂点導出」「CombineAll 計算」「無変化頂点出力」「結果出力」の 6 つのフェイズからなる。このうち「Combine2 計算」「変化頂点導出」「CombineAll 計算」「無変化頂点出力」の 4 つは反復計算を行うため、複数のステップからなる。Master は、各フェイズ各ステップの開始を各 Worker に伝え、Worker 間の通信によりステップを実行し、実行が完了したことを (反復計算が収束したか否かも含めて) Master に伝える。全ての Worker からのステップの実行完了通知を Master が受け取ったら、次のステップに進む。このような流れで計算を進めていく。次にグラフ処理の各フェイズについて説明する。

- 待機：グラフデータの入力を受け付ける。待機フェイズ以外のフェイズでグラフデータの入力が行われた場合、一時的に保持しておき、実行中の処理が終わり次第グラフに反映し、次の実行フェイズに移る。グラフに反映する際に、構造が更新されたエッジの両端点を変化頂点として登録する。
- Combine2 計算：変化頂点として登録されている頂点に対して Combine2 を計算する。その後、計算した結果を隣接するエッジに対して転送する。
- 変化頂点導出：Combine2 の計算結果が転送されてきた頂点を変化頂点として登録する。
- CombineAll 計算：変化頂点として登録されている頂点に対して CombineAll, assign を計算する。
- 無変化頂点出力：CombineAll の計算の結果、値に変化がない場合、その頂点を無変化頂点として扱う。
- 結果出力：各頂点の計算結果をシステムから出力し、待機フェイズに戻る。その際に変化頂点の情報は保持しておく。

6. アルゴリズムの実装方法

Incremental GIM-V モデル (IGIM-V モデル) を計算モデルとしてみた時、インターフェースとして以下のものを提供しており、これらをアルゴリズムごとに実装する必要がある。GIM-V の演算である combine2, combineAll, assign である。このうち、PEGASUS と同様に assign は代入処理だけでなく、頂点ごとの収束判定も記述する必要がある。関数の他には、行列の要素 (辺の重み)、ベクトルの要素 (頂点の値)、および combine2 の戻り値の型 (TM, Tv, Tc) と、反復計算の上限回数 IT LIMIT を定義する必要がある。以

上を実装することで、グラフ処理のアルゴリズムを Incremental GIM-V の計算モデル上で実行できるようになる。これらのインターフェースは、Worker オペレータが参照する C++ のヘッダファイルとして提供されており、アルゴリズム実装者はヘッダファイルのテンプレートを元に、上記の関数定義、型定義およびマクロ定義を行う。アプリケーションの実装に関しては GIM-V モデル¹⁾ と Incremental GIM-V モデルは本質的に同一であるため、GIM-V ユーザはプログラムを変更することなく Incremental GIM-V モデルを使い、動的な解析を行うことができる。

6.1 Incremental GIM-V による Single Source Shortest Paths

最短経路問題¹¹⁾ は元々の GIM-V モデルでは記述されていないアプリケーションである。しかし、最短経路問題は Graph500 でカーネルとして選択されているなど、注目を集めているアルゴリズムであり、GIM-V モデルで実装可能であるため今回のアプリケーションに選択した。最短経路問題を解くアルゴリズムは Ramalingam らのアルゴリズム¹¹⁾ をベースとして実装としている。異なる点としては、GIM-V モデルでは計算の優先度を示すキュー構造 (SSSP アルゴリズムではこれを優先キューと呼んでいる) を持っていない。優先キュー構造を実装するためにはすべての頂点を一元的に管理する必要があるが、大規模分散環境ですべての頂点を一元的に管理するためには 1 対全、全対 1 の通信が必要となるため効率的ではない。計算効率は落ちるが優先キュー構造を持たせなくても SSSP は計算可能であるため、今回は優先キュー構造を定義せずに実装を行っている。最短経路問題を combine2, combineAll, assign の 3 つの演算に一般化すると combine2 は始点からの距離の計算、combineAll は最短となる経路の選択、assign は頂点情報の更新にあたる。最短経路問題の計算式を IGIM-V モデルの 3 つのオペレーションで表現すると次のようになる。

$$(1) \quad \text{combine2}(m_{i,j}, v_j) = m_{i,j} + v_j$$

$$(2) \quad \text{combineAll}_i(x_1, \dots, x_n) = \min$$

$$(3) \quad \text{assign}(v_i, v_{\text{new}}) = v_{\text{new}}$$

7. 評価

この章では前章での実装に対する評価を行う。既存の PEGASUS¹⁾ の GIM-V モデルはファイルベースである Hadoop³⁾ 上に実装しており、メモリベースの実装である IGIM-V モデルの比較対象としては適当でないので、メモリベースの GIM-V モデルを独自に System S¹⁰⁾ 上に実装して実験の比較対象として利用

