大規模 RDF データに対するクエリ分散処理における値域情報利用の検討

金子 舟[1,a)]　千代 英一郎[1,b)]　高野 保真[2,c)]

概要：大規模な RDF データに対する検索処理を効率的に行うため，複数のサーバを用いた分散処理によって，クエリの処理性能を向上する試みがなされている．その一手法として，クエリの結果が取り得る値域に着目して処理を分割する手法の効果が確認されている．この手法では，事前にクエリの結果として取り得る値の集合のグラフ（値域グラフ）を求めておき，その値域グラフを処理の分割の指標として分散検索を行う．RDF データ自体は分割せずに済むため導入がしやすく，値域グラフを用意することで，細粒度のクエリを繰り返す必要はないという利点がある．しかしながら，対象の RDF グラフが大規模であるほど，値域グラフの作成を実用的な時間，メモリ使用量で行うことは難しく，利用上の課題となっていた．本研究は，値域に注目した分散処理手法における値域グラフの生成に関する課題に対し，グラフを分割し分散処理によって値域グラフを求める手法を提案する．また，その手法を実際の RDF グラフに適用したときの予備評価について報告する．

キーワード：RDF グラフ，分散処理

1. はじめに

Web 上にある情報を統一に記述する枠組みとして，RDF（Resource Description Framework）と呼ばれる枠組みが W3C によって規格化されている [5]．RDF は，主語，述語，目的語の 3 つ組で各情報を表現し，この 3 つ組に対して主語を始点ノード，述語を辺（エッジ）へのラベル，目的語を終点ノードとするグラフ構造（RDF グラフ）で表現する．そのような RDF グラフに対して，SPARQL[7] と呼ばれる標準の問い合わせ言語を用いて検索/操作などを行う．

[1] 成蹊大学大学院 理工学研究科
[2] 成蹊大学 理工学部
[a)] dm166202@cc.seikei.ac.jp
[b)] chisiro@st.seikei.ac.jp
[c)] yasunao-takano@st.seikei.ac.jp

RDF が普及し，検索対象となる RDF グラフが大規模になるにつれて，複数のサーバを用いた分散処理によって，クエリの処理性能を向上する試みがなされている [1][3]．これらの手法では，RDF データを分割し分散処理するもの，クエリを分解して処理を行うものなどがある．RDF データを分散する手法では，述語ごとにデータを分割し，分散して配置することでマスター/ワーカー型の処理によって性能向上が得られることが確かめられているが，複雑なクエリに対して大きな性能低下が起きる可能性がある．また，クエリを分解する手法は，クエリのパターンをあらかじめ分解して細粒度のクエリを繰り返すことで処理性能の向上を狙っているが，クエリによっては分散のオーバーヘッドが高く，十分な性能が得られない．

それに対して，我々はクエリ処理の結果となる

変数の取り得る値集合（以降では**値域**とよぶ）に着目し，これを分割することで分散処理を実現するアプローチに取り組んでいる [8]．この手法では，事前にクエリの結果として取り得る値の集合のグラフ（**値域グラフ**）を求めておき，その値域グラフを処理の分割の指標として分散検索を行う．RDF データ自体は分割せずに済むため導入がしやすく，また，値域グラフを用意することで，細粒度のクエリを繰り返す必要はないという利点がある．既に標準的なベンチマークである LUBM[2] においても，ほぼ線形に近い台数効果が得られている．しかし，そのような利点がある一方で，値域グラフの作成に必要な時間・メモリ効率の両面で，実用上は改善の余地があった．また，分散検索処理の性能においても，検索処理量が少ない場合のオーバーヘッド，結果集合が大きい場合の転送処理にかかる時間においては，並列処理した際の台数効果で改善の余地があった．

本研究では，値域に注目した分散処理手法をベースとして，RDF クエリの処理性能の向上をめざす．ここではまず，先行研究の値域分割によるクエリの分散処理について触れ，中でも本研究の主題である値域グラフの生成について説明する（2 節）．続いて，値域グラフの生成に関する課題に関し，対象の RDF グラフから部分的な値域グラフを生成して，それらを統合することで，近似的な値域グラフの生成を行う方法を示す (3 節)．また，提案手法の実装に関する部分（4 節）や，改善効果についての予備的な実験の結果を示す (5 節)．実験の結果，分散処理による値域グラフ生成時間の短縮が確認された．

