冪乗則を考慮したデータ分割手法によるグラフ処理基盤の高速化

菊島　健治†　　山室　　健†　　本庄　利守†　　岩村　相哲†

† 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
E-mail: †{kikushima.kenji,yamamuro.takeshi,honjo.toshimori,iwamura.sotetsu}@lab.ntt.co.jp

あらまし　ソーシャルメディアにおける人の関係性を表すソーシャルグラフの分析など，大規模なグラフデータの分析に対する需要が高まってきている．このような大規模なグラフデータを処理するために，さまざまな分散処理基盤が提案されているが，いずれにおいてもグラフ全体をいかに分割し，いかに効率的に分散配置するかが性能の鍵となっている．グラフの分割手法として，単純に辺の始点・終点の情報を用いて分割する 2 次元パーティショナと呼ばれる手法があるが，そのような単純な方式ではソーシャルグラフなどが冪乗則に従う特性が考慮されておらず効率的に処理が行えない．そこで本研究では，この 2 次元パーティショナを基に冪乗則を考慮して頂点の複製を低減させる手法を提案する．本手法を Apache Spark 上のグラフ処理系である GraphX に実装し，オリジナルの 2 次元パーティショナに比べて、性能が改善することを示す.

キーワード　分散並列処理, グラフパーティショナ, Apache Spark

1. はじめに

人やモノのつながりにおける関連性を分析するには，人やモノを頂点とし，そのつながりを辺とすることで表現できるグラフ構造が適している．近年，SNS などのソーシャルメディアが普及し，その利用者から得られるつながりの情報をソーシャルグラフと呼び，グラフデータ分析の適用対象例の 1 つに挙げられる．また，M2M や IoT といったトレンドに沿ったデバイスが登場しつつあり，近い将来多様なデバイス群が接続される時代が到来するとみられている．これらの接続されたデバイス群をグラフと見なすと，デバイス間の関連性を分析することも可能になる．しかし，2020 年には出荷デバイス数が約 260〜300 億台に到達すると推測されており [1] [2] ，今後グラフデータの大規模化は避けられない状況にある．

このような大規模グラフデータ処理のためには，分散並列処理が有用と考えられ，これまで複数の方式やプロダクトが提案されてきた．著名なものでは，Graphlab [3] や Pregel [4] ，Apache Hama [5] ，Apache Giraph [6] ，Apache Spark(GraphX) [7] 等が挙げられるが，いずれの場合も処理対象となるグラフを複数に分割し，分散配置した上で並列実行している．

しかし，グラフデータは，その構造上均等かつ頂点や辺を重複なく分割することが難しく，適切に分割できないと分散並列処理の性能低下を引き起こす．例えば，分割後のデータに頂点や辺の数の偏りがあると，複数の処理間でスキューが発生して並列度が下がる要因となる．また，システム間をまたがる頂点や辺が多くなると，分散実行した個々の出力結果を集約する処理に時間が掛かるため，この場合も処理結果を得るまでの時間に影響を及ぼす．グラフ分散処理基盤を効率的に動作させるためには，グラフをいかに偏りなくかつ頂点や辺を重複させずに分割できるかが重要である．

グラフ分割のための手法は，これまでいくつか提案されており，METIS [8] のように分割のための前処理を行うものと，Graphlab の Grid パーティショナのように前処理を行わないものに分類できる．前者は，前処理によりグラフの特性に応じた分割が可能になるが，処理に非常に時間がかかる欠点があり，後者は分割速度は速いものの，グラフの特性を考慮しないため分割結果が前者に劣るという欠点がある．

近年のデータ大規模化に伴い，後者のような分割速度が高速なものが多くオープンソースソフトウェア (OSS) として実装されているが，前述のとおりグラフの特性を考慮していないため，ソーシャルグラフのような冪乗則に従う特性を持つデータのグラフ処理を行うには非効率である．具体的には，グラフ分割によって生じる頂点の複製が増加すること，また分割した個々のパーティションのサイズに偏りが発生し，並列処理におけるスキューの要因となることである．そこで，本研究では既存の 2 次元パーティショナを基に，冪乗則を考慮したパーティショナを提案する．冪乗則を考慮したグラフ分割を行うことで，頂点の複製とパーティションサイズの偏りを低減することが可能である．

提案手法を Apache Spark のグラフ処理系である GraphX 上に実装して評価を行った結果，既存の実装に対して頂点の複製を最大 33%削減，パーティションサイズの偏りを最大 99%削減し，PageRank アルゴリズムを最大 1.8 倍高速化出来ることを確認した．

以下，2 章にて冪乗則に従う特性を持つグラフについて，分散処理基盤である Apache Spark 及び GraphX について，さらに GraphX に実装されている 2 次元パーティショナの問題点について述べる．3 章にて，冪乗則を考慮した 2 次元パーティショナを提案する．4 章では，実験による頂点の複製数及びパーティションの偏り評価とそれらによって得られるグラフ処理高速化の効果を示す．5 章で関連研究について述べ，6 章で本論文をまとめる．

