分散グラフ処理におけるグラフ分割の最適化

藤森　俊匡† 　塩川　浩昭†† 　鬼塚　真†††

† 大阪大学電子情報工学科　〒 565–0871 大阪府吹田市山田丘 2-1
†† 日本電信電話株式会社 NTT ソフトウェアイノベーションセンタ　〒 180–8585 東京都武蔵野市緑町 3-9-11
††† 大学院情報科学研究科　〒 565–0871 大阪府吹田市山田丘 2-1
E-mail: †{fujimori.toshimasa,onizuka}@ist.osaka-u.ac.jp, ††shiokawa.hiroaki@lab.ntt.co.jp

あらまし　現実世界で見られるグラフ構造のデータが大規模になるにつれ，それらのデータを処理するための分散グラフ処理フレームワークに対する需要が高まっている．分散グラフ処理フレームワークは入力となるグラフを分割し各計算機に割り当ててから目的の分析処理を行うが，グラフの分割がどのようになされたかによってその後の分析処理に要する時間は左右される．既存のグラフ分割手法には，グラフを高速に分割し，かつその後の分析処理も高速となるようなものが存在しない．そこで本稿では，グラフを高速に分割しかつ分析処理も高速に行えるようなグラフ分割手法を提案し，それを既存の分散グラフ処理フレームワークである PowerGraph に組み込むことで大規模グラフを高速に処理する．提案手法は高速にグラフをクラスタ分割する技術を拡張することで，各計算機に配置されるエッジ数の等粒度化，切断されるバーテックス数の削減を実現する．そして，提案手法により分析処理に要する時間を既存のグラフ分割手法である random に対し最大 2.6 倍，oblivious に対し最大 1.2 倍高速化可能であることを示した．

キーワード　グラフ分割，グラフマイニング，分散処理

1. はじめに

近年，現実世界においてソーシャルグラフや Web グラフなどの大規模なグラフ構造のデータが見られるようになった．例えば，2013 年末の Facebook のユーザ数は 12 億 3000 万人であり，その一年間でのユーザ増加数は 1 億 7000 万人であったと報告されている (注1)．これに伴い，大規模なグラフ構造のデータに対する分析処理を行うための分散グラフ処理フレームワークへの需要が高まっている．

分散グラフ処理フレームワークとして，Pregel [1]，GraphLab [2] や PowerGraph [3] が挙げられる．これらのフレームワークは，Bulk Synchronous Parallel（BSP）という計算モデルに基づいて設計されている．BSP モデルは並列処理を実現するためのモデルの一つである．BSP モデルは，各計算機でローカルの処理を行い，その計算結果を通信によって交換しあい，全ての計算機が通信を終えるまで待機するという一連の処理を繰り返すことで目的とする分析処理を行う．

分散グラフ処理フレームワークは入力となるグラフが大規模となることから，まず入力となるグラフを分割し，得られた部分グラフを各計算機に割り当てる．その後，BSP モデルに基づいて分析処理を行う．各計算機は自身に割り当てられた部分グラフのバーテックス一つ一つに対しユーザ定義の処理を実行し，その後全計算機が計算結果を交換することによって計算結果の整合性を保つということを繰り返す．分析処理に要する時間はグラフがどのように分割されたかによって左右される．例えば，各計算機に割り当てられたタスク量が偏っている場合，各計算機が分析処理に要する時間にも偏りが生じる．分散グラフ処理フレームワークにおいては，多くの場合割り当てられたエッジ数が多いほどタスク量は増大する [3]．したがって，各計算機に割り当てられるタスク量の偏りを小さくするためには，各計算機に割り当てられるエッジ数の偏りを小さくする必要がある．ここで，本論文では各部分グラフに内包されるエッジ数の偏りの小ささを評価する観点として等粒度性を用いる．また，分散グラフ処理フレームワークでは分析処理中にグラフ分割によって切断されたバーテックスあるいはエッジを通じて通信が行われるため，グラフ分割により切断されるバーテックスあるいはエッジの数が多いと通信コストが高くなる．したがって，計算機間の通信コストを低くするためには，グラフ分割によって切断されるバーテックスまたはエッジの数を少なくする必要がある．本論文では，グラフ分割によって切断されるバーテックスあるいはエッジの数がどれほど多いかを表す観点として，replication factor [3] を用いる．

分散グラフ処理フレームワークにおいて使用されるグラフ分割方法は，大きく分けて二種類存在する．ひとつは，グラフ分割アルゴリズムを用いる方法である．グラフ分割アルゴリズムはグラフ全体の構造を利用して高い等粒度性と低い replication factor を実現することができるが，扱うデータが大規模になるとグラフの分割に膨大な実行時間を要するという欠点がある．もうひとつの方法として，グラフ全体の構造を利用することなく，バーテックスやエッジが読み込まれた時に割り当て先となる計算機を逐次決定するという方法がある．この方法は高速にグラフを分割でき，統計情報を用いることで高い等粒度性を実現することができるが，一方で replication factor が高くなってしまうという欠点がある．

したがって，本論文では大規模なグラフ構造のデータを高速

(注1)：http://www.theguardian.com/technology/2014/feb/04/facebook-10-years-mark-zuckerberg

に分割し，かつ高い等粒度性と低い replication factor を実現するグラフ分割手法を採用し [5]，これを PowerGraph に組み込むことで大規模なグラフを高速に分析処理するフレームワークを提案する．なお，本稿で提案するグラフ分割手法は，他の分散グラフ処理フレームワークにも適用可能である．

提案手法は，2 つのステップを経てグラフを計算機台数と同じ k 個のグラフに分割する．具体的な手順は以下の通りである．まず第一ステップで，クラスタ間の等粒度性を保ちつつ，マージした際にグラフの Modularity [6] が高くなるようにクラスタをマージする．Modularity はよいクラスタ分割ができているかを計る指標のひとつであり，この値を大きくすることで replication factor を低くすることができる．このステップでは，最終的に得られるクラスタの等粒度性を高めるために k 個よりも多いクラスタを導出する．第二ステップでは，最終的に得られるクラスタの等粒度性が高くなるようにクラスタの組をマージする．また，隣接するクラスタ同士をマージすることで replication factor を低くする．第一ステップに用いられている手法 [4] が高速であり，かつ第二ステップでの処理はマージするべきクラスタ数が第一ステップの数百〜数千分の一であることから，この手法は全体として高速な処理を実現する．本稿では，提案手法を PowerGraph に組み込み，実データと人工データを用いた評価実験を行った．その結果，本手法により分析処理を既存のグラフ分割手法である random に対し最大 2.6 倍，oblivious に対し最大 1.2 倍高速化可能であることを確認した．

