ロードバランス重視のグラフ分割手法

袁　沛霖† 鬼塚　真† 佐々木勇和†

† 大阪大学大学院情報科学研究科　〒 565-0871 大阪府吹田市山田丘 1 − 5
E-mail: †{yuan.peilin,onizuka,sasaki}@ist.osaka-u.ac.jp

あらまし　近年，数億ノードから構成される大規模なグラフが登場し，より高効率のグラフ計算をできる分散グラフ処理は重要になりつつある．分散グラフ処理ではグラフをいかに分割するかは現在注目されている課題の 1 つである．既存研究では Neighbor-Expansion と呼ばれるエッジのクラスタ成長策略を用いた分割手法が提案されている．しかし既存手法ではクラスタ内のノード数を均等に分割することが難しいためロードバランスが悪くなり，その結果，各マシンの利用率が低くなるという問題が存在する．そこで本稿では既存手法を改良し，よりロードバランスの取れる手法を提案する．提案手法では，常に最も小さいクラスタを拡張することによってロードバランスを維持する．実験では，本稿の提案手法が常に既存手法より優れたロードバランスを取れることを確認した．そして，処理時間がノードのバランスに依存するアルゴリズムを実行する際の処理時間の短縮も観察された．

キーワード　分散グラフ処理，グラフ分割，ロードバランス

1 はじめに

現在，大規模なグラフが世の中に多く存在している．例えば，インターネット上のサービスから得られる Twitter や Facebook のようなソーシャルグラフ，ウエブページの間のリンク関係を表すウェブグラフ，各種の道路の空間的な分布状態を表現する道路網やセンサネットワークによって構成される IoT(Internet of Things) からなるネットワークグラフなど様々なグラフ構造データが実世界に存在する．そして，これらのグラフの大規模化が進んでいるだけではなく，そのペースは早くなっている．特にソーシャルグラフにおいて，そのノード数が既に数億を超えていることは決して少なくない，Facebook の全世界アクティブユーザ数は 10 億以上である [1]．豊かかつ有用な情報がグラフデータの中に含まれているが，それは顕在的な情報と潜在的な情報の二種類がある．顕在的な情報とは，グラフの形や構造から見るだけで理解できる情報のことを指す．例えば，ソーシャルグラフの中のスーパーノードを見つけることで有名人特定するという問題がある．一方，潜在的な情報はグラフデータを一定な分析パターンによって分析しなければ発見できないものである．もしこれらのグラフデータをデータマイニングアルゴリズムによって適切に扱えば，貴重なデータを手に入れることができる．一例として，Google が提案したウェブページの重要度を評価する PageRank アルゴリズムによって検索エンジンでのウェブページランキング問題を解決し，最もユーザーに役立ちそうなページを優先的に検索結果に表示することができた [2], [3], [22]．しかし，グラフノード数の増加に伴って，このようなグラフにおける必要となる計算時間も相当増加している．従って，グラフ計算の更なる高速化技術への需要が高まっている [33].

現在数多くの分散グラフ処理高速化の研究が行われている [18], [19], [20], [21], [23]．シングルマシンでグラフを処理する手法も提案されているが [4], [31], [32], 高速に処理できるグラフデータ規模の上限があるため，より注目されているのは分散処理技術を利用した手法である．これは定められたルールに従ってグラフ計算を複数のマシンに分割して実行し，マシン間の通信でやり取りしながら結果を求める技術である．本稿では，各マシンに割り当てられるグラフをオリジナルグラフの部分グラフと呼ぶ．分散処理技術が発展することで高速な大規模グラフ計算が可能になり，いくつかの分散グラフ処理エンジンが既に提案されている．例えば，Graphlab [5], Pregel [6], PEGASUS [7], [8], PowerGraph [11], PowerLyra [34], Spark GraphX [24], Chaos [25] などがある．

分散グラフ処理効率に影響を与える要因にはマシン間の通信コストと各部分グラフのエッジ数とノード数のロードバランスがある [9], [10]．ロードバランスが悪い場合は，処理が速いマシンは最も遅いマシンを待つ必要があるため，マシンの利用率が低い．通信コストが大きい場合は，通信待ちで計算が止まってしまう．更に，マシン間通信は一般的に遅いため，通信回数が多ければ多いほど分散処理の効率が悪くなる．これまでこの 2 つの要因を改善するためのグラフ分割手法に関する議論が様々ある [15], [16].