2. 前提知識

2.1 Apache Spark

Apache Spark(以下，Spark) [9] とは，UCB AMPLab で開発され，現在は Apache トッププロジェクトの一つとなっているインメモリ分散並列処理基盤である．Spark は RDD(Resilient Distributed Dataset) [10] と呼ばれるデータ操作の抽象化層を提供しており，コレクションと同等の操作をユーザに提供しながら，その操作が分散並列処理として実行される基盤となっている．利用者は RDD が提供する API を通じて，MapReduce [11] など所望のアルゴリズムを実装することで，分散並列処理を容易に実現できる．

また，Spark は基本的にインメモリで動作させることを想定しており，ユーザが生成した RDD をメモリ上に保持するか否かをユーザ自身で指定することができる．この機能を活用することで，利用頻度の高いデータのみをメモリ上に保持し，ディスク I/O を削減させることができる．この仕組みは同一のデータに対して収束するまで演算を繰り返す機械学習分野に有効であると期待されており，グラフデータ処理もそのメリットを享受できると考えられている．

なお，Spark には SQL ライクなクエリ言語処理系 [12] も実装されており，SQL ライクなクエリで生データを加工することにより，加工したデータをそのまま Spark 上の機械学習ライブラリ (MLlib) [13] やグラフ処理系 (GraphX) への入力として利用することができる．

2.2 GraphX

GraphX は，Spark 上に実装されたグラフ処理系であり，上記で述べた Spark が提供する機能が利用されている．例えば，GraphX 上のグラフデータは Spark の RDD を用いて，頂点の情報を VertexRDD，辺の情報を EdgeRDD として保持されている．

また，GraphX は Spark API を用いて BSP アルゴリズム [14] を実装しており，利用者向けには BSP を抽象化し，GraphX API として提供している．PageRank [15] などの主要なアルゴリズムはこの BSP API を基に既に実装されているため，利用者はあらためてグラフアルゴリズムを個別に実装することなく利用可能である．

GraphX はグラフデータの分割方式として，辺を分割せずに頂点を複製することで分割する，いわゆる Vertex Cut 方式を採用している．これは，Powergraph [16] が，冪乗則に従う特性を持つグラフ処理において，辺を分割する Edge Cut 方式よりも Vertex Cut 方式の方が高速に実行できることを示したことから，GraphX も Powergraph にならって同様の分割方式を採用したためである．本研究では，この Vertex Cut 時に行われる，辺の集合である EdgeRDD を複数のパーティションに分割する処理に注目する．この分割処理の良否により，グラフ処理時間に影響を与えるためである．

2.3 2 次元パーティショナ

2 次元マトリックスを利用したパーティショナは，いくつかの手法が提案されているが，ここでは，GraphX に実装されている Partition2D パーティショナについて述べる．Partition2D パーティショナは，辺を構成する始点・終点の情報を基に，各辺の該当パーティションを決定する手法である．

ここで，$\{V_0, V_1 ... V_n\}$ からなる頂点の集合を V，$\{E_0, E_1 ... E_m\}$ からなる辺の集合を E とした任意のグラフ $G = (V, E)$ を考え，G を P 個に分割するものとする．まず，頂点 V を縦横両軸に同一順で並べたマトリックス M を考える．この M における一方の軸を各辺が持つ始点 (図中 Source)，もう一方の軸を終点 (図中 Destination) に相当するものとする．この両軸に対して，始点軸・終点軸ともに $V/\sqrt{P}$ 個ごとに境界を定めると，総計 P 個に分割された格子を生成できる．そして，$e \in E$ を満たす辺 e の始点・終点をそれぞれの軸に当てはめると，e が属するパーティションを決定できる．

このパーティショナの利点は，その原理上， 各頂点の複製数を最大でも $2\sqrt{P} - 1$ 以下に抑制できる点にある．ある頂点 v に着目した場合，v を始点とする辺は必ず特定の列に存在し，その格子数は $\sqrt{P}$ となる．同様に，v を終点とする辺は必ず特定の行に存在し，その格子数は $\sqrt{P}$ となる．この列と行は交差することから，1 つ分のパーティションは重複し，頂点 v が存在するパーティションは $2\sqrt{P} - 1$ 以下となる．

図 1 に $P = 9$ とした場合の分割例を示す．辺 $e(v_a, v_b)$，$e(v_c, v_d)$，$e(v_e, v_f)$ が始点・終点の違いにより，パーティション 0, 4, 7 にそれぞれ割り当てられている．

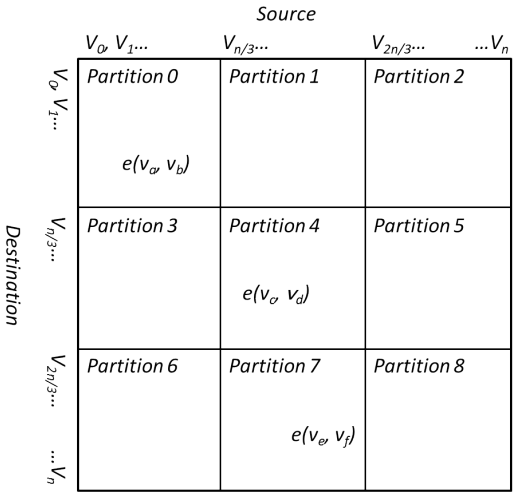

図 1 Partition2D パーティショナによるグラフ分割例 (P=9)

