要約情報を用いた SPARQL クエリ分散処理におけるデータの分割手法の検討

金子 舟[1,a)]　高野 保真[2,b)]　千代 英一郎[1,c)]

概要：大規模な Resource Description Framework（RDF）データに対して，複数のサーバを用いた分散検索手法が広く研究されている．これまで我々も，RDF データの要約情報を利用した分散検索手法について扱ってきた．要約情報を用いて解の範囲の絞り込み，負荷の分散を考慮することで，マスタ・ワーカ型の分散処理においてクエリ処理効率が向上することを既に確認している．しかし，各ワーカの持つ RDF データが同一であることを前提としていたため，大規模な RDF データを扱う際には，改善の余地があった．そこで本研究では，要約情報を利用する分散処理において，RDF データを各ワーカに分割して保持する手法を提案する．マスタがクエリの解析結果と要約情報を基に処理を割り振ることで，一部の情報しか持たないワーカ同士を組み合わせて検索が可能となる．本手法を，動画サイトの実データに対する計算処理に適用することで評価を行った．

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，Resource Description Framework（RDF）が W3C により規格化されている [4]．RDF は，主語，述語，目的語の 3 つ組（トリプル）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．各情報は Uniform Resource Identifer（URI）と呼ばれる識別子や，数値・文字列等からなるリテラルによって表現される．RDF データを RDF ストアと呼ばれるデータベースに格納し，SPARQL[5] と呼ばれる標準の問い合わせ言語を用いてクエリにマッチする情報を引き出すことができる．

すべての情報がトリプルで表現されるため，リレーショナルデータベースと比べて柔軟性が高く，様々なデータが RDF によって表現されるようになっている．たとえば，インターネット百科事典 Wikipedia の RDF 版である DBpedia（英語版で約 88 億トリプル）や，タンパク質のアミノ酸配列知識データベース Uniprot （約 300 億トリプル）が代表的な例である．これらのデータのように，RDF グラフが大規模化しているため，SPARQL クエリの効率的な処理について，多くの研究がなされている [1][2]．

これまで我々も，効率的な RDF クエリ処理を目指して，要約情報を用いて解の範囲を絞り込む RDF データの要約情報を利用した分散検索手法に関して研究を進めてきた [6][8]．この手法では，

[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)] yasunao-takano@st.seikei.ac.jp
[c)] chishiro@st.seikei.ac.jp

クエリ処理の結果となる変数の取り得る値に着目し，事前に RDF データの要約情報を作成しておく．要約情報を用いると，解の範囲の絞り込みや負荷の分散を考慮して，クエリを再構成することができ，検索効率を向上することができる．一方で，分散処理に用いる各ワーカには同じデータを配置しておく必要があることから，大規模なデータを扱う場合には各ワーカに十分な性能があることを前提としていた．

そこで本研究では，要約情報を用いたクエリ分散手法をベースとし，データを物理的に分割して各ワーカに配置する手法を提案する．具体的には，予め分割しておいたデータを各ワーカに配置し，それぞれに要約情報の生成を行う．検索を行う際は，与えられたクエリをマスタ上の要約情報に対して検索した後，その結果を反映したクエリを各ワーカにおける分散処理によって検索を行う．それぞれのワーカで求めた部分的な解集合を，マスタで統合することによってデータ全体に対する解を求める．各ワーカがデータの一部しか持たなくなるため，集約計算を含む一部のクエリに対しては，クエリの変換や要約情報の構築方法が，既存の要約情報を用いた手法とは異なる．本論文では，データの分割方法と，重複除去に関して考察を行い，その手法に基づいた実装を用いた実験により，データセットを分割して配置した場合のクエリの分散処理時間を評価した．その結果，データの複製配置，要約情報を併用した場合と比較して最大で 4.08 倍程度の性能向上がみられ，ディスク上におけるデータセット容量は約 1/3 程度に削減できた．