本研究では従来手法を踏まえて新しいグラフ分割手法を提案する．提案手法はロードバランスを重視し，Neighbor Expansion と呼ばれるクラスタ拡張手法に基づいている [17]．この手法は常にノードの core 集合と boundary 集合を維持し，1 つの boundary 集合の中にいるノードとノード間のエッジが 1 つのクラスタを構成する．boundary 集合の中の特定の条件に満たすノードを core 集合に移動すると同時にこのノードに繋がる一部のエッジをクラスタの中に追加する．既存の Neighbor Expansion を利用したグラフ分割手法では，個々のクラスタを順次成長させるため実験によりクラスタ間でのノードのバランスが悪い欠点があることが分かった．提案手法はノード数とエッジ数のロードバランスを重視するよう，Neighbor Expansion

を拡張するため，複数のクラスタを徐々に成長させるという戦略を取る．提案手法を多様なグラフデータに適用することで，提案手法の方がより均等にエッジとノードを各クラスタに割り当てることを確認した．また，処理時間がノードのバランスに依存するグラフ分析アルゴリズムを実行する際の処理時間の短縮も観察された．

本稿の構成は，次の通りである．2 章で既存研究について述べる．3 章において提案手法の事前準備に関して詳説する．4 章で既存手法の問題点を分析し提案手法の詳細について説明する．5 章において提案手法の評価と分析を行う．6 章で本稿をまとめる．

2 既存研究

既存のグラフ分割手法はノードを一意に割り当てる edge-cut とエッジを一意に割り当てる vertex-cut という大きく二種類の手法があるが，Power-law に従うグラフを対象とする場合は，vertex-cut を適用する方針が計算効率が良いことが確認されている．生成したノードのコピー数の期待値から考えると，edge-cut によるノードのコピー数を g とすると，vertex-cut によるノードのコピー数は g より小さくかつ厳密なる上限がある [11]．続いて vertex-cut に関する既存研究について説明する．

Chaos [25] は RAND と呼ばれる手法を用いて高いネットワーク帯域幅を活用しランダムにエッジを各部分グラフに割り当てる．このような処理は非常に速いスピードを保証することができるが，その代わりにグラフの構造を無視して高いノードの replication factor をもたらす．replication factor が高ければ高いほどノードがより多く切断されていることを意味する．切断回数が多くなると，グラフ処理で計算する必要があるノードも多くなり， 処理時間が長引いてしまう．

DBH [26] はエッジの密度を主なパラメータとするハッシュ関数に基づくグラフ分割手法である．この手法は最初に全てのノードをランダムに割り当て，そしてエッジ集合 E をトラバースし，任意のエッジ (i, j) に関してその時点で $degree(i)$ と $degree(j)$ のどっちが小さいかを判断し，次数が小さい方が存在する部分グラフに割り当てられる．DBH は power-law グラフのノード次数分布を利用しているが，RAND と同様にグラフの構造を活用していないため，ノードの replication factor はまだ高い．

Oblivious [11] は 1 つのエッジをどの部分グラフに送るかを決める際に既に各部分グラフに割り当てられたエッジを考慮するストリーミングアルゴリズムである．実行する際，1 つの部分グラフに割り当てようとしているエッジは常により多くのノードを共有するエッジを持つ部分グラフに割り当てる．

HDRF [27] は Oblivious を拡張したストリーミングアルゴリズムであり，DBH のように更に power-law グラフのノード次数分布を活用している．HDRF も Oblivious もエッジを割り当てる際に既に割り当てられたエッジにしか依存していないため，グラフの構造の一部のみを活用している．

Sheep [28] は分割統治法を用いたグラフ分割手法である．HDRF や Obilivious と比べてみるとその利点はグラフの構造の全体を考えできることである．しかし欠点として，Sheep を用いる際に良いパフォーマンスを発揮できるのは木構造に近いグラフのみである．

前述したのは代表的な vertex-cut 手法であるが，いずれもグラフの構造を活用できないので，結果として高い replication factor となる欠点がある．また， edge-cut 手法の中で一番注目されていた METIS [29] アルゴリズムも vertex-cut をできるように拡張された [30]．この手法はグラフ全体の構造を重視し，メモリの中でグラフの分割を行う．確かにこれによってグラフを分割して得られる replication factor は他の vertex-cut 手法より優るが，大規模グラフには通用できないという欠点がある．