2.4 グラフ分割における評価指標

Vertex Cut 方式のグラフ分割では，分割した結果を評価するために，2 つの指標を用いることが知られている．1 つは頂点の複製数を示すレプリケーションファクタであり，もう 1 つは分割後の各パーティションが有する辺の数を比較し，その偏りを示すロードバランスファクタである．

ここで，頂点の集合を V，辺の集合を E とする任意のグラフ $G = (V, E)$ を考える．分割数を p，分割時に配置されるノード群を A とすると，$e \in E$ の辺 e は $A(e) \in \{1, ..., p\}$ を満たすよう分割される．同様に，$v \in V$ の頂点 v は $A(v) \subseteq \{1, ..., p\}$ を満たすよう分割される．この時，もっともバランスよく分割さ

れた状態を式 (1) 及び式 (2) として定義する [16].

$$min_A \frac{1}{|V|} \sum_{v \in V} |A(v)| \tag{1}$$

$$max_m |\{e \in E | A(e) = m\}| < \lambda \frac{|E|}{p} \tag{2}$$

レプリケーションファクタは式 (1) で示され，頂点の複製数が少ない方が望ましいことを示している．ロードバランスファクタは式 (2) で示され，各パーティションが保持する辺数に偏りが少ない方が望ましいことを示している．また，今後ロードバランスファクタを評価する場合には，式 (2) 中の λ 値を用いる．

レプリケーションファクタとロードバランスファクタは基本的にはトレードオフの関係にある．例えば GraphX に実装されている既存のパーティショナを例にとると，辺をランダムに配置する Random パーティショナでは，各パーティションに均等に辺が配置されるため，ロードバランスファクタが 1.0 に近い良好な値となるが，頂点の配置に規則性がなく，レプリケーションファクタは他パーティショナに比べて大きくなる．一方，2.3 で説明する 2 次元パーティショナは頂点の複製数に上限を設けているため，レプリケーションファクタは Random パーティショナより良くなるが，辺の配置が均等にはならないため，ロードバランスファクタは劣化する．

2.5 冪乗則に従う特性を持つグラフ

これまで，ソーシャルグラフなどの実世界におけるグラフが研究された結果，冪乗則に従う特性を持つことが示されてきた [17] ．グラフにおける冪乗則とは，頂点における次数を d とし，次数が d となる頂点の割合を $P(d)$ とすると，

$$P(d) \propto d^{-\alpha} \tag{3}$$

の関係が成り立つ特性である．なお，実世界におけるグラフでは式中の α 値が概ね 2.0 前後になることも知られている．

このような次数に偏りがあるグラフを Vertex Cut 方式で分割する場合，次数の大きな頂点を多数のパーティションに割り当てるように分割すると頂点の複製数が増加してしまい，レプリケーションファクタが悪化する．効率的なグラフ分散並列処理を行うには，複製する頂点を適切に選択し，レプリケーションファクタの悪化を抑える必要がある．

2.6 Partition2D パーティショナの問題点

Partition2D パーティショナは，平易なアルゴリズムで，Random パーティショナ等の他の単純な分割手法より良好な性能が得られることが多いが，以下の 2 つの問題を抱えている．

1 つ目は，各辺の持つ頂点の組み合わせのみで該当パーティションを決定しており，グラフの持つ特性を考慮していない問題である．

ソーシャルグラフのような実世界のグラフデータは，多くの場合に次数に偏りを持つ，いわゆる冪乗則に従う特性を持っている．しかし，Partition2D はグラフの分割数を P とすると，個々の頂点に対して複製数が $2\sqrt{P}-1$ 以下になることを保証するのみであり，冪乗則に従う特性を活かしてレプリケーションファクタを向上させたり，次数の情報を活かしてロードバランスファクタを向上させるような仕組みにはなっていない．

問題点の 2 つ目は，GraphX 上の実装では $\sqrt{P}$ が自然数ではない場合，個々のパーティションに割り当てられる頂点数が均等にならないため，ロードバランスファクタに悪影響を及ぼす問題である．

前述のような頂点 V，辺 E で構成されるグラフ $G = (V, E)$ を，P 個に分割する場合を考える．ここで，$N = \lceil \sqrt{P} \rceil$ とすると，頂点の総数 V_n から，マトリックス M の縦横それぞれの軸は，V_n/N 個に分割されることになる．しかし，この時 $N^2 - P \neq 0$ の場合には $N^2 - P$ 個の余計な格子が生成され，このままでは，P 個に分割することができない．現状の実装では，格子 p $\{p | P < p \leqq N\}$ は，$p\%N$ 番目のパーティションに格子ごと割り当てられる．そのため，仮に各格子に同一数の辺が存在したとすると $(N^2 - p)/p$ 分の格子が他の格子とは倍の辺を持つことになり，ロードバランスファクタに悪影響を及ぼす．