なお，データを分割することによってワーカ間に分かれてしまうようなトリプルが発生し得るが，このようなトリプルパターンを対象とするクエリに関して本論文では取り扱わない．

2. 要約情報を用いたクエリ分散処理

本節では，先行研究 [6][8] のクエリ分散処理について説明する．この手法は，前処理として要約情報を作成し，要約情報への検索処理の結果に基づいて，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．以降，要約情報の生成と，それを用いた検索についてそれぞれ述べる．

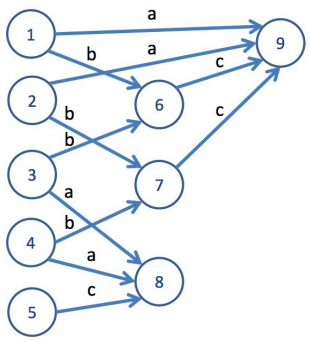


図 1　対象とする RDF グラフ

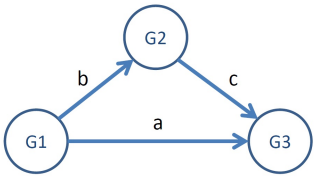


図 2　要約結果のグラフ

2.1 要約情報の生成

図 1 に示す簡単な RDF グラフを使って，要約情報の生成手法を説明する． この RDF グラフは，説明のため，ノードに番号 1 ～ 9 が振られており，各ノードからは述語にあたる a，b，c の 3 種類のエッジが張られていることとする．また，本来 RDF であれば，リテラルとリソースは分けられて表現されるが，ここでは同一のものとして扱う．

論文 [6] の要約アルゴリズムでは「ノードに入るエッジ」，「ノードから出ていくエッジ」の双方を基準として要約情報の生成を行っているが，決まった要約基準を設けているわけではないため，要約基準を自由に定めることができる．この例においては「ノードから出ていくエッジ」を基準に要約情報を生成することで検索に十分有効なグラフを作成することが出来る．このときの要約情報は，図 2 のようになる．つまり，ここでの要約情報は，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．

まず，図 2 中の要約結果のグループに含まれる元の RDF のノード番号との対応はそれぞれ以下のとおりになる．

G1 = {1, 2, 3, 4}

G2 = {5, 6, 7}

G3 = {8, 9}

それぞれ，G1 は述語 a と b の始点となるノードの集合，G2 は述語 c の始点となるノードの集合，G3 は述語の始点とならないノードの集合である．

要約結果を求めたら，それぞれのグループと，それに含まれる元の RDF データのノード間に，エッジを新たに張る．たとえば，G1 から 1 〜 4 のそれぞれのノードに要約を示す述語のエッジ（ここでは abstract とする）を張ることで，要約結果の情報を保持する．

2.2 要約情報の分割生成

要約情報の生成処理に関して，ノードを述語に基づいてグループ分けする必要があるため，時間・メモリ効率に関するコストが高い処理である．そこで，対象の RDF グラフを分割し，部分的なグラフへの要約情報を分散処理によって生成することで，分割前の RDF グラフ対する近似的な要約情報を生成する手法を提案した [8].

この手法では，任意の数に分割したグラフのそれぞれに対し，要約情報の生成アルゴリズムを適用する．そこで得られた複数の要約情報を統合し，再度要約情報の生成を行うことで，グラフ全体に対する要約情報を得るというものである．

分割によって，グラフサイズを小さくすることが出来き，要約情報の生成時に課題となっていた時間・メモリ効率について改善を図っている．評価実験では最大 64 分割まで細分化して要約情報の生成を行い，生成時間が約 1/10 程度に短縮されることを確認した．

2.3 要約情報を用いた検索処理

前節の方法で得られた要約情報を使って，以下の手順で検索処理を行う．

(1) 元のクエリを用いて要約情報に対する検索を行い，範囲を絞り込む

(2) その結果を基に，各ワーカサーバで分散検索する

(3) 各ワーカサーバで得られた結果を統合する

図 1 の RDF データに対する以下のようなクエリを考える．

```
select ?y
where {
  ?x b ?y.
  ?y c ?z.
}
```