既存の vertex-cut 手法の中では最も良い分割結果を出せるのは Neighbor Expansion [17] と呼ばれるアルゴリズムを利用した順次にエッジのクラスタを拡張する手法である． 3 章がそれについて詳しく分析する．

3 事前準備

前述した既存手法を説明するためまず Neighbor Expansion アルゴリズムを説明する [17]．Neighbor Expansion は一定なルールに従ってクラスタの中に吸収するノードを選び，持つエッジ数が上限数に達するまでノードの選択を繰り返し実行するアルゴリズムである．Neighbor Expansion アルゴリズムでは．ノード吸収のルールを実現するためノードの core 集合 C と boundary 集合 S が事前に定義され，以下の関係に従う．

$$C = \emptyset, S = \emptyset$$

$$C \subseteq S \text{ が常に成立する}$$

初期化する際，オリジナルグラフデータからランダムにノードを選び，S と C の中に一緒に追加され，そしてこの node の全ての隣接ノードも S の中に追加される．もし S の中ではエッジによって繋がっている 2 つのノードが存在するなら，このエッジをエッジのクラスタに割り当てる．それと同時に，オリジナルグラフからこのエッジを削除する．

続いて Neighbor Expansion ではエッジのクラスタを拡張するためのループが始まる．本稿は 図 1 を用いてこのループを説明する．

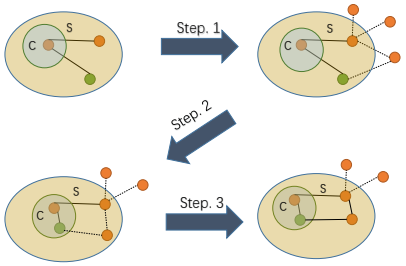

図 1 Neighbor Expansion の例（緑色のノードは今回選ばれた C に追加するノード，実線はクラスタに吸収されたエッジ， 点線はまだオリジナルグラフに存在するエッジ）

ループは大きく 3 つのステップに分けられている．

Step.1 において，$S-C$ に属する任意のノード n を探索し，n の隣接ノードであり，S に属さないノードの件数を集計する．そして，この数が一番小さいノードを選出する．もし $S-C$ が $\emptyset$ であれば，オリジナルグラフからノードをランダムに選出して C と S に追加する． 図 1 では，最初 $S-C$ に属する 2 つのノードに対して，上のオレンジ色のノードが持つ S に属さないノードの数は 3 であり，下の緑色のノードが持つ S に属さないノードの数は 1 である．緑色のノードの方が小さいため，このノードは選ばれる．

Step.2 では，C が先に *Step*.1 で選ばれたノードを追加する．そして，C に追加されたノードに対して，その S に属さないノードは全て $S-C$ に追加される． 図 1 では，選ばれた緑色のノードが C に追加される．このノードの隣接ノードは 1 つしかないかつ $S-C$ に属されていないので，隣接ノードは S に追加される．

Step.3 はループ最後のステップである．このステップでは *Step*.2 において新しく $S-C$ の中に入った全てのノードを探索する．もしあるノードの隣接ノードが既に S に属しているなら，このノードと S に属する隣接ノードをつなぐエッジは拡張されているエッジのクラスタに追加される．同時に，オリジナルグラフからエッジを削除する．これによってオリジナルでの削除されたエッジと関わるノードの次数が小さくなる．ここで，i を拡張されているエッジのクラスタの番号として，クラスタを C_i と定義する． 図 1 では，前述した条件を満たすエッジは 2 本であり，C_i に吸収されたので点線から実線に変化した．

オリジナルグラフを p 個に分割する場合，このグラフ分割手法は C_1 から C_p まで順次にエッジのクラスタを拡張する．拡張する際，クラスタの中のエッジの数が $\alpha * \frac{|E|}{p}$ に達するまで前述した Neighbor Expansion アルゴリズムのループが実行される．ここで，α はクラスタのエッジの数の上限をコントロールするためのユーザーが定義できる変量である．クラスタの拡張が全部終了すると，得られた各クラスタはオリジナルグラフから切り離した部分グラフである．