2. 値域分割によるクエリ分散処理

先行研究 [8] の値域グラフを用いたクエリ分散処理では，前処理として，クエリを解析し，値域グラフを作成する．その後，作成した値域グラフを元に，クエリ処理を分散して，部分的な検索結果を得る．最後に，得られた検索結果を統合し，全体のクエリ処理の結果とする．本節では，本研究の主題である値域グラフの生成に重点を置いて，分散検索処理の概要を説明する．

```
 1: procedure CONTRACTION(G)
 2: α := λx.⊥
 3: for all D ∈ N(G)/ ∼ do
 4:     a := freshURI
 5:     for all n ∈ D do
 6:         α(n) := a
 7:     end for
 8: end for
 9: for all p ∈ { p | (s, p, o) ∈ G } do
10:     α(p) := p
11: end for
12: return α(G)
13: end procedure
```

（$freshURI$ はこれまで出現していない URI を返す）

図 1 値域グラフ生成アルゴリズム

2.1 値域グラフの生成

値域グラフは，クエリの結果となる変数の述語集合が同一のものをグループとしてまとめたものであるため，図 1 に示す生成アルゴリズムで生成できる．

手続き CONTRACTION は RDF グラフ G を受け取り，その値域グラフを生成する．関数 α はグループ化関数であり，G のノード集合 $N(G)$ に対し，接続しているエッジのラベル集合が同一のノードをまとめてひとつのノードとする．ここで $\sim$ は以下のように定義される同値関係である．

$$n_1 \sim n_2 \Leftrightarrow I(n_1) = I(n_2) \wedge O(n_1) = O(n_2)$$

ただし，

$$I(n) = \{\ p \mid (\_, p, n) \in G\ \}$$

$$O(n) = \{\ p \mid (n, p, \_) \in G\ \}$$

$I(n)$ はノード n に入るすべてのエッジへのラベル，$O(n)$ はノード n から出るすべてのエッジへのラベルである．RDF グラフではエッジのラベルは述語であることに注意されたい．また，関数 α はグループ化関数であり，2-11 行目のように定義され，12 行目でグラフ G に対する値域グラフを返す．

このアルゴリズムにより求められる値域グラフと，その元の RDF グラフの例を図 2 に示す．この RDF グラフでは，ノード（主語，目的語）に人物，エッジ（述語）にその関係性が記述されて

