クエリログの部分的利用を考慮したグラフの集約演算高速化

平方　俊行[†]　　楠　　和馬[††]　　波多野賢治[†††]

† 同志社大学文化情報学部　〒 610-0394 京都府京田辺市多々羅都谷 1-3
†† 同志社大学大学院文化情報学研究科　〒 610-0394 京都府京田辺市多々羅都谷 1-3
††† 同志社大学文化情報学部　〒 610-0394 京都府京田辺市多々羅都谷 1-3
E-mail: †biq0035@mail4.doshisha.ac.jp, ††kusu@mil.doshisha.ac.jp, †††khatano@mail.doshisha.ac.jp

あらまし　近年のソーシャルネットワーキングサービス (SNS) の普及により人やモノ同士の関係性に着目した分析の需要が増加している．そのため，大規模なグラフデータを管理することが可能なグラフデータベース (GDB) では，探索演算と集約演算を順番に実行することでグラフ分析が行われる．グラフ分析のクエリ処理高速化のため，GDB にはクエリログから取得したクエリとその計算結果を保持するキャッシュ機能がある．しかしキャッシュ内には，最終的な計算結果しか保存していないため，クエリの内容が変更されると利用できない．そこで本研究では，クエリログより取得したグラフデータから各節点の入次数を保持しておき部分的に集約演算処理の代用を行う方法を提案する．
キーワード　ソーシャルデータ，グラフデータベース，集約演算

1　は じ め に

近年，Facebook[1] や Twitter[2] といった企業が提供している SNS の普及に伴い，ユーザ間の人間関係の情報を示すソーシャルデータと呼ばれる複雑な構造を持つデータが増加している [1]．ソーシャルデータは，一般的にグラフと呼ばれるデータ構造を用いて表現される．グラフは，データで取り扱っている実体とそれらの関係をそれぞれ節点と辺で表すことで，表現の幅を広げることが可能である [2]．また，ソーシャルデータから得られる人やモノの繋がりを分析することでユーザの嗜好や大衆の傾向を把握できるため，経営や運営において優位性が得られる [3]．そのような背景から，ソーシャルデータに対する分析の需要が増加している．

また，グラフデータの増加に伴いグラフに対するデータ管理やグラフの演算処理に最適化されたデータベースシステムである GDB が開発されている [4]．GDB では該当する節点や辺を取得するために探索演算を実行後，その部分グラフを要約するために集約演算を実行することで節点や辺の総数や経路の集計といった処理を行うグラフ分析のクエリが実行される．このようなグラフ分析のクエリには，各節点が保持している属性値に対して条件をつける場合と節点ごと集計して属性値に対して条件をつけない場合が存在する．

紹介したグラフ分析のクエリを高速に処理するために，探索演算に対しては多くの手法が提案されている [5,6] 一方，集約演算の高速化に関する研究は少なく手法の確立がなされていない．そのため既存の GDB では，集約演算において探索演算の処理で得られたグラフの全ての節点や辺へ参照を行い集計を行うため多くの処理時間を要する．

データ分析のクエリを高速化する GDB での機能の一つにキャッシュがある [7] この機能は，発行されたクエリの集合であるクエリログを基にその処理手順と演算結果を保持しておき，以降にクエリログと同じクエリが発行された際にその保持しているデータを再利用することで，データベースへの参照回数を削減する機能である．つまり，初回のデータ参照ではデータベースからメモリに読み込む必要があるが，それ以降はメモリ上にあるキャッシュされた演算結果を返すため，高速な応答が可能である．キャッシュは，計算結果を吟味しながら条件を変えて何度も集計処理を実行する [8] データ分析においては，同一のデータに対して参照が発生する可能性が高くなるため使用可能な状況は多い．しかし既存の GDB では，キャッシュはクエリとその計算結果しか保存しておらず，クエリが問合せがデータがキャッシュと完全に一致しなければ再利用できないため，キャッシュの中に再利用可能な部分グラフが存在する状況が限られるという問題がある．

そこで本研究では，グラフ分析に関するクエリの中でもまず節点ごとに集計する場合のクエリに対して対応を行う．そのため，集約演算において既存手法では処理できないクエリとクエリログより得られたクエリが部分的に一致する場合であっても演算結果の再利用が可能となる演算処理の代用法を提案する．