4 提 案 手 法

本章では既存手法の問題点を分析し，解決手法を提案する．

4.1 既存手法の問題点

3 章で述べた通り，Neighbor-Expansion のループにおいて core ノード集合 C の中に追加されるのは boundary ノード集合 S に属するノードであり，且つ S に属さない隣接ノードの数がもっとも少ないノードである．この手法より C に追加するノードを選んだ後，新しく S に追加するノードは少ないため，Neighbor Expansion によって新しくクラスタに追加するエッジも少ない．エッジ数が増えないように制御できる．しかし，Power-law グラフの特徴として全てのノードの次数の分布が冪級数分布になり，スーパーノードと呼ばれる次数が他のノードを遥かに超えているノードが存在する．もしスーパーノードの隣接ノードが選択されて C の中に追加されたら，自然に隣接しているスーパーノードが S の中に追加される．スーパーノードの次数がアルゴリズムの実行と伴って減っていくが，順次拡張する際，特に一番目や二番目に拡張されるクラスタに対して，スーパーノードの次数はまだ大きい．この場合ではスーパーノードが S の中に入っても次数が小さくなるまで C の中に追加されることは難しい．一方，1 つのノードが S に追加される手段として，ランダムに選択される以外は隣接ノードが C に追加されるのを待つしかない．つまり，スーパーノードと隣接している次数が小さいノードにとって，スーパーノードが C に追加されるまで S に追加されない可能性は高い．従って，スーパーノードと繋がるノードの次数は小さければ小さいほど順次拡張の最後のクラスタに残される可能性が高い．特にスーパーノードと繋がる次数が 1 のノードは確実に最後に残される（ランダムに選択された次数が 1 のノードを除く）．

ここでは問題が発生する．次数が小さいノードが S に追加される際，既に S の中にされた隣接ノードの数も少ないので，新しくクラスタに吸収されるエッジの数も少ない．例えば，一番最後に残されやすい次数が 1 のノードにとって，S に追加されたら新しくクラスタに吸収されるエッジの数も 1 である．逆に，順次拡張において優先に拡張されるクラスタでは S に追加されるノードに対して，S に追加されると同時にクラスタに吸収されるエッジの数は比較的に多い．従って，Neighbor-Expansion では拡張されるクラスタがエッジ数の上限を持っているので，拡張順番が後ろのクラスタ内のエッジ数は設定されたエッジ数上限に達するまで吸収するノード数は比較的に多いだと思われる，クラスタ間のノードのアンバランスが避けられない．

この仮説を確認するため，実際に live-journal [14] と com-orkut [13] という 2 つのソーシャルグラフを利用して確認したところ，ノードのアンバランスな状況が 2 つのデータで 図 2 と図 3 の状況になったことを確認した．赤い線はトレンドラインであり，クラスタの中のノードの数の変化の傾向を表す．既存手法によって分割されたグラフではノードのバランスが悪い，より先に拡張されたクラスタではより少ないノードを持つ傾向があることが分かった．このような現象をもたらす主な原因としてはスーパーノードに繋がる低次数のノード，特に次数が 1 のノードは最後のクラスタにより集中的に吸収されることである．

数多くの分散グラフ計算は分割後エッジとノード両方のロードバランスと関わっている．ノードのバランスが悪い場合は，前述したような分散グラフ計算では処理が速いマシンは最も遅いマシンを待つ必要があるため，マシンの利用率が低い．従って，いかにノードのロードバランスを取るかが本研究の課題になる．

4.2 ロードバランス重視のグラフ分割

既存手法の問題を改善するため本研究ではクラスタの拡張順番を変える．既存手法のような 1 つのクラスタがエッジ数の上限まで拡張されたら次のクラスタの拡張操作を始めるのではなく，本研究では常に一番少ないエッジを持つクラスタから拡張