した．

7.1 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AMD Phenom 9850(2.5GHz,4コア)，メモリは8GB，OSはCentOS 5.4，ソフトウェアは，InfoSphere Streams 1.2.0(System S)，gcc 4.1.2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．gccのコンパイルは全て最適化オプション“-O3”で行った．

7.2 実験に行うアプリケーション

今回の実験では対象とするアプリケーションとして，PageRank¹²⁾，Random Walk with Restart (RWR)¹³⁾，Single Source Shortest Path(SSSP)¹¹⁾を選択した．選択の理由としては，以下の3つがあげられる．

- (1) 3例とも大規模なグラフを処理する必要があり，リアルタイムな処理の需要が高いという共通点が存在する
- (2) PageRankは有向グラフを対象とし，並列性の確保が容易なアプリケーションであるのに対し，RWRは無向グラフを対象とし，並列性の確保が容易，SSSPは無向グラフを対象とし，並列性の確保が困難と，3例ともアプリケーションの特性が異なり，IGIM-Vの汎用的な性能を示すのに適している．
- (3) それぞれに適したグラフを生成するジェネレータが存在し，実験が容易である

7.3 対象とするデータ

今回の実験では入力グラフとして，Kronecker Graph¹⁴⁾と対数正規分布グラフ²⁾，それと参考のための実データを用いて実験を行った．Kronecker GraphはGraph500¹⁵⁾のベンチマークでも使われているグラフで，今回はGraph500のベンチマーク用のジェネレータであるKronecker Generatorを用いて生成する．Kronecker Graphは，大規模グラフの特徴であるクラスタ性，スケールフリー性，クラスタ性を持っており，実際のソーシャルグラフに近い性質を持っていると言える．Kronecker Graphで生成するグラフは重みなしの無向グラフであるため，RWRとSSSPの入力データとして用いている．対数正規分布グラフは，頂点の次数が対数正規分布と一致するように生成されるグラフである．L. Brechettiら¹⁶⁾による研究によると，PageRankによって計算されるノード重要度は対数正規分布に従うということが明らかになっている．また，Pregelでは対数正規分布をWebなどのソーシャルネットワークと同様の性質を保持するグラフの例として対数正規分布を挙げている．ま

た，対数正規分布グラフは有向グラフを生成することができるためPageRankの入力データとして対数正規分布グラフを用いている．本稿ではグラフの大きさをGraph500で採択されているSCALEという単位を用いて管理している．SCALEとは頂点数の指標であり，頂点数 $= 2^{SCALE}$ として計算されている．すなわち，SCALE12では頂点数は4,096，SCALE13では8,192，SCALE14では16,384となっており，SCALEが1上がるごとに頂点数は2倍となっている．辺数は生成するグラフによって異なり，表によってまとめると表1のようになる．計算時間はグラフのエッジ数に依存するため，Kronecker Graphと対数正規分布グラフのエッジ数が同程度になるようにSCALEを調整している．Incremental PageRankによる高速化を実験するためには動的に変化するグラフを生成しなければならない．今回はエッジの重みをランダムに変化させることでグラフの変化を示している．実装としては，Kronecker Graphと対数正規分布グラフの両者共に重みなしグラフを生成するためのジェネレータであるため，普段はエッジの重みを1として扱っているが，ランダムにエッジを選択し，エッジの重みを2に変化させ，差分実行を行っている．実データはWeb上で公開されている百科事典サイトWikipedia¹⁷⁾のページ間リンク情報を利用した．Wikipediaは通常の百科事典サイトと異なり，ユーザが自由に百科事典の内容を編集することができ，解説する単語と単語の間にリンク構造を張ることでグラフ構造を形成している．今回は時系列で成長するグラフ構造を持った実データの例としてWikipediaのページ間リンク構造を利用し，PageRankの計算対象として用いた．対象とするデータはWikipedia創立(2001/1/15)から2010/1/31までのデータで，2010/1/31時点で頂点数は5,918,351個，エッジ数は71,890,229個のグラフを形成している．Wikipediaの累計編集回数は約3億回であり，平均して1日に約10万回の編集が行われている計算となる．