2　関 連 研 究

グラフ探索演算の処理高速化を図った研究は多数存在しているが，グラフ分析の集約演算の高速化に向けた研究は少ない．そこで本節では，探索演算実行時の節点や辺への参照数削減に関する研究の中から，提案手法の着想を得た研究を紹介する．

グラフの探索の高速化に関する研究には，事前計算結果を利用して演算処理の代用を行い処理時間の高速化を図る手法が存在する [5]．GDB 内での探索演算は，全ての節点や辺に対して参照を行う必要があるため問題である．よってこの研究は，経路を事前計算しておくことで GDB 内で実行される探索演算の

1 : Facebook. https://www.facebook.com/ (閲覧日 2020/3/17)
2 : Twitter. https://twitter.com/ (閲覧日 2020/3/17)

処理回数の削減が可能である．事前計算は，同様なグラフデータに対して再帰的に探索を実行しており GDB への参照回数の削減が見込めるため，探索演算の処理高速化が可能となる．

また Rodriguez らは，ユーザが発行したクエリを集めたクエリログには規則性があることに着目して，事前にグラフデータを取得する手法を提案している [6]．グラフ分析ではグラフの探索は再帰的に行われるため，事前にクエリログから特徴的なクエリの結果を取得することで同様の結果を取得することで同様のクエリが発行された際に GDB 内を参照することなく結果を返すことができ，処理時間の短縮につながる．この手法ではまず，クエリに対する実行手順の記録を取りその記録に対して分析を行う．次に，その分析結果から頻繁に参照されるグラフ構造を算出しシステム内に保存した後，クエリにおいて代用可能なグラフ構造の選定を行う．最後に，ユーザから与えられたクエリに対して処理の代用ができる部分を抽出してクエリ最適化を行う．このようにクエリログを解析して，そこから取得した演算結果を保存することは事前計算の方法として有効であることが分かる．

しかし，文献 [6] では，グラフデータに対するクエリ内で実行される集約演算の高速化に着目していない．よって，各節点や辺への参照数の削減を行えず，集約演算においてクエリログを部分的に代用することができないため，クエリを効率的に実行できないという問題がある．

3 提案手法

本研究では，キャッシュがクエリログと実行するクエリグラフデータが一致しないと利用できないという既存手法の問題を解決するために，節点ごとに集計を行うクエリに対して事前計算の保存手法とその代用法の提案を行う．そのために既存手法と同様にクエリログから得られた計算結果を利用するが，本研究ではさらにクエリログを部分的利用を可能にすることで，処理の既存手法よりも多くの状況において処理の代用を可能とする．

3.1 前提知識

本研究で扱うグラフは，プロパティグラフ $G(V, E, \mathtt{p}, \mathtt{l})$ を前提としている．このグラフは，節点の集合 V と辺の集合 E から成るラベル付き属性付き有向グラフ G である．各節点 $v \in V$ には節点のエンティティを示すラベル $\mathtt{l}(v)$ と属性情報 $\mathtt{p}(v)$ が付与されており，新しい節点を作成する時は，$G.\mathtt{create_node()}$，新しい辺を作成するときは，$G.\mathtt{create_edge()}$ とする．$\mathtt{p}(v)$ は辞書型で管理されており，v に隣接する節点 v_n は $(v, v_n) \in E$ と表す．その際，新しい節点を作成する時は，$I.\mathtt{create_node()}$ とする．また，新しい辺を作成する時は，$I.\mathtt{create_edge()}$ とする．

節点に接続する辺のうちその節点を終点とする辺の総数である入次数の関数は，ある節点 $v \in V$ とすると $\mathtt{cnt_in_degree}(v)$ と表す．この関数は式 (1) で定義される．

$$\mathtt{cnt_in_degree}(v) = |\{v_s \mid (v_s, v) \in E\}| \qquad (1)$$

アルゴリズム 1 事前計算の保存方法

Input: 部分グラフ：$G_q = (V_q, E_q, \mathtt{p}, \mathtt{l})$, 集計ラベル：$\lambda$
Output: 事前計算グラフ：$I = (V', E', \mathtt{p}', \mathtt{l}')$

```
1: for {v_s | ∀v_s ∈ V_q(l(v_s) = λ)} do
2:   queue ← ∅
3:   queue.push(v_s)
4:   v ← I.create_node()
5:   p'(r)["AggregatedLabel"] ← λ
6:   v' ← I.create_node()
7:   while queue ≠ ∅ do
8:     v ← queue.pop()
9:     in_degree ← G_q.cnt_in_degree(v)
10:    out_degree ← G_q.cnt_out_degree(v)
11:    p'(v')[l(v)][v] ← (in_degree, out_degree)
12:    neighbors ← {v_n | (v, v_n) ∈ E_q}
13:    for v ∈ neighbors do
14:      queue.push(v)
15:    end for
16:  end while
17:  e' ← I.create_edge()
18:  e'[v][v'] ← v_s
19: end for
20: return I
```