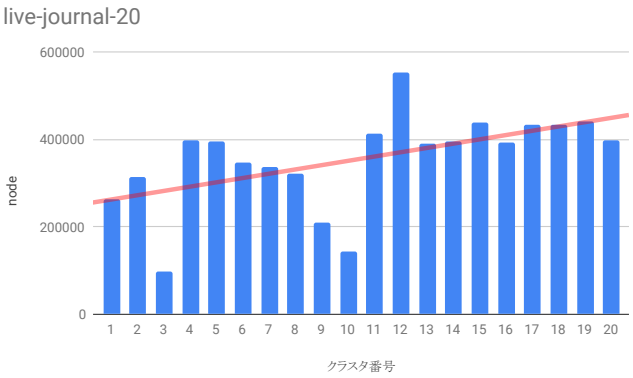

図 2　live-journal を 20 個に分割した時のノード数分布

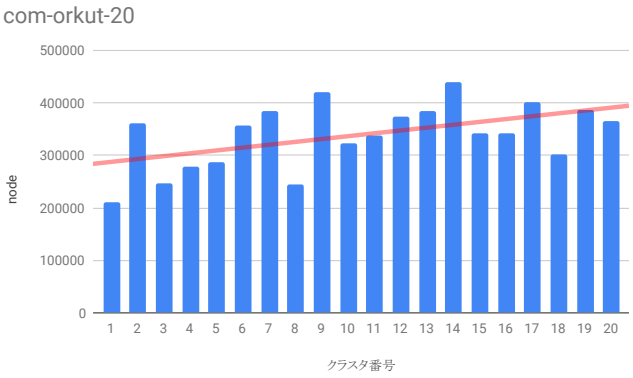

図 3　com-orkut を 20 個に分割した時のノード数分布

するロードバランス重視の順位付き拡張手法を提案する．理由としては 2 つある．1 つ目はこうするとエッジのバランスが取りやすいこと，2 つ目は拡張するクラスタの切り替えを頻繁に行うことによってスーパーノードに繋がる低次数ノードを各クラスタに分散できる．

提案手法では分割数と同じ数のクラスタを初期化して任意のクラスタから Neighbor Expansion の単一ループ処理を実行し，新しいエッジがクラスタに入る度に次に拡張するクラスタを再選択する．この手法の擬似コードを Algorithm 1 に示す．

ロードバランスを重視する順位付き拡張手法

- グラフを p 個に分割する際，p 個のクラスタを一挙に初期化する．そして Neighbor Expansion のノードやエッジを吸収するループを開始し，持つエッジの数が一番少ないクラスタから拡張というルールに従って次に拡張するクラスタを選ぶ．もしこのようなクラスタが 2 つ以上存在するなら，ランダムに拡張するクラスタを選択する．
- Neighbor Expansion の単一ループ処理によってクラスタを拡張する．初期 node がスーパーノードである場合を考慮し，既存手法と同じく各クラスタのエッジ数の上限を $\alpha * \frac{|E|}{p}$ に設定する． 推奨設定としては，α は 1 である．この設定によって分割結果でのクラスタ間のエッジのバランスは既存手法と同じぐらい取ることができる．
- 新たなノードが boundary ノード集合 S に追加されて

Algorithm 1 ロードバランスを重視する順位付き拡張手法

1: $\{C_1, ..., C_p\} = \{\emptyset, ..., \emptyset\}$
2: **while** $E \neq \emptyset$ **do**
3: $\quad z \in \arg\min_{1 \leqq m \leqq p} edgesize(C_m)$
4: $\quad E = Neighbor\_Expansion(C_z)$
5: **end while**

クラスタにエッジの吸収が終わったら，今回の Neighbor Expansion を終了し，提案手法最初のクラスタ選択のステップに戻す．

上記のような手法により，グラフ分割する際に各クラスタ内のエッジ数のバランスを取ることができるだけではなく，最後に残された元々スーパーノードと繋がった低次数ノードも各クラスタに分散することができる．従って，エッジとノード両方のバランスを取れる．

5　実験・評価

5.1　実 験 目 的

1，ロードバランス重視のグラフ分割は良いノードバランスを取れるかどうかを確認する．

2，提案手法は既存手法のように良いエッジバランスを維持するかどうかを確認する．

3，提案手法によって処理時間がノードバランスに依存する分散グラフ処理が速くなれるかどうかを確認する．

5.2　実験環境とグラフデータ

本研究では Amazon Web Service を利用して実験を行う．使用した環境についてを 表 1 に示す．

表 1　AWS の実験環境

Amazon マシンイメージ (AMI)	ami-b9206089
インスタンスタイプ	m4.xlarge
ルートデバイスタイプ	ebs
cpu	Intel Xeon E5-2676v3
cpu 数	4
マシン台数	5 ～ 30
動作周波数 (GHz)	2.40
メモリ (GB)	16
OS	Ubuntu 14.04
ネットワークパフォーマンス	高い