ここで ?x，?y，?z はクエリ変数を示しており，where 節内の条件にマッチする ?y に当たるノードを検索する．つまり，上のクエリは，述語 b の終点となり，かつ c の始点となるノードを求めるためのクエリである．

手順（1) で，上のクエリを図 2 の要約結果のグラフに対するクエリとして，予備的な検索を行う．図 2 のグラフ中で，b の終点かつ c の始点となるのは G2 であるため，?y として，G2 が得られる．要約結果のグラフは，元のグラフより規模が小さくなっているため，この検索は元グラフでクエリを処理した場合よりも容易であると想定している．

この結果をもとに，以下のようなクエリを生成し，手順（2）で元の RDF データへのクエリとする．以降，このような要約情報の結果を受けて新たに生成したクエリのことを部分クエリと呼ぶ．

```
select ?y
where {
  G2 abstract ?y.
  ?x b ?y.
  ?y c ?z.
}
```

また，要約情報の生成時に追加した abstract のエッジを使ったトリプルパターンを限定節と呼び，これを追加することで，元の RDF データから直接検索をするのに比べて，検索範囲を絞り込むことができる．G2 に含まれるのは 5, 6, 7 のノードであると要約生成時に分かっているため，検索

対象のノードが平準化されるように，ワーカサーバ毎に検索対象の範囲を与えて分散処理する．先行研究では，すべてのワーカサーバが同一の RDF データを持っていると仮定している．

手順（3）で，それらの結果を統合することで，元のクエリの結果を得ることが出来る．前節で述べたように，G2 に含まれるノードは，5，6，7 であり，それらのノードに範囲が絞り込まれた検索により，クエリの結果である

$$?y = \{6, 7\}$$

が得られる．5 は G2 に含まれるが，b の終点とはならないので，結果には含まれない．

以上のように，要約情報を利用することで，規模が大きいと想定する元の RDF に対する検索時には，範囲を絞ってクエリを処理することができる．さらには，要約情報を用いることで，手順（2）で分散処理の各サーバの仕事量の平準化を図ることが可能となる．

2.4 先行研究の検索処理の特徴

上に示したように先行研究では，元の RDF データに要約情報と，それを検索で利用するための限定節を付加して利用する．各ワーカが同じようにデータの全体を保持していることを前提とすることで，処理の分散の面では要約情報を用いて各ワーカの処理の均等化を図ることができていた．

一方で，大規模なデータを扱う場合にはメモリなどの面でワーカの性能が必要となっていた．一般に RDF ストアに対する処理性能は，ディスクキャッシュなどメモリ容量と RDF データのトリプル数に依るところが大きく，必要のないデータをワーカに持たないことが望ましい．

3. 提案手法

前節で示した既存研究をベースとして，本研究では大規模 RDF データを分割し，ワーカへ分散配置した場合の要約情報を用いた SPARQL クエリ分散処理手法を提案する．各ワーカの持つデータに合わせて，各ワーカの要約情報に自身が持つデータか識別するための分割情報を加える．データを分割するため，集約計算を含むクエリにおいては，クエリの変換が必要となる．特に count と distinct が併用されている場合を対象として，分散処理によって集約計算への対応を行う．集約計算は，述語以外，つまり URI やリテラルを対象とし，この対象は要約情報の生成前に決定しているものとして考える．また，データの分割によってワーカ間に分かれてしまうようなトリプルが発生し得るが，このようなトリプルパターンを対象とするクエリに関して，本論文では取り扱わない．本節では，データの分割方法ついて述べた後，集約計算への対応方法について説明する．