また，始点とする辺の総数である出次数の関数は v $\mathtt{cnt_out_degree}(v)$ と表す．この関数は式 (2) で定義される．

$$\mathtt{cnt_out_degree}(v) = |\{v_e \mid (v, v_e) \in E\}| \qquad (2)$$

3.2 事前計算の保持方法

集約演算は，探索演算によって各節点から辺を辿りグラフの経路を取得してそこから該当する節点や経路を集計する．そのため，各節点の入力辺や出力辺の本数を計算することで節点や経路の数を集計することができる．そこで本研究では，集約演算処理の代用を行うために各節点に接続されている辺の総数を事前に集計しておき，節点情報と共にメモリ上に保存することで，集約演算時に事前計算結果を可能な限り再利用できる方法を提案する．具体的なアルゴリズムはアルゴリズム 1 に示す．

入力は，クエリ Q より取得した該当するグラフ G_q と集計を行う節点が持つラベルである集計ラベル λ とする．出力は，グラフ G_q の集計結果を保存した事前計算結果の保存先をグラフ I とする．経路の長さが変化しても部分的に再利用可能とするために，経路によって変化することがない接続されている辺の数を事前に計算して保存する．

アルゴリズム 1 ではまず，クエリログから集計ラベルを取得してそのラベルをもつ節点を起点とした事前計算を行うことで，経路長が変化したとしても集計ラベルの集計を高速に実行できる．そこで，集計ラベルをもつ節点をグラフ G_q から取得して，その節点を起点としてグラフを分解するために，アルゴリズム 1 の 1 行目では， GDB のクエリ言語を用いてクエリログより取得した集計ラベル λ を持つ節点を取得する．2 から 3 行目, 7 から 14 行目では，各節点の入・出次数を事前に集計する．集約演算は経路長が変化すると，同じラベルを持つ節点

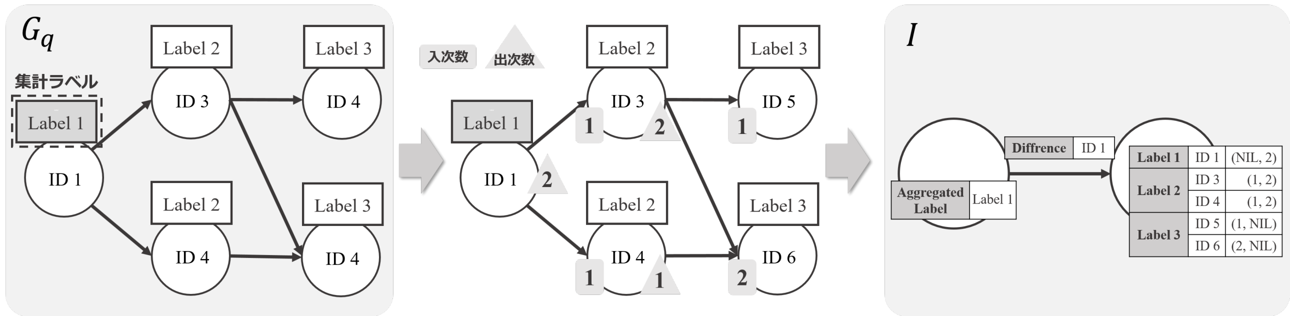

図 1: クエリログより取得したグラフデータの事前計算

を集計しても該当する経路を持っていない節点が存在するため集計結果が変化する．そのため，節点だけを事前に集計しても演算結果が変化するため使用できる状況が少ない．そこで，集計結果が節点に接続する辺の次数によって決まることに着目して，節点だけでなく，各節点の辺数を事前に計算する．図 1 では，各節点に対する集計方法を示す．ID 1 の入次数はなく，出次数は 2 本となる．また ID 1 につづく Label 2 の ID 3 と ID 4 の入次数はそれぞれ 1 本と 1 本となり，出次数はそれぞれ 2 本と 1 本であることが分かる．また ID 1 につづく Label 2 の ID 3 の入次数と出次数はそれぞれ 1 本と 2 本となり，ID 4 の入次数と出次数はそれぞれ 1 本と 2 本となる．それ以外の節点も同様に導出することができる．