7.4 実験

IGIM-Vモデルでは，計算の無駄を省くことで既存のグラフ処理においてもGIM-Vモデルと比較して高速化を実現している．図4, 5, 6ではグラフの規模を変化させた場合の高速化について述べている．既存のグラフ処理においてもAdaptive PageRankによる高速化が得られるため，IGIM-Vモデルのほうが高速に処理を実行していることが分かる．PageRankではSCALE16の時点で33%の高速化を達成している．同様にRWR法ではSCALE18で69%の高速化

表 1 各 SCALE における頂点と辺数の関係

Kronecker Graph	SCALE14	SCALE15	SCALE16	SCALE17	SCALE18
vertex	16,384	32,768	65,536	131,072	262,144
edge	524,288	1,048,576	2,097,152	4,194,304	8,388,608
Log-normal Graph	SCALE12	SCALE13	SCALE14	SCALE15	SCALE16
vertex	4,096	8,192	16,384	32,768	65,536
edge	522,213	1,028,991	2,027,556	4,122,395	8,364,525



図 4 SCALE 数による PageRank 実行時間比較

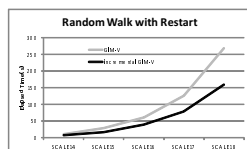


図 5 SCALE 数による RWR 実行時間比較

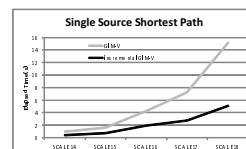


図 6 SCALE 数による SSSP 実行時間比較



図 7 各反復による PageRank 実行時間比較

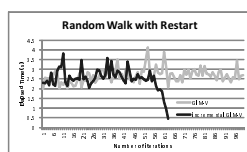


図 8 各反復による RWR 実行時間比較

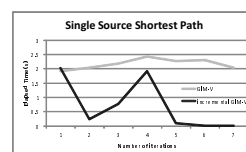


図 9 各反復による SSSP 実行時間比較

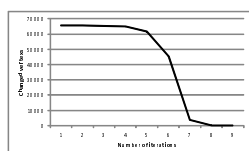


図 10 各反復による PageRank 変更点数比較

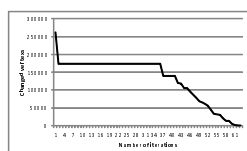


図 11 各反復による RWR 変更点数比較

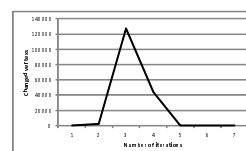


図 12 各反復による SSSP 変更点数比較

を、SSSP では SCALE18 で 198%の高速化を実現している。

次に、SCALE18(PageRank では SCALE16)のグラフの計算時間の反復による変化の違いについて比較する。図 7, 8, 9 に実験結果を示している。Adaptive PageRank の提唱通り、PageRank の収束速度は次数によって異なっており、反復の後半部分に対して無駄な計算の削除が行われ、高速化が得られている。RWR も同様であるが、それに加えて GIM-V と IGIM-V の間で収束回数も異なっている。これは、IGIM-V では隣接頂点の閾値以下の変化の合計が対象頂点の閾値以上の変化を示す時を考慮していないからである。この点に対しては、考察の章で述べる。図 10, 11, 12 では、計算を行う必要のある頂点の数についてそれぞれの反復で計測したものである。この図からも、反復の後半部分に対して無駄な計算の削除が行われ、高速化が得られていることが分かる。

次に、IGIM-V のコアによるスケーリングについて比較を行う。図 13, 14, 15 ではコア数を 4, 8, 12, 16 と変化させて実験を行い、実行時間の変化を比較している。PageRank, RWR 共にスケーリングは GIM-V と同等の性能が出ている一方で、SSSP に対しては僅か