本稿の構成は以下の通りである．2. 節で本稿の前提となる知識について概説する．3. 節で提案手法の詳細について説明し，4. 節にて提案手法の評価と分析を行う．5. 節で関連研究について述べ，6. 節で本稿をまとめ，今後の課題について述べる．

2. 前提知識

2.1 PowerGraph

本稿では，提案するグラフ分割手法を PowerGraph [3] に組み込むことで大規模グラフデータを高速に分析処理するフレームワークを提案する．よって本節では，既存の分散グラフ処理フレームワークである PowerGraph について概説する．

PowerGraph は Pregel [1] と GraphLab [2] を参考に設計，実装された分散グラフ処理フレームワークである．PowerGraph は各バーテックスに対し vertex-program と呼ばれるユーザ定義の処理を繰り返し実行することで目的の分析結果を得る．vertex-program の処理時間は，多くの場合対象とするバーテックスの次数が大きいほど長くなる [3]．したがって，各計算機に割り当てられるエッジ数に偏りがあると，各計算機の処理時間に偏りが生じることが予想される．PowerGraph は，他の計算機に比べ早く処理が終了した計算機が存在する場合，全ての計算機が処理を終えるまでそれらを待機させる．これは，処理が短時間で終わった計算機の計算リソースを持て余すことになり非効率的である．したがって，PowerGraph は各計算機の処理時間の偏りを小さくするために，分割されたグラフが内包するエッジ数が同程度になるようにグラフを分割する．

PowerGraph は，エッジを計算機に割り当てバーテックスを切断する vertex-cut によってグラフを分割する．切断されたバーテックスは異なる部分グラフをまたがることになるが，PowerGraph はそれらの部分グラフが割り当てられる全計算機において切断されたバーテックスの複製を作成して管理する．あるバーテックスが切断されたことにより作成された複製をそのバーテックスのレプリカと呼ぶ．レプリカにはコピーとマスターが存在する．PowerGraph では，データの整合性を保つために分析処理中にコピーとマスター間で通信が行われる．したがって，計算機上に存在するレプリカの数が増えるほど通信コストは増大する．PowerGraph では，グラフ分割の指標の一つとして replication factor と呼ばれる指標を定義している．replication factor とは，計算機上に存在する合計レプリカ数をグラフが内包する合計バーテックス数で割ったものである．PowerGraph はグラフを分割するにあたり，replication factor が低くなることを目標としている．

PowerGraph におけるグラフ分割手法について説明する．PowerGraph は対象とするデータが大規模であることから，バーテックスやエッジのデータが読み込まれると割り当て先となる計算機を逐次決定することでグラフを分割する．PowerGraph は vertex-cut によってグラフを分割するため，エッジの割り当て先をどのように決めるかはグラフをどのように分割するかと同義である．PowerGraph におけるグラフ分割手法は，大きく分けて二種類存在する．一つ目は random と呼ばれる手法である．これは，エッジの割り当て先をランダムに決定する手法である．この手法は高速にグラフを分割することができるが，グラフ全体の構造を利用しているわけではないため切断されるバーテックスの数が多くなり，replication factor が高くなる．二つ目は oblivious と呼ばれる手法である．これは，replication factor が低くなるようにエッジの割り当て先を決める手法である．エッジが割り当てられた計算機には，そのエッジの両端のバーテックスのレプリカが作成される．したがって，oblivious では各計算機にどのバーテックスのレプリカが作成されるかを記憶しておき，それを元に作成されるレプリカの数が少なくなるようにエッジの割り当て先を決める．この手法は random に比べグラフ分割に要する時間が長くなるものの，replication factor を低くすることで通信コストが削減され，全体の処理でみると random より高速となる場合がある．どちらの手法も高速にグラフを分割でき等粒度性を高くすることが可能であるが，グラフ分割アルゴリズムを用いてグラフを分割した場合よりも replication factor が高くなるであろうことが予想される．oblivious は replication factor が低くなるようにエッジを各計算機に割り当てるが，バーテックスの割り当て方法はランダムであるためグラフ分割による切断バーテックスの数を減らすのには限界があると考えられる．

PowerGraph は，現実世界にあるような大規模グラフをグラフ分割アルゴリズムで分割することは困難である [7] という前提のもと上記の手法を採用している．そこで本稿では，グラフを高速にグラフをクラスタ分割する技術 [4] を拡張した高速なグラフ分割手法を提案する．

2.2 Modularity

提案手法における第一ステップでは，クラスタ間の等粒度性を保ちながらグラフの Modularity [6] が高くなるようにクラスタをマージする．そこで，本節ではクラスタリング指標の一つである Modularity について概説する．

Modularity はクラスタに内包されるエッジが密であり, クラスタ間に存在するエッジが疎となるほど高い数値を示す指標である．グラフ分割により得られるクラスタの集合を C，クラスタ i からクラスタ j へ接続されているエッジ数を e_{ij}, グラフ内の合計エッジ数を m とした時，Modularity$Q$ は以下の式で定義される．

$$Q = \sum_{i \in C} \left\{ \frac{e_{ii}}{2m} - \left(\frac{\sum_{j \in C} e_{ij}}{2m} \right)^2 \right\} \tag{1}$$

第一ステップではマージするクラスタの組を決定する際に，そのクラスタの組をマージした場合の Modularity の変化量のみを計算する．Blondel らは，上記の Modularity の定義式から 2 つのクラスタ i, j を統合した際の Modularity の変化量 ΔQ を導出し, 以下のように定義している [8].