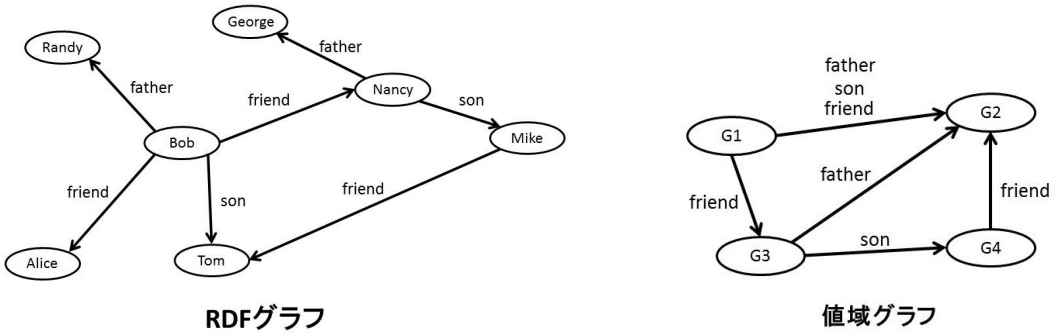

図 **2**　RDF グラフとその値域グラフ

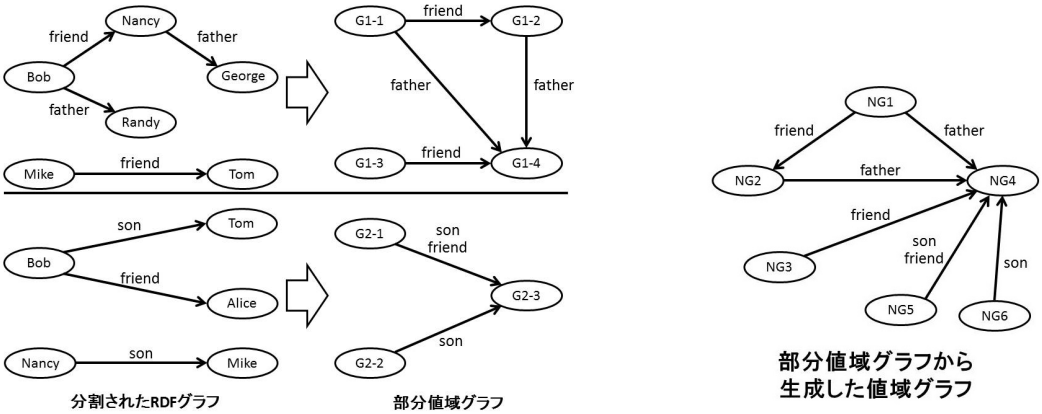

図 **3**　部分値域グラフの生成と，それらから生成した値域グラフ

いる．例えば，Bob の息子（son）は Tom である．このとき，値域グラフのノードの各要素は，

$G_1 = \{Bob\}$,
$G_2 = \{Randy, Alice, Tom, George\}$,
$G_3 = \{Nancy\}$,
$G_4 = \{Mike\}$

である．先ほど例に挙げた関係性で言えば (Bob, son, Tom) は (G1, son, G2) の関係に該当する．また，RDF グラフ中のエッジへのラベルが，friend, father，son の 3 種類であるので，値域グラフもその 3 種類のエッジで表現される．この例から分かるように，値域グラフは RDF グラフの要約であるといえる．

2.2　分散検索処理

得られた値域グラフを基に，以下の手順で分散

検索処理を行う．
(1) 値域グラフから，クエリ変数の値域を求める
(2) 値域を分割し，各ワーカーサーバで分散検索する
(3) 各ワーカーサーバで得られた結果を統合する

3. 提案手法

2 節の図 1 で示した値域グラフ生成アルゴリズムについて，最終的に行うグラフの変換 $\alpha(G)$ までの間，複数回にわたってグラフ全体を参照する必要があるため，グラフの規模が大きくなるほど処理量が増加する．一方で取り扱えるデータの規模はメモリ量による制約を受けるため，この点が大規模 RDF グラフに対する値域グラフ生成の課題となっていた．

しかしながら，何らかの方法でグラフサイズを小さくすることができれば，処理量の増加を避け，かつメモリ使用量を抑えることが可能である．そこで本論文が提案するのは，対象の RDF グラフを複数に分割し，それぞれの部分ごとの値域グラフ生成を行った後，各値域グラフを統合してから，グラフ全体の値域グラフを生成する手法である．また，部分値域グラフの生成を分散処理とすることで，短時間での値域グラフ生成が見込まれる．

手順は次の通りである．
(1) 対象の RDF グラフを任意の数 n に分割する．
(2) 各部分グラフに対して図 1 のアルゴリズムで部分値域グラフを生成する．
(3) 生成された部分値域グラフを統合する．
(4) 統合後のグラフに対し，再度値域グラフ生成アルゴリズムを適用する．

まず（1）について，元となる RDF グラフを分割する．分割の位置や，条件等によっては，その後の部分値域グラフ生成の結果に作用する懸念もあるが，本論文では各処理ノードへの処理量を均等に割り当てすることとしたため，分割位置については特に制限を設けていない．

続いて，(2) 部分値域グラフ生成を分散処理として行う．ここでのアルゴリズムは前節で説明した値域グラフ生成のものを利用する．生成後の戻り値は，元のグラフのノードが，部分値域グラフにおいてどのノードにグループ化されたのかの対応を示すマップと，部分値域グラフにおけるグループ間の 3 つ組集合である．また，各処理ノードへは重複のない値を付与し，戻り値の識別が可能であるようにする．

次に（3）で，(2) の戻り値であるグループ間の 3 つ組集合を 1 つに統合する．同じエッジ（述語）をもつ 3 つ組が存在しても，この時点ではグループ間の識別が行えるため，重複は発生しない．