図 2 に $P = 20$ とした場合の例を示す．この場合，$N = \lceil \sqrt{20} \rceil = 5$ となり，両軸が 5 つに分割され計 25 個の格子が作られる．グラフの辺はそれぞれこの格子のいずれかに一旦割り当てられるが，分割数 p を超過する 21〜25 個目の格子に該当する辺は，格子ごと 1〜5 番目パーティションのいずれかに割り当てられる．辺 $e(v_a, v_b)$ と $e(v_c, v_d)$ は図に記載されているとおり，それぞれパーティション 0, 11 に配置されるが，$e(v_e, v_f)$ は 22 % 20 = 2 となり，パーティション 2 に割り当てられる．

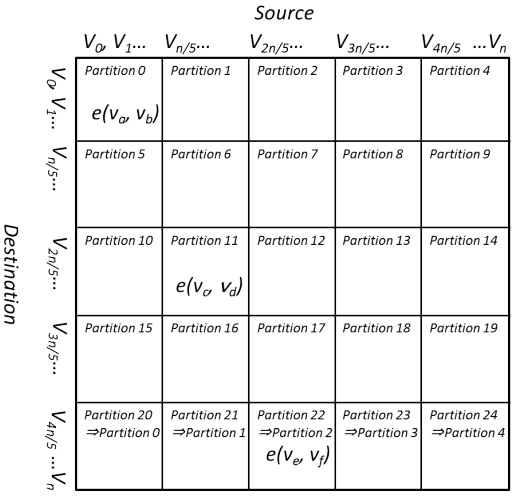

図 2　Partition2D パーティショナによるグラフ分割例 (P=20)

3. 提案手法

2.6 で述べた Partition2D パーティショナの問題点解決のために，以下の 2 つの手法を提案する．本手法は，ロードバランスファクタを 1.0 に近い値に保ちながら，レプリケーションファクタを低減させていく方針となっている．

3.1 分割数の約数を用いた境界設定によるパーティションあたりの辺の均一化

分割数 P における $\sqrt{P}$ が自然数ではない場合に発生する，

ロードバランスファクタへの悪影響を解消するために，前述のマトリックス M の始点側と終点側の軸に設ける境界を $\sqrt{P}$ のような値は使わず，P の約数を用いて境界を設定する．

境界の設定には，$P = m \cdot n$ を満たす m, n (ただし $m, n \in \mathbf{N}$) の組み合わせにおいて，$|m - n|$ が最小となるペアを用いる．これにより，$\sqrt{P}$ の切り上げによる P より大きな格子の生成を抑止することができる．また，ここで $|m - n|$ が最小のペアを用いる理由は，レプリケーションファクタを悪化させないためである．

図 3 に $P = 20$ とした場合の例を示す．20 の約数である 1, 2, 4, 5, 10, 20 の組み合わせから，始点側・終点側で $[1, 20], [20, 1], [2, 10], [10, 2], [4, 5], [5, 4]$ の 6 通りが考えられ，この内，差分の絶対値の小さな $[4, 5]$, $[5, 4]$ のいずれかを採用する．図では $[4, 5]$ の場合を示す．結果，生成される格子数は 20 になり，図 2 で例示したような問題は発生しない．

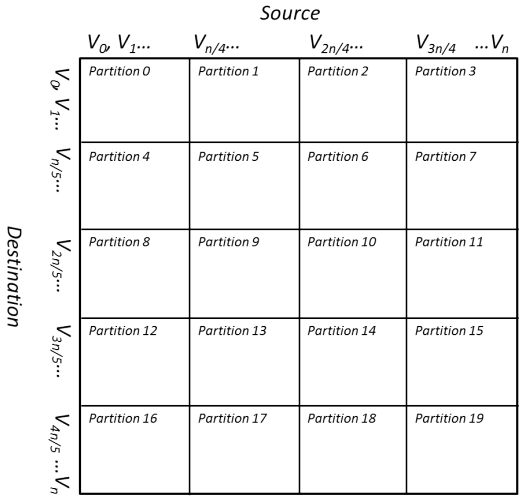

図 3　分割数の約数を用いた境界設定の例 (P=20)

3.2　冪乗則に従うグラフ特性を用いた格子設定による頂点複製の低減

次に，冪乗則を考慮したグラフ分割を行う．これまで分割時に作成するマトリックスにおける縦横両軸の頂点の並び順は，特に意味を持ったものではなく，単に頂点ごとに割り振られた実装上の番号が用いられていた．

ここでは，グラフの冪乗則に従う特性を考慮して頂点の並び順を変えることで，レプリケーションファクタの向上を図る．この手法は，次数の大きな頂点は互いに同じ頂点から参照されている可能性が高いため，次数順に終点を並べて同一パーティションに集約することで，頂点の複製数を低減できるという考え方に基づいている．また，この際に高次数の終点を集約するパーティションには多くの辺が存在することから，ロードバランスファクタの悪化を招かないように，格子の区切り方を狭くして辺の数に偏りが発生しないようにする．