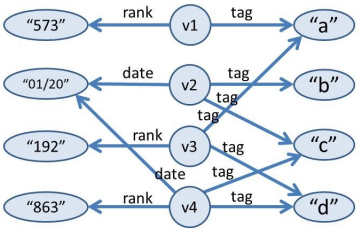


図 3 動画に関する RDF グラフ

3.1 データの分割

分割基準について，データの分割には様々な方法が考えられるが，本論文では基本的に最も出現数の多い述語を基準として，その述語の目的語にあたるノードの数が，各ワーカに均一に割り振られるよう事前にデータの分割する．たとえば，図 3 のような動画に関する簡単な RDF データを 2 つのワーカに分割することを考える．ここで，v1 から v4 が動画を示すノードで，各動画に対する tag や date などの述語でその動画のメタ情報が与えられている．含まれる述語は rank，date，tag の 3 種類であり，このなかで最も多い述語は tag である．そこで，述語 tag を基準に分割を行うと，図 4 のようになる．図 4 中のノード "a" や "c" のように，各ワーカがもつデータの間には内容の重複が発生し得る．このことを考慮して 3.2 節の手法で対応するため，分割時に重複の有無を確認しない． なお，RDF データは，単なるトリプルパターンの並びで表現されているため，トリプルを分割すればデータの複製などを考慮することなく

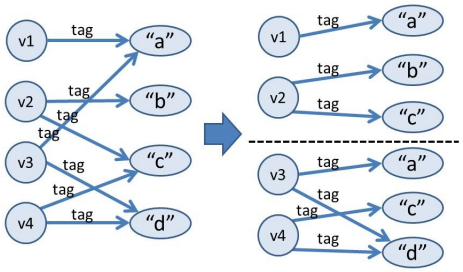


図 4　述語 tag を基準としたグラフの分割

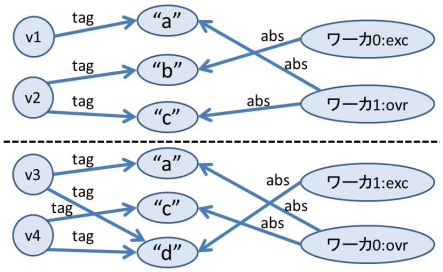


図 5　分割情報の追加

データを分割できる．最も出現数の多い述語を分割基準とするメリットとしては，分割後の各データのサイズをなるべく均一にすることが出来るという点が挙げられる．分割基準以外の述語については，全ワーカが同じものを共通して持つことになる．

分割したデータは各ワーカに配置し，要約情報の生成を行ってから，データストアへロードを行う．このときの要約情報の生成については，各ワーカを活用した分散処理とすることが出来るため，ワーカ台数分の分割数に留まるものの，実質的に 2.2 節で紹介した手法と同様の生成時間短縮効果を得ることが出来る．

しかしながら，マスタ側に配置する要約情報に関しては，本論文では分割前のグラフから直接生成した要約情報を利用している．これは，後述する各ワーカへのクエリ変換を行うことを考慮したためである．

3.2　集約計算への対応

集約計算を組み合わせたようなクエリが与えられた際，部分クエリから元クエリの結果が得られない場合が存在する．例えば次のようなクエリが与えられた場合について考える．

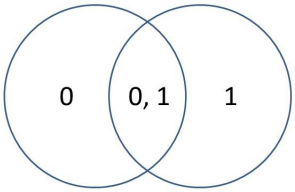
```
select (count (distinct ?o) as ?count)
where {
  ?s p ?o.
}
```

このクエリは，述語 p の終点となるノード ?o について，distinct 修飾子が用いられているため，?o の重複を除いた上でその数を求めるものである．これを複数台のワーカで分散検索を行うと，各ワーカからの結果を単純に統合しても重複を含んでしまうため，元クエリの要求する distinct に従った結果を求めることが出来るとは限らない．それは，あらかじめ作成した要約情報が，クエリの結果を重複除去できるようにワーカに仕事を分散しているわけではないためである．たとえば図 4 であれば "a" と "c" が重複してしまうため，正しい答えの 4 ではなく，6 が集約結果となってしまう．同様の問題は，平均化関数 avg 等でも生じる．