に GIM-V に比べて IGIM-V はスケールしていない。

ここまでの実験では GIM-V と同じ処理を IGIM-V が行った場合、Adaptive PageRank の手法でどれだけの高速化が得られるかといった内容を実験してきたが、これから行う実験は GIM-V では不可能であり、IGIM-V 特有の処理である Incremental PageRank による差分逐次実行による高速化について述べる。図 16, 17 では、それぞれグラフ変化率が全体の 1%だった場合について、実行時間の変化と反復回数の変化について計測した結果である。グラフ変化率が全体の 1%だった場合、必要な実行時間はグラフ全体を処理する場合に比べて PageRank の場合 89%, RWR の場合 5%, SSSP の場合 49%に低減している。対象とするアプリケーションによって程度は異なるが、いずれの場合も差分逐次実行による高速化に成功している。なお、既存の GIM-V では差分実行をサポートしていないので、Incremental PageRank による高速化を既存 GIM-V と比較することは計算モデル的に不可能である。

以上がジェネレータによって生成されたグラフに対する評価であるが、実際のデータと比較して、どの程度の差異があるのかを実験した。図 15, 16, 17 はそ

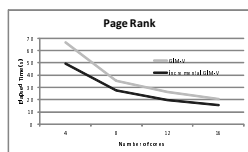


図 13 各コア数による PageRank 実行時間比較

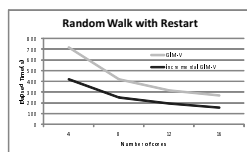


図 14 各コア数による RWR 実行時間比較

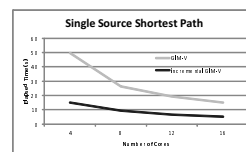


図 15 各コア数による SSSP 実行時間比較

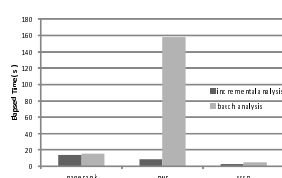


図 16 逐次実行による実行時間の変化

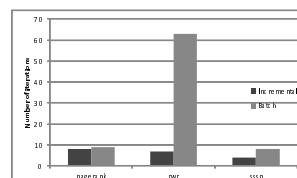


図 17 逐次実行による反復回数の変化

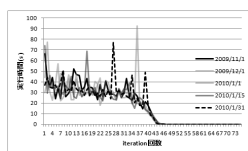


図 18 差分実行を行わない場合の実行時間推移

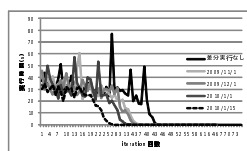


図 19 差分実行を行う場合の実行時間推移

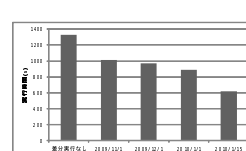


図 20 差分実行を行う場合の合計実行時間

の結果を示している．図 15 は，差分実行を行わずに PageRank の計算を行った場合の実行時間計測結果である．差分実行を行わなければ，それぞれの時点で計算時間に変化がないことがわかる．一方，差分実行による総計実行時間とバッチ的な総計処理時間を比較すると 114%の高速化がおこなわれている．図 16 に関しては，差分実行を行った際の反復ごとの実行時間を測定したものである．図 16, 17 では，いずれも図中の系列の示す時刻時点の計算結果から 2010/1/31 までの PageRank を差分計算している．差分が少ないほど収束までの反復回数は減っており，データストリームの細粒度で逐次実行を行えば微分関数的に反復回数を減らせる可能性がある．正確に評価するならば，実データに対しても既存の GIM-V 処理系と比較して評価をするべきであるが，枚数の関係上割愛し，単純に差分実行による高速化のみ記述する．

計算の無駄を省くことによる高速化と逐次差分実行による高速化の両者の結果を合わせると，合計で PageRank では 48%，RWR では 3023%，SSSP では 501%の高速化を実現した．また実データを用いて差分計算を行い 114%の高速化を実現した．

8. 考 察

8.1 アプリケーションによるスケーリングの違い

SSSP は PageRank や RWR 法と比較すると並列して行える計算の数が少ないアプリケーションである．

GIM-V モデルでの計算ではグラフ全体に対して計算を繰り返しているため，SSSP では無駄となる計算が多い．一方で IGIM-V では無駄な計算を行わないため GIM-V と比較すると高速化が著しいが，計算の並列数が少ないため，無駄な計算を省略すると，大規模処理系の性能を十分引き出すことができなく，結果としてスケーリング性能が落ちてしまう．