ここで 3.1 で述べたマトリックス M を再び用いる．終点側に用いる約数を m，始点側に用いる約数を n とする．まず，終点側の頂点の並び順を次数順にソートする．次に，次数の高い方から順に k 個の頂点を選出する．その際，k 個の頂点を終点とする辺の数の合計が $E/P \cdot 2n$ となるようにする．k 個の頂点に対して，始点をもとに $2n$ 個にパーティショニングする．なお，これらの頂点に対する辺の数が多いことを鑑みて，パーティション数が $2n$ 個になるように始点側の境界を n ではなく，$2n$ で定める．その他の辺は $(m - 2) \cdot n$ の格子で区切るようにして，既存の Partition2D と同様のアルゴリズムで分割する．このようにすると，終点側の次数の大きな頂点は同一の始点を持つ可能性が高いことから，同一パーティションにすることで，頂点の複製数を低減することができる．また，あらかじめ 1 パーティションあたりの辺数が平滑化されるように辺を抽出しているため，ロードバランスファクタには悪影響を及ぼさない．

図 4 に $P = 20$ とした場合の例を示す．始点側・終点側を $[4, 5]$ で分割後，終点側の最も次数が大きな 2 段分を用いて，終点側には境界を設けずに 8 つの格子を生成している．

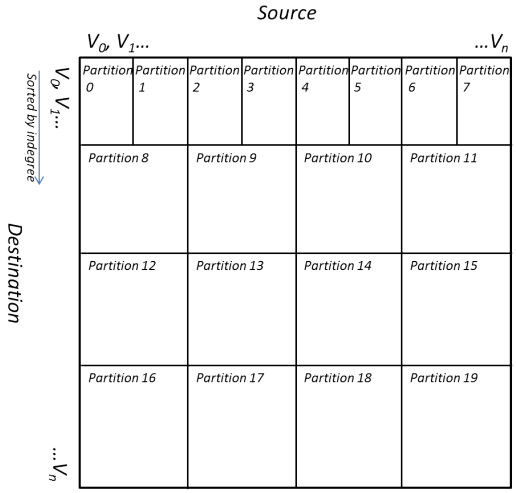

図 4　冪乗則を考慮した格子設定の例 (P=20)

さらに，上記の格子設定を最も次数の多い頂点群だけでなく，その次の頂点群にも同様に適用することで，レプリケーションファクタをより改善することが出来る．図 5 に $P = 20$ とした時，$4n$ 個分のパーティションに対して提案手法を適用した場合の例を示す．

4.　評　　価

GraphX に実装されている既存パーティショナ (Random パーティショナ及び Partition2D パーティショナ) と提案手法について，グラフ分割指標とグラフ処理性能の比較のために実験を行った．

実験に用いたグラフデータセットを表 1 に示す．このデータセットは，実世界のデータを基にしており，冪乗則に従う特性を持つことが知られている．

性能評価は，Amazon EC2 における r3.4xlarge(CPU:16cores, Memory:122GiB Storage:320GB SSD) インスタンスを 9 台 (master:1 台，slaves:8 台) 用いて実施した．全ノードで Amazon Linux 2014.09.1，Oracle JDK 1.7.0_67，Spark 1.1.1 を用

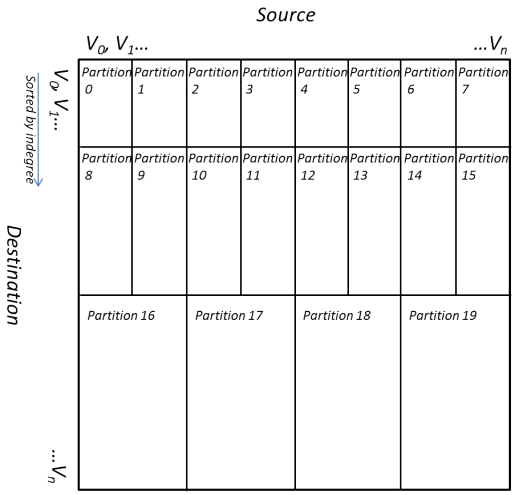

図 5　冪乗則を考慮した格子設定の例 (2) (P=20)

表 1　実験に用いたグラフデータセット

名称	$\|V\|$	$\|E\|$
web-Google [18]	$0.9M$	$5M$
LiveJournal [18]	$4M$	$68M$
uk-2002 [19]	$18M$	$298M$
webbase-2001 [19]	$116M$	$1020M$

いている．

4.1　レプリケーションファクタに与える影響の比較

図 6 にグラフ分割数を 8/16/32/48/64 とした場合の各分割手法におけるレプリケーションファクタを示す．

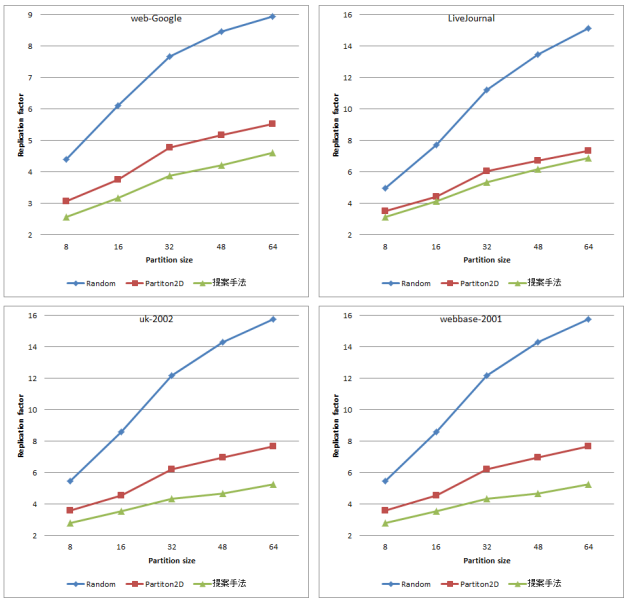

図 6　レプリケーションファクタ

提案手法を適用することで，全ての分割数でレプリケーションファクタが向上している．Partition2D と比べると，web-Google では平均 17.1%，LiveJournal では平均 8.6%，uk-2002 では平均 27.5%，webbase-2001 では平均 23.9% の向上が見られた．特に，uk-2002 の場合では最大 33%レプリケーションファクタが改善された．