6 と 9 行目では，節点内に様々なデータを保持できるプロパティグラフ である事前計算グラフ I に節点情報やその集計結果を保存する．

さらに 15 から 16 行目で，集計結果を保存している節点と集計ラベルを保存している節点を辺で結ぶ．集計ラベルは，集約演算の処理代用においてどの節点に対する事前計算結果であるのか判定するために保存している. 図 1 では，事前計算結果をプロパティグラフモデルへ保存する例を示す．各節点には，各節点のラベルと図 1 で示した次数の集計結果を保存する．そして各辺は集計ラベルの情報を持つ節点と事前計算結果を保持している節点で結び，どのような節点が起点となるか辺に保存する．

上記のアルゴリズムにより，集計ラベルが保存されている節点を中心とした事前計算結果を保存したグラフが構築される．

3.3 事前計算結果の代用

集約演算時に節点や辺への参照を行う前に事前計算の結果を用いて一部集計を代用することで，GDB での節点の参照回数を削減することが可能になるため，演算の回数を削減することが可能となる．そのため，クエリが与えられたとき，該当する部分グラフに対して集約演算を行う際に処理の代用を行うフローチャートを図 2 に示す．

フローチャートの実行手順に関して解説する．まず，クエリが与えられた際に処理の代用が可能である事前計算結果を保持しているか判定を行う．その際，Aggregated Label に保存されている値を参照してどのような節点に対して集計を行ったクエリログを保存しているのか参照を行う．その中で，該当する

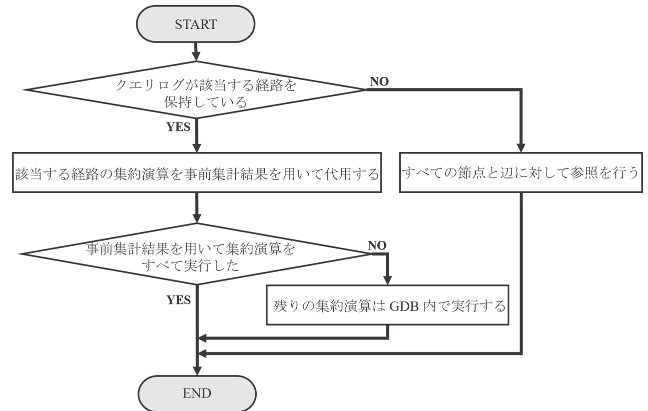

図 2: クエリログの代用手順

ラベルを持つ節点に対して事前集計を行っていることが判明した事前計算結果を全て列挙する．次に，列挙された事前計算結果がクエリに対して適用できるかどうか判定を行う．列挙された事前計算結果がクエリに対して適用することができない場合，事前計算結果を使用することなく全ての節点と辺に対して参照を行う．適用することができる場合，事前計算をクエリに適用する．集約演算処理の代用は，全ての節点や辺に対して代用できず，部分的なグラフの集計のみ実行できる場合がある．そのため，全ての節点や辺を代用できているのかどうか GDB に対して問合せをする．集計しきれていない節点や辺が存在した場合，GDB 内で残りの集約演算処理を実行する．このような手順を実行することでメモリ上に保持している事前計算結果を用いて集約演算処理の代用が可能となる．

図 2 より，プロパティグラフモデルに保存された演算結果を参照して集約演算処理の代用を行うことで，GDB 内のみで実行されている節点や辺への参照方法よりも節点や辺への参照回数を削減することができる．そのため，全ての節点や辺へ参照を行い演算を実行する必要となる既存手法に比べて，節点や辺への参照数の削減が可能であるため処理時間の高速化が見込まれる．

4 評 価 実 験

本節では，提案手法がグラフデータに対するデータ分析のクエリの処理性能にどのような効果をもたらすのか明らかにするために実験を行う．そこで，集約演算の処理時間だけでなく，

事前計算の性能についても評価する必要がある．よって，提案手法を適用することで集約演算時に処理性能が改善されたかどうか評価を行うために，事前計算の作成時間と処理実行時間を計測する．本実験の実行環境を表 1 に示す．

表 1: 実行環境

CPU	Intel Xeon Processor 3.39 GHz 8 core ×1
OS	CentOS 7.7 x86
RAM	64 GB
GDB	Neo4j 3.5.13