最後に（4）の操作を行うことで，同じエッジを持つグループ同士を再グループ化する．新たにできたグループの構成要素の取得には，(2) の戻り値であったノード集合マップを利用し，元のノードの構成要素を新たなグループの要素に再編する．

こうした手順により，部分値域グラフ同士を統合して，グラフ全体の値域グラフを生成する．

図 2 に示したグラフを，2 つに分割した場合の部分値域グラフの生成，および生成された部分値域グラフを統合することで求めたグラフ全体に対しての値域グラフを図 3 に示す．図のうち，部分値域グラフにおける各ノードの要素は，

$G_{1-1} = \{Bob\}$,
$G_{1-2} = \{Nancy\}$,
$G_{1-3} = \{Mike\}$,
$G_{1-4} = \{Randy, George, Tom\}$,
$G_{2-1} = \{Bob\}$,
$G_{2-2} = \{Nancy\}$,
$G_{2-3} = \{Alice, Tom, Mike\}$
である．

また，統合後の値域グラフにおける各ノードの要素は，

$NG_1 = \{Bob\}$,
$NG_2 = \{Nancy\}$,
$NG_3 = \{Mike\}$,
$NG_4 = \{Randy, George, Tom, Alice, Mike\}$,
$NG_5 = \{Bob\}$,
$NG_6 = \{Nancy\}$
である．ここで生成される値域グラフは，対象のグラフを分割せずに生成した値域グラフと必ずしも同型ではない．しかしながら，対象のグラフを構成していた 3 つ組はすべて含まれているため，

表 1　各グラフのサイズ

グラフの種類	ノード数	エッジ数
dbpedia	750,268	377,314
dragbank	281,071	517,023

近似的に値域グラフとして利用することが可能である．例えば，2 節で用いた例 (Bob, son, Tom) はこの場合，(NG_5, son, NG_4) という関係性として保たれていることが分かる．

4. 実装

実装には Scala を用いた．部分値域グラフの生成を分散して行うため，アクターモデルにもとづく分散処理基盤である akka[4] を用いたほか，集合への変換など単純な処理には Scala の並列コレクションを取り入れて，値域グラフ生成時間の短縮を図った．5 節の評価で後述するが，提案手法との比較として，グラフの分割を行わない場合の値域グラフ生成プログラムは，akka を用いずに並列コレクションクラスのみ実装している．

5. 評価

前節で説明した提案手法による値域グラフ生成時間短縮の効果を確認するため，表 1 に示す 2 つのグラフを対象として，値域グラフの生成を行った．この 2 つはどちらも N トリプル形式 [6] であり，主語，述語，目的語の各情報が 3 つ組（トリプル）で記述されている．dbpedia は，dbpedia の日本語版アブストラクト（2014 年）で，主語と述語は URI，目的語には記事内容のリテラルが記述されている．drugbank は，薬剤の製品名や化合物，特許関連情報などに関する内容が記録された RDF グラフで，主語，述語，目的語のいずれも URI で記述されている．

評価に用いた環境は，CPU: AMD Opteron 6386SE 2.8GHz (32 cores)，OS: CentOS 6.4 (2.6.32-358.el6.x86_64)，Memory: 256GB である．

分割数を 8 から 64 まで変えて実行時間を測定した結果を図 4 に示す．比較のため，分割せずに並列化だけ行うもの par の値も載せている．実行時間の内訳は図中の凡例にある通り，下から「読み込み・格納時間」,「部分値域グラフ生成」,「部分値域グラフ統合」,「ノード集合の生成」,「ノード間のエッジ生成」の 5 つである．これらのうち，分割を行わない par については，「部分値域グラフ生成」のラベルに値域グラフ生成の実行時間を，「部分値域グラフ統合」は行わないため 0 秒として取り扱っている．