4.2　ロードバランスファクタに与える影響の比較

図 7 にグラフ分割数を 8/16/32/48/64 とした場合の各分割手法におけるロードバランスファクタを示す．

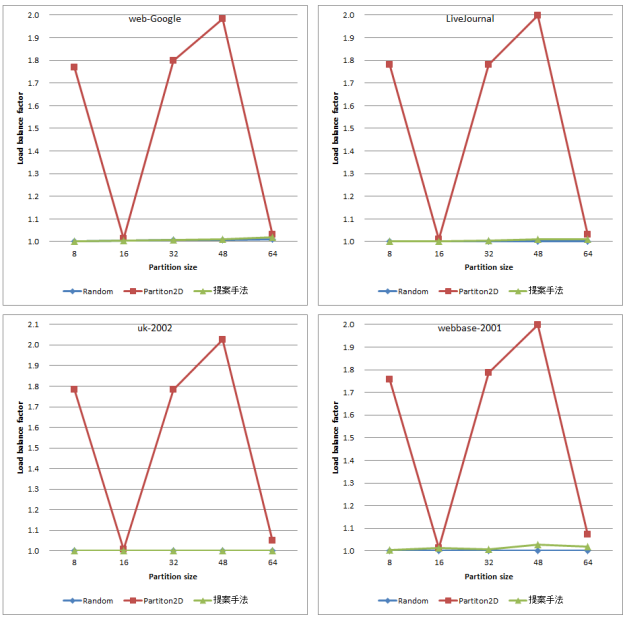

図 7　ロードバランスファクタ

Partition2D パーティショナは，2.6 で述べたように，分割数によってはロードバランスファクタが大きく劣化していることが分かる．Random パーティショナに対しては，全ての場合でロードバランスファクタが劣っている．提案手法は始点の分布によって，パーティションの偏りがどうしても出てしまうが，ほぼ 1.0 に近い値となっており，全ての分割数で Partition2D よりロードバランスファクタが向上している．Partition2D と比べると，web-Google では平均 80.0%，LiveJournal では平均 89.5%，uk-2002 では平均 96.8%，webbase-2001 では平均 74.6% の向上が見られた．

4.3　グラフ処理実行時間の比較 (PageRank)

PageRank は，グラフの各頂点において隣接頂点が持つ PageRank 値の総和をその頂点の PageRank 値とする PageRank アルゴリズム [15] に基づき，グラフ上の全ての頂点に対し PageRank 値を求める処理である．

図 8 に PageRank 処理を実行時間の計測結果を示す．なお，グラフ分割数は Spark クラスタのスレーブノード CPU コア総数と同じ 128 とした．

図 9，図 10 に分割数 128 の場合のレプリケーションファクタとロードバランスファクタを示す．どちらも前節のレプリケーションファクタ評価・ロードバランスファクタ評価で述べた内容と同じ傾向にある．

PageRank 実行時間について，Random パーティショナと比べて平均 1.79 倍，最大 2.38 倍，Partition2D と比べて平均 1.40 倍，最大 1.80 倍の高速化を確認した．これは，頂点の複製低減による PageRank 値集約の効率化と，各パーティションの平滑化により PageRank 値計算のスキューが解消されたためと思われる．

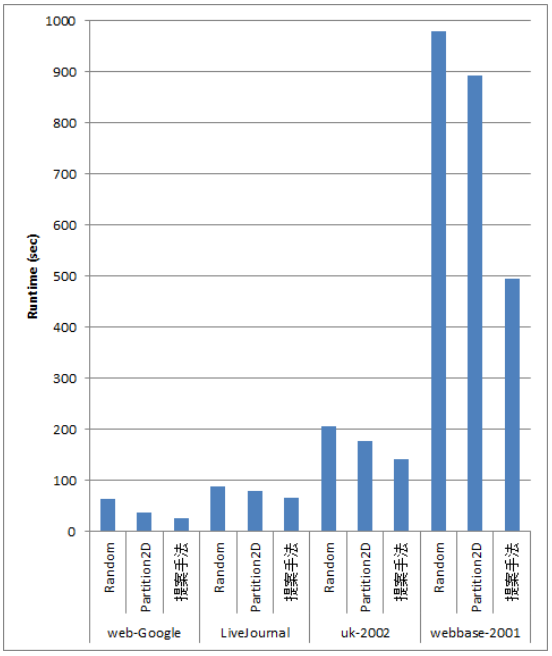

図 8　PageRank 処理実行時間 (20 iterations)