$$\Delta Q = 2 \left\{ \frac{e_{ij}}{2m} - \left(\frac{\sum_{k \in C} e_{ik}}{2m} \right) \left(\frac{\sum_{k \in C} e_{jk}}{2m} \right) \right\} \tag{2}$$

第一ステップでは上記の ΔQ の値とそれぞれのクラスタが内包するエッジ数の比率を合成した指標を用いる．詳しくは 3.1 節で述べる．

3. 提 案 手 法

本稿で提案する手法は，グラフを高速に分割し，かつ高い等粒度性と低い replication factor を実現する．本手法は，2 つのステップを経てグラフを k 個のクラスタに分割する．一つ目のステップは Modularity クラスタイリングステップである．このステップでは，クラスタの等粒度性を保ちつつ Modularity が高くなるようにクラスタをマージしていく．また，最終的に得られるクラスタの等粒度性を高めるために k 個よりも多いクラスタを導出する．二つ目のステップは等粒度クラスタリングステップである．このステップでは，最終的に得られるクラスタの等粒度性が高くなるように隣接するクラスタをマージしていく．以下 3.1 節，3.2 節で詳細を述べる．なお，この手法はバーテックスを各クラスタに割り当てエッジを切断する edge-cut によりグラフを分割する．PowerGraph に組み込むには，最終的に得られた edge-cut のクラスタを vertex-cut のクラスタに変換する必要がある．本手法の PowerGraph 上における実装方法については 3.3 節で述べる．

3.1 Modularity クラスタリングステップ

Modularity クラスタリングステップでは，クラスタの等粒度性を保ちつつ Modularity が高くなるようにクラスタをマージしていく．そのために，サイズが同程度であり，かつマージすることでグラフの Modularity が高くなるようなクラスタの組を選択しマージする．ここで，クラスタのサイズとはそのクラスタが内包するエッジ数の多さのことをいう．サイズが同程度のクラスタをマージすることで，各クラスタのサイズの増大の偏りを抑えることができる．また，Modularity はクラスタ間に存在するエッジが疎であるほど高い数値を示すので，Modularity を高くすることで切断エッジ数を削減し，replication factor を低くすることができる．このようなクラスタの組をマージするために，マージ候補である 2 つのクラスタ i, j に対する評価関数を以下のように定義する．

$$\Delta Q' = min \left(\frac{e_{ii}}{e_{jj}}, \frac{e_{jj}}{e_{ii}} \right) \times \Delta Q \tag{3}$$

ここで，ΔQ は 2.2 節で述べた Modularity の変化量（(2) 式）である．Modularity クラスタリングステップでは，上記の $\Delta Q'$ の値が最も大きくなるようなクラスタの組を選択しマージする．$min \left(\frac{e_{ii}}{e_{jj}}, \frac{e_{jj}}{e_{ii}} \right)$ は，クラスタ i, j の内包するエッジ数の差が小さいほど高い数値となる．したがって，$\Delta Q'$ の値を大きくなるようなクラスタの組をマージすることで，等粒度性を保ちつつ Modularity を高くすることができる．

Modularity クラスタリングステップでは，あるクラスタとマージするべきクラスタが選択された時点でクラスタをマージし，かつ最初に次数の少ないクラスタからマージ判定を実行する手法 [4] を採用することで，Louvain 法 [8] のように複数クラスタをまとめてマージし，かつマージ判定の際に参照するエッジの数を削減する．これにより，ランダムにクラスタをマージしていく場合よりも高速にグラフをクラスタ分割することができる．Modularity クラスタリングステップでは最終的に得られるクラスタの等粒度性を高めるために k 個よりも多いクラスタを導出する．これは，クラスタが k 個になるまでこの手法でマージを続けてしまうと，グラフ内の合計バーテックスの数が k で割り切れない場合，マージした回数が一回多いものとそうでないものとで内包するエッジ数の差が大きくなってしまうからである．

Modularity クラスタリングステップの処理の流れをアルゴリズム 1 に示す．また，主な記号の定義を表 1 に示す． このアルゴリズムは，入力としてグラフデータ $G(V, E)$ と最終的に得たいクラスタの数 k，そして任意の 1 以上の数 a を受け取る．a は閾値を決定するための数値である．このアルゴリズムは，クラスタの数が $a * k$ 個となるか，Modularity の値がそれ以上上がらなくなるまでクラスタの組をマージする処理（4 行〜28 行）を繰り返す．具体的には，最初に現在存在するクラスタを全て ws に加え（3 行と 9 行），wc からクラスタを一つ取り出し（12 行），そのクラスタとマージするべきクラスタを選択しマージする処理（13 行〜27 行）を ws が空になるまで繰り返す．ws が空になった場合は，現在の ws が処理される間にクラ

表 1　アルゴリズム 1 で使用されている主な記号の定義

$G(V, E)$	グラフデータ．V はバーテックス集合，E はエッジ集合
k	最終的なクラスタの数
a	閾値を決定するための 1 以上の定数
C_i	現在存在するクラスタの集合
ws	マージ判定の対象となるクラスタの集合
n_{prev}	現在の ws がセットされる前のクラスタの数
$inner\_edge(c)$	クラスタ c が内包するエッジの数
$m(c)$	クラスタ c とマージするべきクラスタ
$adjacent\_clusters(c)$	クラスタ c と隣接するクラスタの集合
$eval\_func(c, c_a)$	2 つのクラスタ c, c_a に対する評価関数 ((3) 式)