実験に使用する GDB には，オープンソフトウェアであり既存の GDB の中では最も汎用的に使用されている [3] Neo4j 社の Neo4j [4] を採用する．

4.1 データセット

データセットは，非営利団体の Linked Data Benchmark Council (以下，LDBC) が提供している ソーシャルネットワークベンチマーク (以下，LDBC SNB) を用いる [9]．LDBC は，近年のソーシャルデータに対する需要の増加により，グラフ処理や分析性能の測定が可能となるデータを作成する団体である．その中でも LDBC SNB は図 3 に示す通り，グラフにおいて代表的な SNS サイトで管理される人間やモノ同士の関係についてモデル化したデータセットである．

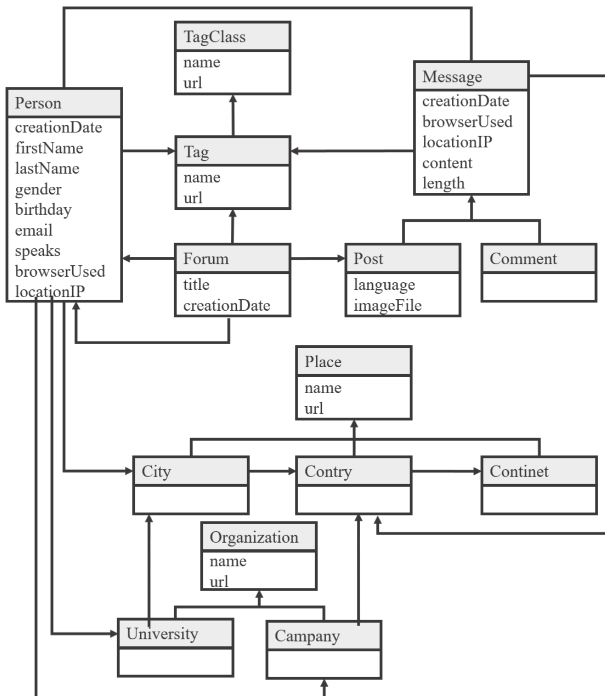

図 3: データセットの詳細

LDBC SNB は，さまざまな規模のデータセットに対してクエリの性能を測るため Scale Factor (以下，SF) の値によりグラフデータの規模を変換することができる．そこで本研究では，データサイズの変化により提案手法の有用性に変化が生じることを考慮して LDBC SNB のデータ生成プログラムを使用し，SF 値が 0.1, 1 に対応するデータセットを作成する．各 SF 値のデータセットを LDBC SNB のデータ格納のパラメタ作成プログラムを利用し，Neo4j に格納する．データ生成プログラムを用いて生成されたデータセットの節点と辺とプロパティの詳細を表 2 に示す．

表 2: データセットの規模

SF	節点数	辺数
0.1	327,588	1,477,965
1	3,181,724	17,256,038

4.2 実 験 内 容

提案手法は，クエリログから事前に演算結果を取得することを想定しているため，クエリログと実験で使用するクエリが必要である．提案手法は，属性への指定がないクエリを想定した手法である．そこでその性能を評価するために，集約演算処理を行うクエリの中でも属性を利用していない処理を実行するクエリをクエリログとして作成した．詳細を表 3 に示す．このクエリは，ユーザが投稿しているメッセージに対して各ユーザの投稿数を集計を実行している．

表 3: クエリログの詳細

MATCH (k:Person)←(n:Forum)→(m:Message)→(l:Tag) RETURN k, count(m)

また評価実験では，クエリログが使用可能な状況における処理時間の測定を行いたい．そこで，既存手法ではクエリログを使用することができない，経路が部分的に一致している属性への指定がないクエリを 3 種類作成した．query 1, 2 では，経路長がクエリログよりも短い場合において事前計算結果の使用が可能かどうか確認するために，経路長を変化させたクエリを作成した．query 3 では，経路長がクエリログと同等で集計条件が変わった場合においても事前計算結果の利用が可能かどうか確認するために，集計条件を変化させたクエリを作成した．詳細を表 4 に示す．

表 4: クエリの詳細

query 1	MATCH (k:Person)←(n:Forum)→(m:Message) RETURN k, count(n)
query 2	MATCH (k:Person)←(n:Forum) RETURN k, count(n)
query 3	MATCH (k:Person)←(n:Forum)→(m:Message)→(l:Tag) RETURN k, n, count(l)