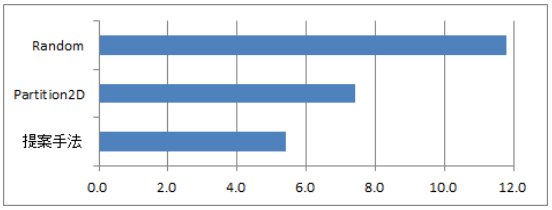

図 9　レプリケーションファクタ (webbase-2001; 分割数 128)

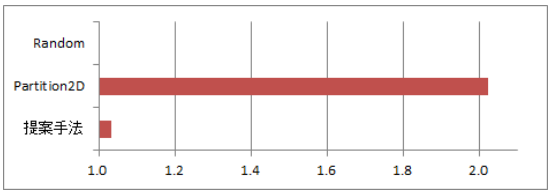

図 10　ロードバランスファクタ (webbase-2001; 分割数 128)

4.4　グラフ処理実行時間の比較（ConnectedComponent）

ConnectedComponent は，任意のグラフにおいて頂点が連結された極大な部分グラフを求める処理である．

図 11 に ConnectedComponent 処理実行時間の計測結果を示す．前項と同様にグラフ分割数は 128 とした．なお，データセットを webbase-2001 とした場合は，本評価環境では 24 時間以上経っても処理が完了しなかったため，計測対象から除外している．

ConnectedComponent 実行時間について，Random パーティショナと比べて平均 1.37 倍，最大 1.46 倍，Partition2D と比べて平均 1.16 倍，最大 1.25 倍の高速化を確認した．

4.5　クラスタリソースへの影響

図 12 に 4.3 で計測した PageRank 処理実行時 (webbase-2001; 分割数 128; 20 iterations) の Spark クラスタのスレーブノードにおけるネットワーク通信量を示す．

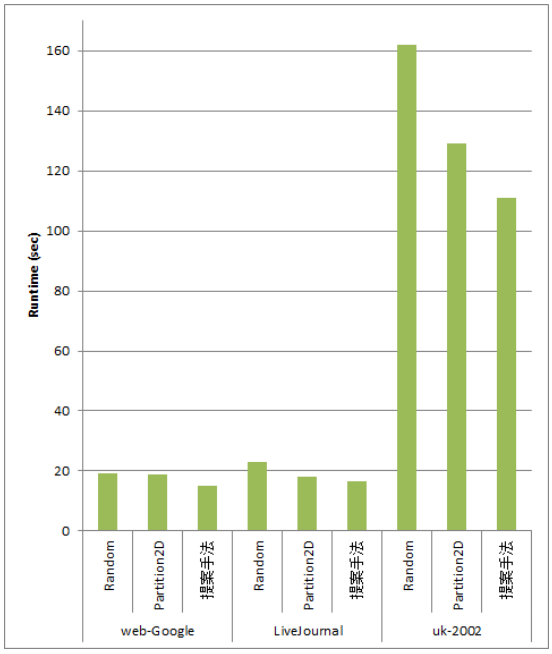

図 11　ConnectedComponent 処理実行時間

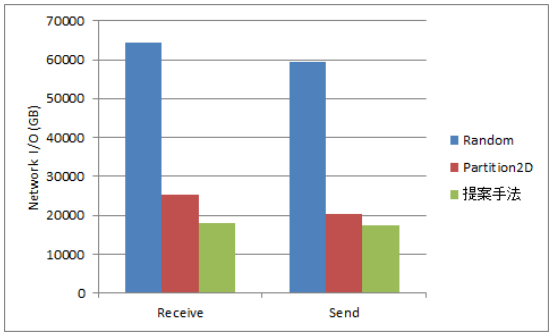

図 12　分割処理方式ごとのネットワーク通信量

Random パーティショナと比べて 71.6%，Partition2D と比べて 22.7%の通信量が削減されている．この計測条件ではレプリケーションファクタが Partition2D 比で 27.0%削減されており，頂点の複製数削減が通信量削減に貢献していることが分かる．

次に，図 13，図 14 に 4.3 で計測した PageRank 処理実行時 (webbase-2001; 分割数 128) の Spark クラスタの各スレーブノードにおける CPU 使用率を示す．この図は，どちらも PageRank 処理の 1 イテレーション分を示している．

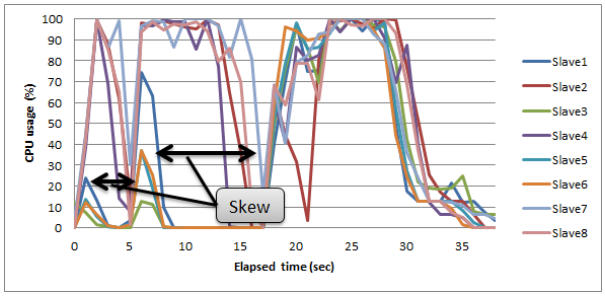

図 13　PageRank 処理時のスレーブノード CPU 使用率 (Partition2D)