Algorithm 1 Modularity クラスタリングステップの流れ

Input: $G(V,E), k, a$
Output: C_i

$C_i \leftarrow V$
$ws \leftarrow C_i$
$n_{prev} \leftarrow |C_i|$
while $|C_i| > a * k$ **do**
  if $ws = \emptyset$ **then**
    if $n_{prev} = |C_i|$ **then**
      break
    else
      $ws \leftarrow C_i$
      $n_{prev} \leftarrow |C_i|$
    end if
  end if
  get cluster $c \in ws$, delete c from ws
  if $inner\_edge(c) \geqq |E|/k$ **then**
    continue
  end if
  $C_a(c) \leftarrow adjacent\_clusters(c)$
  $\Delta Q'(c) \leftarrow 0$
  $m(c) \leftarrow c$
  for each $c_a \in C_a(c)$ **do**
    $\Delta Q'(c, c_a) \leftarrow eval\_func(c, c_a)$
    if $\Delta Q'(c, c_a) > \Delta Q'(c)$ **then**
      $\Delta Q'(c) \leftarrow \Delta Q'(c, c_a)$
      $m(c) \leftarrow c_a$
    end if
  end for
  if $m(c) \neq c$ **then**
    merge cluster c and $m(c)$, generate cluster $(c, m(c))$
    delete c and $m(c)$ from C_i, insert $(c, m(c))$ into C_i
  end if
end while
return C_i

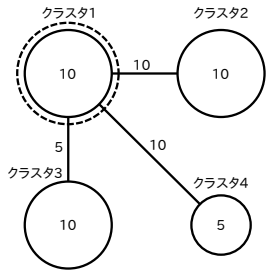


図 1　Modularity クラスタリングステップのマージ例

スタの総数に変化がないかを評価し（6 行），なければループから抜け出す．

Modularity クラスタリングステップにおけるマージの例を図 1 に示す．図中の円がクラスタを表し，円内の数値がクラスタの内包するエッジ数を，線分上の数がクラスタ間に存在するエッジ数を表す．例ではクラスタ 1 とマージするクラスタを決定したいとする．(3) 式によりクラスタ 1 と各隣接クラスタとの $\Delta Q'$ を求めると，クラスタ 2 は 0.069，クラスタ 3 は 0.010，クラスタ 4 は 0.047 となる．よって，クラスタ 1 とクラスタ 2 がマージされる．このように，Modularity クラスタリングステップではサイズが同程度であり，かつクラスタ間のエッジ数が多いクラスタの組がマージされる．

3.2　等粒度クラスタリングステップ

等粒度クラスタリングステップでは，最終的に得られるクラスタの等粒度性が高くなるようにクラスタをマージしていく．そのために，まずサイズの大きいクラスタ上位 k 個を選択し，各クラスタを隣接するクラスタとマージさせる．サイズの大きいクラスタからマージを始めることで，等粒度クラスタリングステップの後半においてクラスタの等粒度性が大きく崩れるのを避ける．また，隣接するクラスタ同士をマージすることでクラスタ間のエッジ数を削減し，replication factor を低くすることができる．最初に選択された k 個のクラスタからエッジを通じて辿りつけない独立したクラスタは，k 個のクラスタのうちサイズの最も小さいクラスタとマージさせる．そうすることで等粒度性を高めることができる．等粒度クラスタリングステップは，マージするべき組が Modularity クラスタリングステップに比べ数百分の一であるため高速に実行できる．

Algorithm 2 等粒度クラスタリングステップの流れ

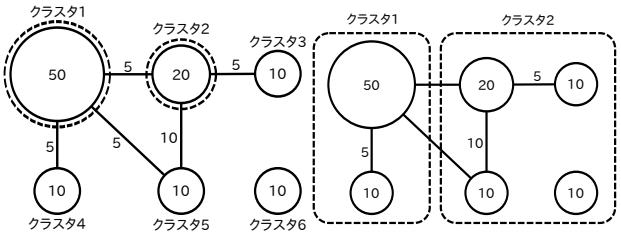
Input: $G(V,E), C_i, k$
Output: C_o

$C_o \leftarrow top\_cluster(k)$
delete $\forall c \in C_o$ from C_i
$c_{min} \leftarrow min\_cluster(C_o)$
$n_{prev} \leftarrow |C_i|$
while $C_i \neq \emptyset$ **do**
  if $c_{min} \neq min\_cluster(C_o)$ **then**
    $c_{min} \leftarrow min\_cluster(C_o)$
  end if
  $C_a(c) \leftarrow adjacent\_clusters(c)$
  while $c_{min} = min\_cluster(C_o)$ **and** $C_a(c_{min}) \neq \emptyset$ **do**
    $m(c_{min}) \leftarrow min\_cluster(C_a(c_{min}))$
    merge cluster c_{min} and $m(c_{min})$, generate cluster $(c_{min}, m(c_{min}))$
    delete $m(c_{min})$ from C_i and $C_a(c_{min})$
    delete c_{min} from C_o, insert $(c_{min}, m(c_{min}))$ into C_o
  end while
  if $n_{prev} = |C_i|$ **then**
    break
  else
    $n_{prev} \leftarrow |C_i|$
  end if
end while
while $C_i \neq \emptyset$ **do**
  $n_{prev} \leftarrow |C_i|$
  for each $c \in C_i$ **do**
    $C_a(c) \leftarrow adjacent\_clusters(c)$
    $E_n(c) \leftarrow |E|$
    $m(c) \leftarrow c$
    for each $c_a \in C_a(c)$ **do**
      $E_n(c, c_a) \leftarrow calc\_edge(c, c_a)$
      if $E_n(c) > E_n(c, c_a)$ **then**
        $E_n(c) \leftarrow E_n(c, c_a)$
        $m(c) \leftarrow c_a$
      end if
    end for
    if $m(c) \neq c$ **then**
      merge cluster c and $m(c)$, generate cluster $(c, m(c))$
      delete c from C_i
      delete $m(c)$ from C_o, insert $(c, m(c))$ into C_o
    end if
  end for
  if $n_{prev} = |C_i|$ **then**
    break
  else
    $n_{prev} \leftarrow |C_i|$
  end if
end while
while $C_i \neq \emptyset$ **do**
  $c \leftarrow C_i$
  $m(c) \leftarrow min\_cluster(C_o)$
  $G_s(c) \leftarrow aggregate\_cluster(c)$
  for each $c_a \in G_a(c)$ **do**
    merge cluster c_a and $m(c)$, generate cluster $(c_a, m(c))$
    delete c_a from C_i
    delete $m(c)$ from C_o, insert $(c_a, m(c))$ into C_o
  end for
end while
return C_o