また，実験で使用されたグラフデータを 表 2 に示す．

表 2　実験用のグラフデータ

名称	グラフ種類	ノード数	エッジ数
california [14]	道路網	1,965,206	5,533,214
cit-patent [12]	引用関係グラフ	3,774,768	16,518,948
com-Orkut [13]	ソーシャルグラフ	3,072,441	117,185,083

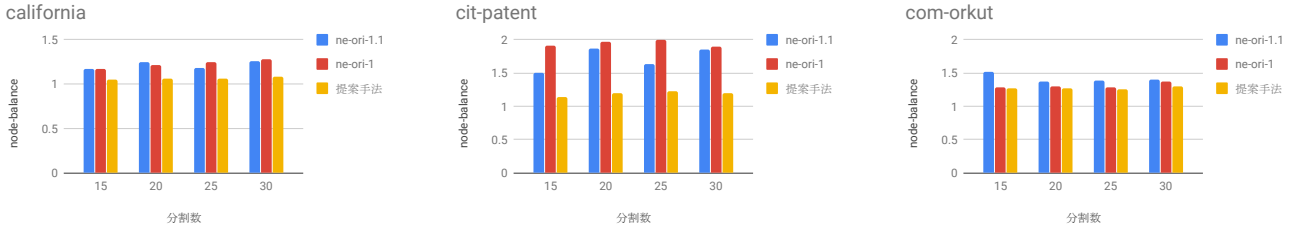

図 4　node balance の実験結果

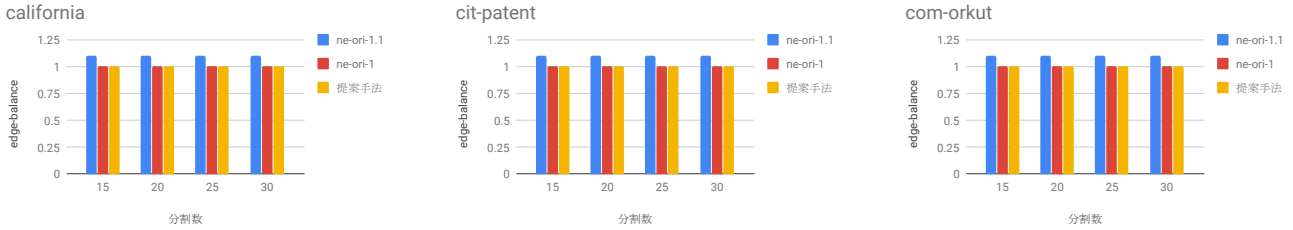

図 5　edge balance の実験結果

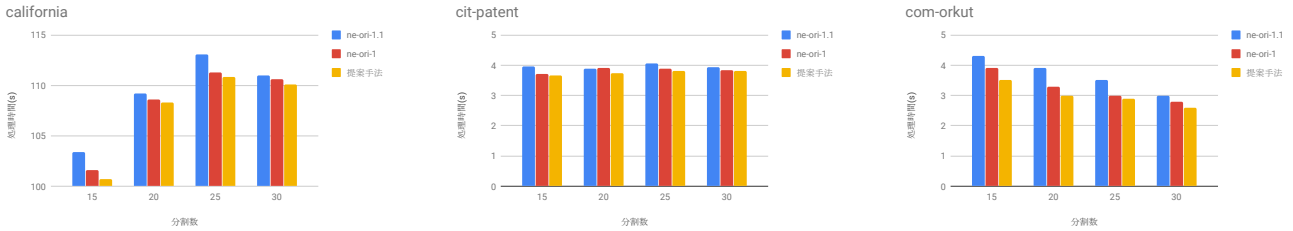

図 6　SSSP の実験結果

5.3　実験結果と分析

本研究の提案手法はロードバランス重視なので，分割後部分グラフ間の既存手法より優れたノードバランス及びエッジバランスを取ることが目標である．ノードバランスとエッジバランスを良いか悪いかを評判するため，本研究では node balance と edge balance を使う．定義として部分グラフのノードあるいはエッジの最大値と期待値の商を意味する．具体的に言うと，まず分割後の各部分グラフのノードの総数及びエッジの総数を求め，分割数で割って各部分グラフのノードとエッジの期待値を計算する．node balance は各部分グラフの中の一番多いノード数を期待値で割って得られた値である．同じように，edge balance は各部分グラフの中の一番多いエッジ数を期待値で割って得られた値である．