そして取得したクエリログを事前に実行しておき，その計算結果をメモリに保持する．その後，各クエリをそれぞれ 10 回ずつ試行して，その実行時間の平均を算出して既存手法のキャッシュ機能を有効にした場合と比較する．

3 : DB-Engines Ranking - Trend of Graph DBMS Popularity. https://db-engines.com/en/ranking_trend/graph+dbms (閲覧日 2020/3/17)
4 : Neo4j. https://neo4j.com/ (閲覧日 2020/3/17)

4.3 実験結果

実行時間の計測結果を表 5 に示す．この表は，各データセットに対して 3 種類のクエリを実行した処理時間を示している．

表 5: 実験結果

	SF 0.1			SF 1		
	提案手法	既存手法	高速化率	提案手法	既存手法	高速化率
query1	0.070	0.808	**11.542**	0.873	9.809	**11.236**
query2	0.203	0.801	**3.946**	0.232	10.079	**43.444**
query3	0.365	3.872	**10.608**	0.912	12.923	**14.170**

クエリログと部分的に一致しているクエリの場合は，事前計算結果を使用することで GDB での各節点への参照回数が削減されるため，実行時間が最大約 40 倍高速化されたことが分かる．また，本研究で使用したデータセット程度の節点数や辺数の場合は，事前計算の結果をメモリ上に保存して使用することも可能であった．そのため，既存手法よりも多くの場合で提案手法は使用可能であることが分かる．

5 おわりに

本研究では，グラフデータに対してデータ分析を実行するクエリにおける集約演算の処理高速化を目的として，既存手法よりも多くの状況において演算処理の代用を実行可能となる手法の提案を行った．属性への指定がないクエリに対して評価実験を行った結果，既存手法よりも最大で約 40 倍高速に処理が可能となった．よって，本研究で対象としていた属性への指定がないクエリへの対応が可能となったため既存手法よりもクエリログを使用できる状況が多くなったと考えられる．

今後は対象としていなかった属性への指定があるクエリの事前計算算出方法のアルゴリズムについて考慮する必要がある．また，提案手法のスケーラビリティについて評価するため，ベンチマークの SF をより大きくして実験を行う必要がある．

謝辞

本研究の一部は，JSPS 科研費 JP18H03342 の助成を受けたものである．ここに記して謝意を表す．

文献

[1] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases: New Opportunities for Connected Data*. O'Reilly Media, 2015.

[2] Robin James Wilson. *Introduction to Graph Theory*, Vol. 5. Prentice Hall, 2015.

[3] Amine Ghrab, Oscar Romero, Salim Jouili, and Sabri Skhiri. Graph BI & Analytics: Current State and Future Challenges. In *International Conference on Big Data Analytics and Knowledge Discovery*, Vol. 11031, pp. 3–18, 2018.

[4] Renzo Angles and Claudio Gutierrez. Survey of Graph Database Models. *ACM Computing Surveys*, Vol. 40, No. 1, pp. 1–39, 2008.

[5] Sherif Sakr and Ghazi Al-Naymat. The Overview of Graph Indexing and Querying Techniques. In Sherif Sakr and Ghazi Al-Naymat, editors, *Graph Data Management: Techniques and Applications*, chapter 3, pp. 71–88. IGI Global, 2012.

[6] Joana Matos Fonseca da Trindade, Konstantinos Karanasos, Carlo Curino, Samuel Madden, and Julian Shun. Kaskade: Graph Views for Efficient Graph Analytics, 2019.

[7] Dana Canzano. Understanding the query plan cache. `https://neo4j.com/developer/kb/understanding-the-query-plan-cache/` (閲覧日 2020/3/17).

[8] Peixiang Zhao, Xialolei Li, Dong Xin, and Jiawei Han. Graph cube: on warehousing and OLAP multidimenJsional networks. In *the 2011 ACM SIGMOD International Conference on Management of Data*, pp. 853–864, 2011.

[9] Alexandru Iosup, Tim Hegeman, Wing Lung Ngai, Stijn Heldens, Arnau Prat-Pérez, Thomas Manhardto, Hassan Chafio, Mihai Capotă, Narayanan Sundaram, Michael Anderson, Ilie Gabriel Tănase, Yinglong Xia, Lifeng Nai, and Peter Boncz. LDBC Graphalytics: A Benchmark for Large-scale Graph Analysis on Parallel and Distributed Platforms. *Proceedings of the VLDB Endowment*, Vol. 9, No. 13, pp. 1317–1328, 2016.