等粒度クラスタリングステップの処理の流れをアルゴリズム 2 に示す．また，主な記号の定義を表 2 に示す． このアルゴリズムは，入力としてグラフデータ $G(V,E)$，Modularity クラスタリングステップの出力として得られたクラスタ集合 C_i と最終的に得たいクラスタの数 k を受け取る．最初に，C_i からサイズの大きいクラスタ上位 k 個を取り出し，C_o に加える．C_o に

表 2　アルゴリズム 2 で使用されている主な記号の定義

$G(V, E)$	グラフデータ．V はバーテックス集合，E はエッジ集合
C_i	Modularity クラスタリングステップの出力クラスタ集合
k	最終的なクラスタの数
C_o	最終的に導出されるクラスタの集合
$top\_cluster(k)$	C_i 内のサイズの大きなクラスタ上位 k 個
$min\_cluster(C_o)$	クラスタ集合 C_o からサイズが最も小さいクラスタ
n_{prev}	現在のマージ処理が始まる前のクラスタの数
$adjacent\_clusters(c)$	クラスタ c と隣接するクラスタの集合
$inner\_edge(c)$	クラスタ c が内包するエッジの数
$m(c)$	クラスタ c とマージするべきクラスタ
$calc\_edge(c, c_a)$	クラスタ c, c_a をマージした場合のクラスタが内包するエッジ数
$aggregate\_cluster(c)$	クラスタ c からエッジを通じて辿りつける全クラスタの集合



(a) 初期状態　　(b) 終了状態

図 2　等粒度クラスタリングステップのマージ例

属する k 個のクラスタを隣接するクラスタとマージさせていくことによって，最終的に k 個のクラスタを導出する．このアルゴリズムは，大きく分けて 3 つの処理から構成される．まず一つ目の処理で，C_o 内で最もサイズの小さなクラスタ c_{min} を取得し，c_{min} が C_o 内で最小でなくなるまで隣接するクラスタとマージする処理（5 行〜28 行）を繰り返す．c_{min} が最小でなくなった場合，また新たに C_o 内からサイズが最小のクラスタを取得し，上の処理を繰り返す．これにより，C_o に属するクラスタの等粒度性を維持しながらクラスタをマージさせていくことができる．以降の処理では，この一つ目の処理でマージできなかった C_i 内のクラスタをマージする．二つ目の処理では，C_o 内のクラスタからエッジを通じて辿りつけるクラスタのマージを行う（29 行〜54 行）．C_i からクラスタ c を取り出し，c と隣接するクラスタに C_o に属しているものがあれば，その中からサイズが最小のクラスタを選択し c とマージする．この処理は C_i に属するクラスタの数が変化しなくなるまで繰り返される．三つ目の処理では，C_o 内のクラスタからエッジを通じて辿りつけないクラスタのマージを行う（55 行〜69 行）．C_i からクラスタを一つ取得し，そのクラスタからエッジを通じて辿りつける全クラスタと，C_o に属するサイズが最小のクラスタとをマージする．この処理は C_i に属するクラスタが存在しなくなるまで繰り返される．

等粒度クラスタリングステップにおけるマージの例を図 2 に示す．例では，C_o に属するクラスタはクラスタ 1 と 2 であり，グラフ全体を 2 個のクラスタに分割したいとする．最初に，C_o 内でサイズが最小であるクラスタ 2 を取得する．添字の数値が小さいクラスタからマージを行うとすると，まずクラスタ 2 とその隣接クラスタであるクラスタ 3 がマージされる．それによりクラスタ 2 の内包するエッジ数が 35 となるが，まだ C_o 内でサイズが最小であるためマージ処理を続けクラスタ 5 とマージする．その結果クラスタ 2 の内包するエッジ数が 55 となりクラスタ 1 の内包するエッジ数を上回るので，今度はクラスタ 1 とクラスタ 4 がマージされ，内包するエッジ数が 65 となる．C_o に属するクラスタから辿りつけるクラスタとのマージは終了したので，独立したクラスタであるクラスタ 6 と，C_o 内でサイズが最小であるクラスタ 2 をマージする．

3.3　PowerGraph への組み込み

本節では，提案手法の PowerGraph 上における実装方法について説明する．

3.3.1　PowerGraph のグラフ分割方法

まず，PowerGraph のグラフ分割方法について説明する．PowerGraph では，バーテックスやエッジのデータを add_vertex 関数と add_edge 関数と呼ばれる関数によって読み込む．add_vertex 関数はバーテックスのデータとそのバーテックスの ID を引数として受け取る．また，add_edge 関数はエッジのデータとそのエッジの両端のバーテックスの ID を引数として受け取る．これらの関数は，データが読み込まれるとその関数内で割り当て先となる計算機を決定する．PowerGraph では，各計算機に 0 から始まる連続整数番号がつけられており，その数値によって計算機を区別している．add_vertex 関数や add_edge 関数によって読み込まれたデータは，その割り当て先の番号と一緒にバッファに保存される．また，バッファのサイズがある閾値より大きくなると，バッファ内のデータは自動で割り当て先に送信される．データが全て読み込まれると，各計算機は finalize 関数と呼ばれる関数によってバッファ内のデータを割り当て先に送信する．そして，各計算機は他の計算機から受け取ったデータからローカルグラフを作成する．この時，エッジを割り当てられた計算機上ではそのエッジの両端のバーテックスのレプリカが作成されるが，これらのレプリカは作成された時点ではバーテックスの ID だけを所持している．したがって，バーテックスのデータを所持するために，データを持っているレプリカのマスターと通信を行う必要がある．PowerGraph では，バーテックスの割り当て先を決める際に，ID に対してハッシュ関数を実行した結果のハッシュ値を用いる．ハッシュ値を計算機の数で割り，その剰余と同じ番号の計算機にバーテックスを割り当てる．そのため，各計算機はバーテックスの ID に対してハッシュ関数を実行することでマスターを管理している計算機を知ることができる．

3.3.2　提案手法を用いたグラフ分割方法

次に，PowerGraph 上における提案手法の実装方法について説明する．提案手法によりグラフを分割するためには，全てのバーテックス，エッジのデータを記憶しておく必要がある．よって，add_vertex 関数や add_edge 関数によって読み込まれたデータをローカルに保存しておき，finalize 関数内でグラフ分割を実行しそれらの割り当て先を決定し送信するようにプログラムを書き換えた．