この 2 つの値をロードバランスを評価する指標として使う理由は分散処理では毎回の同期処理の時間は一番遅いマシンによって決定されるためである．1 つの部分グラフが 1 台のマシンに割り当てられ，部分グラフのノード数やエッジ数は多ければ多いほど処理時間が長引いてしまう．従って，上記 2 つの値は 1 に近ければ近いほど良いバランスを取ったことを意味する．また，本研究の実験も時間計算量 $O(V^2)$ の Single Source Shortest Path(SSSP) の実行に必要な時間を記録した．図 4，図 5，図 6 に実験結果を示す．

実験によると，edge balance の値に関して既存手法 ($\alpha = 1$) と提案手法両方も限りなく期待値の 1 に近い．だが，node balance の値に関しては，提案手法の方が 1 により近い．つまり，提案手法がより均等にノードを各部分グラフに割り当てられることは確認された．

SSSP の実行に要する処理時間に関しては，提案手法の方が既存手法より短いことも観察された．従って，ノードバランスに依存するグラフ計算を実行する際に提案手法の有効性が確認された．

6　ま　と　め

本稿では，分散グラフ処理高速化のための Neighbor Expansion と呼ばれるエッジのクラスタの拡張法を元に既存グラフ分割手法の改良案を提案した．

既存手法はクラスタを順次に拡張し，その結果としてクラスタの番号の増加に伴ってクラスタ内のノード数も増える傾向があり，ノードのバランスが悪い．そこで本稿は常に一番少ないエッジを持つクラスタから拡張を始める手法を提案する．実験によって提案手法は既存手法より均等にノードを各クラスタに割り当てれることが証明された．また，提案手法によりノードの次数分布が Power-law を満たすグラフを分割した後，SSSP のような処理時間がノードバランスに依存するグラフ計算を実行する際の時間が既存手法より短いことも観察された．

今後の課題として，Neighbor Expansion を実行する際の初期ノードの選択が分割結果に与える影響を考察するのを考えられる．

文　　献

[1] Christopher Sibona and Jae Hoon Choi. Factors Affecting EndUser Satisfaction on Facebook. In Proceedings of the 6th International AAAI Conference on Weblogs and Social Media (ICWSM 2012). AAAI Press, 2012.

[2] Kamvar S, Haveliwala T, Golub G. Adaptive methods for the computation of PageRank[J]. Linear Algebra & Its Applications, 2004, 386(2):51-65.

[3] Jeh G, Widom J. Scaling personalized web search[C]. International Conference on World Wide Web. ACM, 2003:271-279.

[4] Kyrola A, Blelloch G, Guestrin C. GraphChi: large-scale graph computation on just a PC[C]. Usenix Conference on Operating Systems Design and Implementation. 2014:31-46.

[5] Low Y, Gonzalez J, Kyrola A, et al. GraphLab: A Distributed Framework for Machine Learning in the Cloud[J]. Eprint Arxiv, 2011.

[6] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing[C]. ACM SIGMOD International Conference on Management of Data. ACM, 2010:135-146.

[7] Kang U, Tsourakakis C E, Faloutsos C. PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations[C]. Ninth IEEE International Conference on Data Mining. IEEE Computer Society, 2009:229-238.

[8] Han M, Daudjee K, Ammar K, et al. An experimental comparison of pregel-like graph processing systems[J]. Proceedings of the Vldb Endowment, 2014, 7(12):1047-1058.

[9] Shiokawa H, Fujiwara Y, Onizuka M. Fast algorithm for modularity-based graph clustering[C]. Twenty-Seventh AAAI Conference on Artificial Intelligence. AAAI Press, 2013:1170-1176.

[10] Rahimian F, Payberah A H, Girdzijauskas S, et al. JA-BE-JA: A Distributed Algorithm for Balanced Graph Partitioning[C]. IEEE, International Conference on Self-Adaptive and Self-Organizing Systems. IEEE Computer Society, 2013:51-60.