8.2 GIM-V モデルとの収束判定の違い

既存の GIM-V モデルでは各頂点について収束判定を行い，すべての頂点が収束した時点で計算を終了している．この際，たとえ一部の頂点が収束していてもすべての頂点が収束していないと判定され，すべての頂点に対して，再計算を行う．一方，IGIM-V モデルでは Adaptive PageRank の手法を利用して，収束した頂点を計算対象から除外するという手法をとっている．この手法では，収束した頂点に閾値以下の変化を後の計算で無視するという特徴がある．この特徴をもっともよく示しているのは，図 9 である．Tong らの論文¹³⁾によると RWR 法は反復による計算では収束が行われにくいと言う問題点が指摘されている．これは，閾地付近で細かな値の変化を繰り返しているため，閾地付近の計算を各頂点ごとに打ち切る RWR 法では GIM-V モデルと比較して収束回数が異なってしまう．GIM-V モデルでは特定頂点群にのみ反復計算を行うことができないため，同様の収束回数で計算時間の比較を行うことはできない．

9. ま と め

メモリベースであるデータストリーム処理系で、既存の大規模グラフ処理系 PEGASUS¹⁾ に使用されている GIM-V モデルを改良し、時系列上で逐次実行可能な IGIM-V モデルを定義した。以下に結論を示す。

- (1) 提案手法である IGIM-V モデルでは、GIM-V モデルの特色であるグラフ処理を行列ベクトル積とみなす計算モデルを損なわずに、時系列上で変化する大規模グラフに対して汎用的に逐次実行可能とした。
- (2) 大規模グラフ処理の逐次実行を達成するため、大規模グラフ処理系をメモリベースであるデータストリーム処理系に実装した。Hadoop³⁾ 上に実装した PEGASUS と比較してよりリアルタイムに処理を行うことが可能になった。
- (3) 処理の逐次実行に加え、計算を効率的に行うことで既存の GIM-V モデルと比較して高速化を実現した。なお、その際に使用した GIM-V モデルは既存のファイルベースである Hadoop 上で実装した PEGASUS ではなく、メモリベースである System S 上に定義した GIM-V モデルであり、対等な条件で比較している。

参 考 文 献

- 1) Kang, U., Tsourakakis, C. E. and Faloutsos, C.: PEGASUS: A Peta-Scale Graph Mining System- Implementation and Observations (2009).
- 2) 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).
- 3) Apache Hadoop: <http://hadoop.apache.org/>
- 4) 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).
- 5) 山田雅信, 高橋俊行, 田浦健次郎, 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).
- 6) Kamvar, S., Haveliwala, T. and Golub, G.: Adaptive Methods for the Computation of PageRank, Technical report, STANFORD UNIVERSITY (2003).
- 7) Neumeyer, L., Robbins, B., Nair, A. and Kesari, A.: S4: Distributed Stream Computing Platform, *Data Mining Workshops, International Conference on*, IEEE Computer Society, pp. 170–177 (2010).
- 8) 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).
- 9) Li, Z.: CS848 Presentation Report TelegraphCQ: Continuous Dataflow Processing for an Uncertain World (2006).
- 10) 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).
- 11) Ramalingam, G. and Reps, T.: An Incremental Algorithm for a Generalization of the Shortest-Path Problem, *Journal of Algorithms*, pp. 267–305 (1992).
- 12) Brin, S. and Page, L.: The Anatomy of a Large-Scale Hypertextual Web Search Engine, *COMPUTER NETWORKS AND ISDN SYSTEMS*, Elsevier Science Publishers B. V., pp. 107–117 (1998).
- 13) 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).
- 14) Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C. and Ghahramani, Z.: Kronecker Graphs: An Approach to Modeling Networks, *J. Mach. Learn. Res.*, pp. 985–1042 (2010).
- 15) Graph500: <http://www.graph500.org/>
- 16) Becchetti, L. and Castillo, C.: The distribution of pagerank follows a power-law only for particular values of the damping factor, *In Proceedings of the 15th international conference on World Wide Web*, ACM Press, pp. 941–942 (2006).
- 17) Wikipedia: <http://ja.wikipedia.org/>