提案手法を組み込んだ PowerGraph において，エッジとバーテックスの割り当て先をどのように決定するかについて説明する．まず入力となるグラフを提案手法により計算機台数と同じ数のクラスタに分割し，各クラスタに 0 から始まる連続整数番号をつける．提案手法はバーテックスを各クラスタに割り当てる edge-cut によってグラフを分割する．したがって，バーテッ

表 3　比較実験で用いたデータセット

データ名	$\|V\|$	$\|E\|$	$\|E\|/\|V\|$
web-Google	875,713	5,105,039	5.87958
wiki-Talk	2,394,385	5,021,410	2.09716
soc-LiveJournal1	4,847,571	68,993,773	14.2326
人工グラフ	10,000,000	227,766,776	22.7766

クスの割り当て先はそのバーテックスが属しているクラスタと同じ番号がつけられた計算機とする．エッジの割り当て先は次のようにして決定する．エッジの割り当て判定は，始点となるバーテックスの ID が小さいものから順番に行う．始点バーテックスの ID が同じ場合は，終点バーテックスの ID が小さいものから順番に割り当て判定を行う．両端のバーテックスが同じクラスタに属しているエッジは，そのクラスタと同じ番号がつけられた計算機に割り当てる．両端のバーテックスが異なるクラスタに属しているエッジは，両バーテックスが属する各クラスタのうち，マージ判定を行った時点でサイズが小さい方のクラスタと同じ番号がつけられた計算機に割り当てる．サイズが小さい方のクラスタにエッジを割り当てることで，各計算機に割り当てられるエッジ数を同程度にし等粒度性を高めることができる．また，本手法ではクラスタ間エッジをその両端のどちらかのクラスタに含めることによって，edge-cut から vertex-cut へと変換している．したがって，切断エッジの数を少なくするようにグラフを分割することで切断バーテックスを減らすことが可能となる．

提案手法ではバーテックスの割り当て先はグラフ分割の結果によって決まるため，既存手法と違いグラフ分割を行った計算機以外の計算機は各バーテックスの割り当て先を知ることができない．したがって，あるバーテックスのレプリカが作成される計算機に，そのバーテックスのマスターが割り当てられた計算機の情報を送信する必要がある．あるエッジの両端のバーテックスが異なるクラスタに属していた場合，そのエッジが割り当てられる計算機上にはその計算機とは異なる計算機に割り当てられたバーテックスのコピーが作成される．したがって，add_edge 関数内でエッジとその割り当て先の情報と一緒に，別の計算機に割り当てられたバーテックスの ID とその割り当て先の情報もバッファに記憶する．そして，finalize 関数によりバッファ内のデータの交換が終わったあとバーテックスの ID とそのバーテックスの割り当て先を関連付けたハッシュマップを作成することにより，各バーテックスのマスターの位置を知ることができる．

4. 評価・分析

提案手法の性能を評価するため，PowerGraph における既存のグラフ分割手法である random，oblivious と提案手法とで比較実験を行った．

本実験で用いたデータセットを表 3 に示す．web-Google(注2) は web ページのリンク関係から得られたデータである．wiki-Talk(注3) は Wikipedia の編集ユーザ関係から得られたデータである．soc-LiveJournal1(注4) は LiveJournal のユーザ関係から得られたデータである．また，人工グラフは PowerGraph のグラフを生成する関数を用いて生成したものである．

本実験では，これらのデータセットに対しグラフ分割を行い，その replication factor，グラフのロード時間を示し比較を行う．また，分割手法の等粒度性を評価するために，各計算機に割り当てられたエッジ数の標準偏差を求め比較する．エッジ数の標準偏差が小さいほど等粒度性が高いと評価する．そして，分析処理の高速化について評価するために，表 3 の各データに対し PageRank，単一始点最短路（SSSP）の 2 つのプログラムを実行し，その分析処理時間を測定し比較を行う．ここで，分析処理時間とは各プログラムの分析処理にのみ要した時間のことである．本実験では，実験環境として Amazon EC2 を使用する．使用したインスタンスは r3.2xlarge の linux インスタンスである．使用した CPU は Intel(R) Xeon(R) CPU E5-2670 v2 であり，クロック数は 2.50GHz，コア数は 4 である．そして，メモリは 60GB のものを使用した．また，hdparm を-t オプションをつけ実行することで得られたディスク読み込み速度はおよそ 103MB/sec である．PowerGraph は C++を用いて実装されており，提案手法も C++を用いて実装した．コンパイルに使用した g++のバージョンは 4.8.1 であり，最適化オプションとして-O3 を使用している．また，実験に使用する計算機台数は 4 台,8 台,16 台,32 台と変化させた．

4.1 グラフ分割の性能比較

グラフ分割の性能比較を行うために，既存の PowerGraph と提案手法を組み込んだ PowerGraph でグラフ分割を行い，replication factor，各計算機に割り当てられたエッジ数の標準偏差とグラフのロード時間を比較した．ここで，グラフのロード時間とは，ファイルからのデータの読み込みが始まってから全ての計算機上でローカルグラフの構築が終了するまでに要した時間のことである．本稿では，全てのデータに対し似たような傾向が見られたため，表 3 のデータのうちデータサイズが大きい soc-LiveJournal1 と人工グラフを入力とした場合の実験結果を載せる．図 3，図 4 に実験結果を示す．横軸は計算機台数であり，縦軸はそれぞれ (a) は replication factor，(b) はエッジ数の標準偏差，(c) はグラフのロード時間である．そして，折れ線は各分割手法の結果を表している．