[11] Gonzalez J E, Low Y, Gu H, et al. PowerGraph: distributed graph-parallel computation on natural graphs[C]. Usenix Conference on Operating Systems Design and Implementation. USENIX Association, 2012:17-30.

[12] J. Leskovec, J. Kleinberg and C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), 2005.

[13] J. Yang and J. Leskovec. Defining and Evaluating Network Communities based on Ground-truth. ICDM, 2012.

[14] J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29–123, 2009.

[15] Sun R, Zhang L, Chen Z, et al. A Balanced Vertex Cut Partition Method in Distributed Graph Computing[C]. Revised Selected Papers. Springer-Verlag New York, Inc. 2015:43-54.

[16] Lang K. Finding good nearly balanced cuts in Power-law graphs[J]. Preprint, 2004.

[17] Chenzi Zhang, Fan Wei, Qin Liu, Zhihao Gavin Tang, and Zhenguo Li. 2017. Graph Edge Partitioning via Neighborhood Heuristic. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17). ACM, New York, NY, USA, 605-614.

[18] Boldi P, Rosa M, Santini M, et al. Layered label propagation:a multiresolution coordinate-free ordering for compressing social networks[C] J Acoust Soc Am, 2011:587-596.

[19] Power R, Li J. Piccolo: building fast, distributed programs with partitioned tables[C]. Usenix Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, Bc, Canada, Proceedings. DBLP, 2010:293-306.

[20] Gregor D, Lumsdaine A. The Parallel BGL: A generic library for distributed graph computations[J]. In Parallel Object-Oriented Scientific Computing. POOSC, 2005.

[21] Ekanayake J, Li H, Zhang B, et al. Twister: a runtime for iterative MapReduce[C]. ACM International Symposium on High PERFORMANCE Distributed Computing. ACM, 2010:810-818.

[22] Guo, Tao, et al. Distributed Algorithms on Exact Personalized PageRank. Proceedings of the 2017 ACM International Conference on Management of Data. ACM, 2017.

[23] LUO, Xinyuan, et al. Tuning the granularity of parallelism for distributed graph processing. Distributed and Parallel Databases, 2017: 1-32.

[24] GONZALEZ, Joseph E., et al. GraphX: Graph Processing in a Distributed Dataflow Framework. In: OSDI. 2014. p. 599-613.

[25] ROY, Amitabha, et al. Chaos: Scale-out graph processing from secondary storage. In: Proceedings of the 25th Symposium on Operating Systems Principles. ACM, 2015. p. 410-424.

[26] XIE, Cong, et al. Distributed power-law graph computing: Theoretical and empirical analysis. In: Advances in Neural Information Processing Systems. 2014. p. 1673-1681.

[27] PETRONI, Fabio, et al. Hdrf: Stream-based partitioning for power-law graphs. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. ACM, 2015. p. 243-252.

[28] Daniel Margo and Margo Seltzer. 2015. A scalable distributed graph partitioner. Proc. VLDB Endow. 8, 12 (August 2015), 1478-1489.

[29] KARYPIS, George; KUMAR, Vipin. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on scientific Computing, 1998, 20.1: 359-392.

[30] BOURSE, Florian; LELARGE, Marc; VOJNOVIC, Milan. Balanced graph edge partition. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014. p. 1456-1465.

[31] Wang K, Hussain A, Zuo Z, et al. Graspan: A Single-machine Disk-based Graph System for Interprocedural Static Analyses of Large-scale Systems Code[J]. Acm Sigplan Notices, 2017, 51(2):389-404.

[32] Maass S, Min C, Kashyap S, et al. Mosaic: Processing a Trillion-Edge Graph on a Single Machine[C]. Twelfth European Conference on Computer Systems. ACM, 2017:527-543.

[33] Xie C, Chen R, Guan H, et al. SYNC or ASYNC: time to fuse for distributed graph-parallel computation[C] ACM Sigplan Symposium on Principles and Practice of Parallel Programming. ACM, 2015:194-204.

[34] Chen R, Shi J, Chen Y, et al. PowerLyra: differentiated graph computation and partitioning on skewed graphs[C]// Tenth European Conference on Computer Systems. ACM, 2015:1.