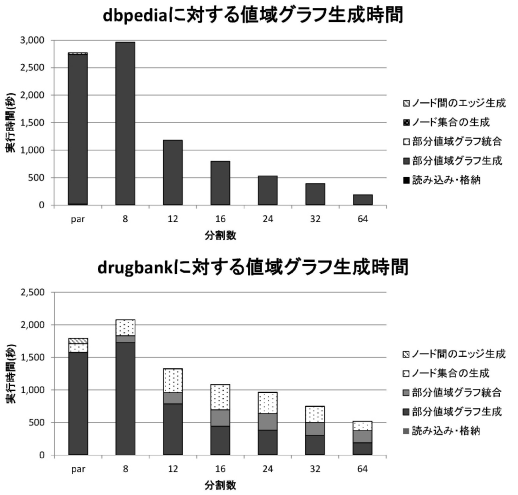

図 4　実験結果

実行結果より，dbpedia と drugbank のどちらの場合においても，分割数をより多くした方が生成時間の短縮には効果的であることがわかる．これは，3 節でも述べたように，与えられるグラフの大きさが小さくなるほど，処理量が少なくて済むためであると考えられる．この差は，分割しないで値域グラフの生成を行った par と，64 分割の間で特に顕著である．今回用いたグラフに対しては，概ね 12 分割以上であれば統合する時間も含めて，そのまま値域グラフを生成するよりも効果的であると言える．一方で 8 分割を下回る場合は， 部分値域グラフの作成に大きく時間を要しており，分散処理の効果が出にくい．

各グラフに対する値域グラフ生成時間の内訳を見てみると，dbpedia ではそのほとんどが部分値域グラフの生成であるのに対し，drugbank では，部分値域グラフの生成，部分値域グラフの統合，ノード集合の生成に時間を要している．この違いは，表 2 に示すように，分割時の値域グラフでの

表 2 分割数ごとの生成された値域グラフのサイズ

	par	8	12	16	24	32
dbpedia						
値域グラフのノード数	2	2	2	2	2	2
値域グラフのエッジ数	1	1	1	1	1	1
dragbank						
値域グラフのノード数	3,420	32,908	39,956	39,981	33,164	27,089
値域グラフのエッジ数	143,333	269,488	261,441	233,431	174,632	137,776

述語数の違いが原因である．また，dbpedia の場合，述語数（エッジ数）が 1 つであるため，分割数にかかわらず生成される値域グラフは全て同じであったが，drugbank では par と 提案手法の間で差が生じている．値域グラフは，あくまで高速な検索を行うための準備処理ではあるが，グラフをそのまま要約したときと比べて，ノード，エッジとも大きく増加しており，元のグラフとの近似性や，検索に対する有効性の差に関して今後検討していく必要がある．

6. おわりに

本研究では，値域に注目した RDF の分散検索手法における値域グラフの生成時間の改善について， 対象の RDF グラフから部分的な値域グラフを生成し，それらを統合することで，近似的な値域グラフの生成を行う方法を提案した．提案手法の Scala による実装により，改善効果についての予備的な効果を確認した．対象のグラフに対し分割数を大きくとることで，より短時間に値域グラフの生成を行うことができる一方，次のような課題が挙げられる．

- データ分割・クエリ分割手法との比較
- 検索量と転送量について
- 台数効果の測定

参考文献

[1] Franke, C., Morin, S., Chebotko, A., Abraham, J. and Brazier, P.: Distributed semantic web data management in HBase and MySQL cluster, *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, IEEE, pp. 105–112 (2011).

[2] Guo, Y., Pan, Z. and Heflin, J.: LUBM: A benchmark for OWL knowledge base systems, *Web Semantics: Science, Services and Agents on the World Wide Web*, Vol. 3, No. 2, pp. 158–182 (2005).

[3] Harris, S., Lamb, N. and Shadbolt, N.: 4store: The design and implementation of a clustered RDF store, *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, pp. 94–109 (2009).

[4] Typesafe: akka, http://akka.io.

[5] W3C: RDF 1.1, http://www.w3.org/TR/rdf11-new.

[6] W3C: RDF 1.1 N-Triples, http://www.w3.org/TR/n-triples.

[7] W3C: SPARQL 1.1 queryLanguage, http://www.w3.org/TR/sparql11-query.

[8] 千代英一郎，宮田康志，横井一仁，西山博泰：値域分割にもとづく SPARQL クエリ分散処理方法，コンピュータソフトウェア， Vol. 30, No. 3, pp. 180–186 (2013).