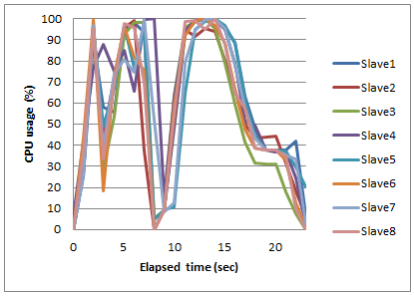

図 14　PageRank 処理時のスレーブノード CPU 使用率 (提案手法)

GraphX における PageRank 実装は，PageRank 値計算以外に頂点データ RDD(VertexRDD) から辺データ RDD(EdgeRDD) へ更新情報を伝達する処理も行うため，1 イテレーション中に計 3 回の同期が発生する．

Partition2D では，ロードバランスファクタ悪化の影響により，1 回目と 2 回目の同期タイミングでタスクのスキューが発生している．この時，他のスレーブノードは待機状態になっており，CPU リソースの有効利用ができていない．提案手法では，Partition2D 時に見られたスキュー発生事象は解消しており，全スレーブノードの CPU リソースを有効利用できている．

以上のように，データセットや処理内容によらず，本手法の適用によりクラスタリソースの利用率が向上し，その結果グラフ処理が高速化されることが確認された．

5. 関 連 研 究

Powergraph は，GraphX と同様の Vertex Cut 方式を採用しており，複数のパーティショナが実装されている．その中でも Grid と呼ばれるパーティショナが，2 次元パーティショナの一種である．このパーティショナは，Partition2D パーティショナと同様に，頂点を並べたマトリックスを利用するが，辺が持つ頂点の組み合わせにより一意にパーティションを決定するのではなく，その辺が該当する格子の上下左右全てを対象として，ランダムに割り当てるパーティションを決定する．Partition2D のように頂点の複製数を抑制しつつ，Random パーティショナのように，パーティションあたりの辺の数の均等化を狙う，Partition2D と Random の中間的な存在と言える．

GraphBuilder [20] では，上記の Grid パーティショナの考え方を用いながら，ランダムに割り当てる対象のパーティションの選出にマトリックスではなく，トーラス状の格子を用いる手法が提案されている．これにより，割り当てる可能性のあるパーティション数が少なくなり，頂点の複製を抑制することができる．Grid ではグラフ分割数を P とする時，頂点の最大複製数が $2\sqrt{P}-1$ であったものが，提案手法では $1.5\sqrt{P}+1$ になると主張している．

なお，いずれの手法も辺の持つ頂点情報のみで分割しており，本研究のようなグラフ特性を考慮した分割は行っていない．

6. ま と め

本研究では，冪乗則に従うグラフに対して，その特性を活かしたグラフ分割を行うことによりロードバランスファクタを良好な状態に保ちつつ，レプリケーションファクタが改善できることを示した．またその結果，グラフ処理時間の短縮に貢献できることを示した．

文　　献

[1] International Data Corporation. Finding success in the new iot ecosystem: Market to reach $3.04 trillion and 30 billion connected ”things” in 2020, idc says. *IDC Press Release*, 2014.

[2] Gartner, Inc. Gartner says the internet of things installed base will grow to 26 billion units by 2020. *Gartner Press Release*, 2013.

[3] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5(8):716–727, April 2012.

[4] Grzegorz Malewicz, Matthew H. Austern, Aart J.C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale graph processing - ”abstract”. In *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*, PODC ’09, pages 6–6, New York, NY, USA, 2009. ACM.

[5] Sangwon Seo, Edward J. Yoon, Jaehong Kim, Seongwook Jin, Jin-Soo Kim, and Seungryoul Maeng. Hama: An efficient matrix computation with the mapreduce framework. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM ’10, pages 721–726, Washington, DC, USA, 2010. IEEE Computer Society.

[6] Apache Giraph. http://giraph.apache.org/.

[7] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. Graphx: Graph processing in a distributed dataflow framework. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI’14, pages 599–613, Berkeley, CA, USA, 2014. USENIX Association.

[8] Amine Abou-Rjeili and George Karypis. Multilevel algorithms for partitioning power-law graphs. In *Proceedings of the 20th International Conference on Parallel and Distributed Processing*, IPDPS’06, pages 124–124, Washington, DC, USA, 2006. IEEE Computer Society.

[9] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud’10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[10] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI’12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[11] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[12] Reynold S. Xin, Josh Rosen, Matei Zaharia, Michael J. Franklin, Scott Shenker, and Ion Stoica. Shark: Sql and rich analytics at scale. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’13, pages 13–24, New York, NY, USA, 2013.

ACM.

[13] MLlib. https://spark.apache.org/mllib/.

[14] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, August 1990.

[15] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[16] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*, OSDI'12, pages 17–30, Berkeley, CA, USA, 2012. USENIX Association.

[17] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *SIGCOMM Comput. Commun. Rev.*, 29(4):251–262, August 1999.

[18] Jure Leskovec and Rok Sosič. SNAP: A general purpose network analysis and graph mining library in C++, June 2014.

[19] Laboratory for Web Algorithmics. http://law.di.unimi.it.

[20] Nilesh Jain, Guangdeng Liao, and Theodore L. Willke. Graphbuilder: Scalable graph etl framework. In *First International Workshop on Graph Data Management Experiences and Systems*, GRADES '13, pages 4:1–4:6, New York, NY, USA, 2013. ACM.