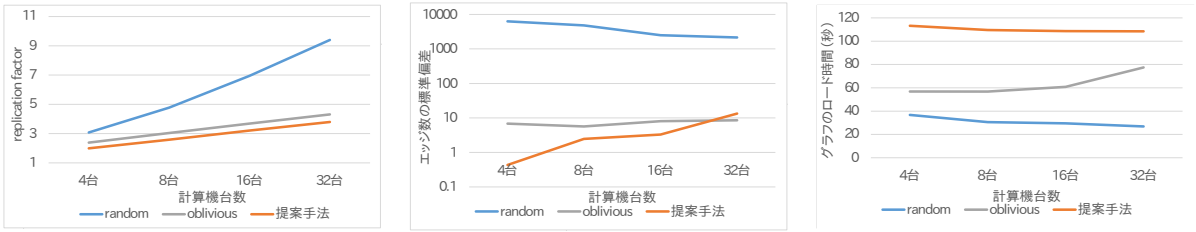
4.1.1 replication factor

図 3(a) より，soc-LiveJournal1 の場合には提案手法は全ての計算機台数において既存手法よりも replication factor が低くなっていることが分かる．web-Google および wiki-Talk の場合にも同様の結果が得られた．それに対し，図 4(a) より，人工グラフの場合，提案手法は計算機台数が 4 台のときに既存手法よりも replication factor が高くなっているのが分かる．提案手法は切断エッジを切断バーテックスに変換しているため，バーテックス数に対するエッジ数の比率が大きくなるほ

(注2)：http://snap.stanford.edu/data/web-Google.html

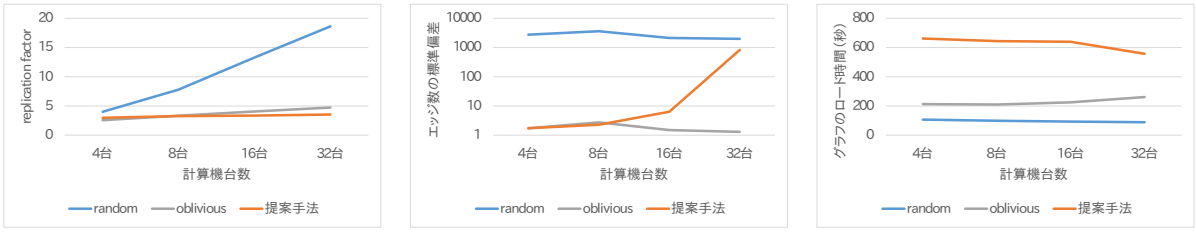
(注3)：http://snap.stanford.edu/data/wiki-Talk.html

(注4)：http://snap.stanford.edu/data/soc-LiveJournal1.html



(a) replication factor　(b) エッジ数の標準偏差　(c) グラフのロード時間

図 3　soc-LiveJournal1 を入力とした場合の性能比較



(a) replication factor　(b) エッジ数の標準偏差　(c) グラフのロード時間

図 4　人工グラフを入力とした場合の性能比較

ど replication factor の軽減効果が小さくなるのだと考えられる．また，提案手法は計算機台数が増えるほど既存手法に対し replication factor は相対的に低くなると予想される．これは，既存手法はバーテックスを各計算機にランダムに割り当てるのに対し，提案手法はグラフ全体の構造を利用して同じクラスタに属するバーテックスは同じ計算機に割り当てることから，計算機台数が増えた場合の切断バーテックス数の増加量が提案手法の方が低いためだと考えられる．

4.1.2　エッジ数の標準偏差

図 3(b)，図 4(b) より，提案手法は計算機台数が少ない場合は既存手法に比べエッジ数の標準偏差が小さくなっているが，計算機台数が多い場合にはエッジ数の標準偏差が大きくなっていることが分かる．web-Google の場合は計算機台数が多い場合にもエッジ数の標準偏差を小さくすることができていたが，wiki-Talk では soc-LiveJournal1，人工グラフと同様の結果が得られた．提案手法においてエッジ数の標準偏差が高くなった場合の各計算機に割り当てられたエッジ数を見てみると，いずれのデータにおいても 1，2 台の計算機に割り当てられたエッジ数が少なくなっており，残りの計算機に割り当てられたエッジ数は同程度であった．その原因として，3.3 節で述べたエッジの割り当て方法に問題があったと考えられる．現在の方法では，エッジの割り当て判定を行う順番は両端のバーテックスの ID によってのみ決まる．そのため，他のクラスタに比べて早く独立した島となってしまうクラスタが現れる場合があると考えられる．一度島となったクラスタは，それ以降の処理においてサイズが大きくならない．そのため，1，2 台の計算機に割り当てられるエッジ数だけが小さくなったのだと考えられる．

4.1.3　グラフのロード時間

図 3(c)，図 4(c) より，提案手法は既存手法に比べグラフのロード時間が長いのが分かる．実験の結果，提案手法のグラフのロード時間は random に対し最大で 7.1 倍，oblivious に対し最大で 3.2 倍の長さとなった．これは，既存手法はデータが読み込まれる時に割り当て先を逐次決定しているのに対し，提案手法は読み込んだデータを全て記憶しておき，クラスタのマージによるグラフ分割を行うためだと考えられる．

4.2　分析処理を行った場合の性能比較

本実験では，既存の PowerGraph と提案手法を組み込んだ PowerGraph において以下の 2 つのプログラムを実行した．

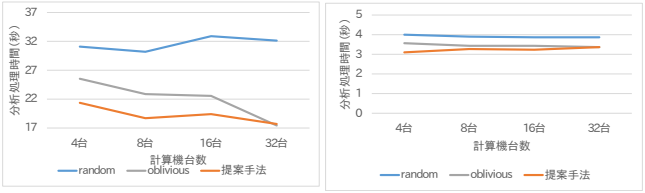
（1） PageRank：ウェブページの重要度を計算するために考えられたアルゴリズムを実装したプログラム．

（2） SSSP：ある一つのバーテックスから他の全てのバーテックスへの最短経路を求めるプログラム．

どちらのプログラムも，各バーテックスを対象に実行される処理の性能はエッジ数に依存する．soc-LiveJournal1 と人工グラフのデータを入力とした場合の各プログラムの分析処理時間を図 5，図 6 に示す．図 5 より，soc-LiveJournal1 では，提案手法は計算機台数が 4 台，8 台，16 台の場合には PageRank，sssp ともに既存手法に比べ分析処理を高速化できていることが分かる．それに対し，計算機台数が 32 台の場合には，既存手法は oblivious より分析処理時間が長くなっているのが分かる．これは，図 3 からも分かるように，計算機台数が少ない場合には提案手法は既存手法に対し replication factor と等粒度性がともに改善されているのに対し，計算機台数が 32 台の場合は等粒度性が低くなっていることから分析処理時間が長くなったのだと考えられる．web-Google，wiki-Talk，人工グラフも同様に，replication factor と等粒度性がともに改善されている場合は分析処理が高速となっていることが確認できた．実験の結果，提案手法により分析処理が random に対し最大 2.6 倍，oblivious に対し最大 1.2 倍高速化されている．

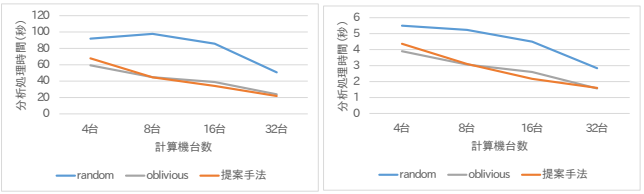
5.　関 連 研 究

代表的な分散処理フレームワークとして MapReduce [9] が挙



(a) PageRank　　(b) sssp

図 5　soc-LiveJournal1 を入力とした場合の分析処理時間



(a) pagerank　　(b) sssp

図 6　人工グラフを入力とした場合の性能比較

げられる．MapReduce は汎用的な大規模データ処理を目的として設計されたフレームワークであり，グラフ構造のデータを直接表現するような機能は持っていない．MapReduce の java による実装である Hadoop を用いてグラフ構造のデータを対象にした処理を行った場合，GraphLab に比べ 40〜60 倍の処理時間を要するという実験結果が報告されている [2]．

PowerGraph 以外の分散グラフ処理フレームワークとして Pregel [1] や GraphLab [2] が挙げられる．Pregel は BSP モデルをもとに設計，実装されたメッセージパッシング型のフレームワークであり，super-step と呼ばれる処理を繰り返し実行することで目的とする分析結果を得る．super-step は前回の super-step で出力されたメッセージを入力とし，各バーテックス上でユーザ定義の処理を実行し，次の super-step の入力となるメッセージを出力する．GraphLab は非同期分散共有メモリ型のフレームワークであり，各バーテックス上で実行されるユーザ定義の処理はメモリ上のグラフデータを共有する．各処理は，その対象とするバーテックスのデータだけでなく隣接エッジと隣接バーテックスのデータにも直接アクセスできる．PowerGraph は，この二つのフレームワークを参考に設計，実装されている．

グラフを分割する方法として METIS [10] を用いる方法が挙げられる．METIS はグラフを分割するための様々なアルゴリズムが実装されたソフトウェアパッケージである．METIS は 3 つのステップを経てグラフを k 個の部分グラフに分割する．まず，最初のステップでバーテックスを集約しつつ段階的にグラフを小さくしていく．二つ目のステップで，小さくなったグラフを切断されるエッジ数が最小になるように，かつそれぞれに内包されるバーテックスの重さが同程度になるように 2 つに分割する．そして，三つ目のステップで一つ目のステップを遡るようにバーテックスを元に戻しつつグラフ分割を更新していく．METIS には，各ステップにいくつかのアルゴリズムが用意されており，それらを選択することでグラフ分割の性能を調整できる．

6. ま と め

本稿では，大規模なグラフ構造のデータを高速に分割し，かつ高い等粒度性と低い replication factor を実現するグラフ分割手法を提案した．また，それを既存の分散グラフ処理フレームワークである PowerGraph に組み込み性能評価を行った．PowerGraph における既存のグラフ分割手法と提案手法とで比較実験を行った結果，入力となるグラフのバーテックス数に対するエッジ数の比率が小さいほど，また計算機台数が多くなるほど既存手法に比べ replication factor が低くなることが分かった．また，台数が少ない場合においては既存手法に比べ等粒度性が高くなることが分かった．そして，replication factor と等粒度性を改善することで分析処理を高速化できることを確認した．本稿では，提案手法により分析処理が既存のグラフ分割手法である random に対し最大 2.6 倍，oblivious に対し最大 1.2 倍高速化可能であることを示した．

今後の課題として，計算機台数が増えた場合の等粒度性の低下を改善する必要がある．そのために，バーテックスの次数や他のクラスタとの間に存在する切断エッジ数などの偏りが大きかった場合でも等粒度性を高くできるようなエッジ割り当て方法について考える必要がある．

文　献

[1] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser and G. Czajkowski, “Pregel: a system for large-scale graph processing,” Proceedings of SIGMOD, 2010.

[2] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola and J. M. Hellerstein, ”Distributed GraphLab: a framework for machine learning and data mining in the cloud,” PVLDB, 2012.

[3] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson and C. Guestrin, “PowerGraph: distributed graph-parallel computation on natural graphs,” Proceedings of OSDI, 2012.

[4] H. Shiokawa, Y. Fujiwara, and M. Onizuka, “Fast algorithm for modularity-based graph clustering,” Proceedings of the 27th AAAI Conference on Artificial Intelligence, 2013.

[5] 飯田恭弘, 岸本康成, 藤原靖宏, 塩川浩昭, 鬼塚真 ”大規模グラフ構造データからのコミュニティ抽出と重要度計算　〜高速化への取り組みと応用〜, 人工知能学会誌, 2014.

[6] M. E. J. Newman and M. Girvan. “Finding and evaluating community structure in networks,” Phys. Rev. E, 2004.

[7] J. Leskovec, K. J. Lang, A. Dasgupta and M. W. Mahoney, ”Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters,” Internet Mathematics 6, 2008.

[8] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. “Fast unfolding of communities in large networks,” Journal of Statistical Mechanics: Theory and Experiment, 2008.

[9] J. Dean and S. Ghemawat, “Mapreduce: simplified data processing on large clusters,” Proceedings of OSDI, 2004.

[10] G. Karypis and V. Kumar, “METIS: unstructured graph partitioning and sparse matrix ordering system” Technical Report, Department of Computer Science, University of Minnesota, 1995.