そこで，データの分割によって生じた各ワーカ間の重複に関する情報として，以下の 2 つの情報を要約情報に追加する．

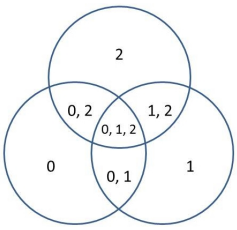
- 全データ中で自ワーカにのみ存在するデータの情報
- 自ワーカ上のデータのうち，他ワーカ上にも存在するデータと，そのワーカの識別番号情報

ここで識別番号とは，ワーカに振られている通し番号のことである．これらの情報は要約情報とともにトリプルパターンを追加することにより実現する．また，この情報は予め設定した集約計算対象のノードに対して行う．例として，図 4 で分割した RDF グラフに，分割情報を加えた様子を図 5 に示す．図中の exc は，「全データ中で自ワーカにのみ存在するデータの情報」，ovr は「自ワーカ上のデータのうち，他ワーカ上にも存在するデータと，そのワーカの番号」を表している．なお，以下に示すような count と distinct の組み合わせが 2 つ以上含まれるようなクエリに対しても，その集約計算対象（クエリ中の ?x, ?y）が事前に決定されているものであれば，集計処理を行うことが可能である．しかし，説明が煩雑になるため本



ワーカ 0 = {(0)), (0, 1)}
ワーカ 1 = {(1)}

図 6 2 分割時のデータの重複とワーカへの割り振り方針



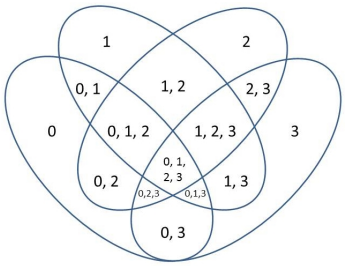
ワーカ 0 = {(0), (0, 1), (0, 2), (0, 1, 2)}
ワーカ 1 = {(1), (1, 2)}
ワーカ 2 = {(2)}

図 7 3 分割時のデータの重複とワーカへの割り振り方針

論文では扱わない．

```
select
  (count (distinct ?x) as ?count_x)
  (count (distinct ?y) as ?count_y)
where {
  ?x p ?y.
}
```

ワーカ間の重複の発生の仕方は分割数によって異なる．例えばデータを 2 分割した場合に起こり得るデータの重複とワーカへの割り振り方針は図 6 のように表される．同様に，3 分割した場合を図 7，4 分割した場合を図 8 に示す．図中の各領域に示した数字は，そのデータを持つワーカの識別番号を表している． 重複なく各ワーカのデータを集計するために，図の各領域に対し 1 台のワーカが担当するように割り振る．そこで識別番号の小さい方から順に，データ内に存在する自ワーカの番号のついた領域のうち，まだ他ワーカが担当していない領域を検索範囲とすることとして割り振る．



ワーカ 0 = {(0), (0, 1), (0, 2),
(0, 3), (0, 1, 2), (0, 1, 3),
(0, 2, 3), (0, 1, 2, 3)}
ワーカ 1 = {(1), (1, 2), (1, 3), (1, 2, 3)}
ワーカ 2 = {(2), (2, 3)}
ワーカ 3 = {(3)}

図 8 4 分割時のデータの重複とワーカへの割り振り方針

このようにすると識別番号が小さいワーカが，多くの領域を対象とするように見えてしまうが，以下に示すように，SPARQL のクエリでは，解の候補から特定の条件に当てはまるものを除外する処理のほうが比較的時間がかかるため，各ワーカの処理は平準化される．

これらの領域の分担は，要約情報に含ませた分割情報をクエリに反映させることによって行う．マスタに対し以下のような元クエリが与えられた場合の，4 分割時における変換後クエリを示す．なお，要約情報への予備検索結果に基づいて追加される限定節については，ここでは省略する．

元クエリ

```
select (count (distinct ?x) as ?count)
where {
  ?x p ?y.
}
```

変換後クエリ

```
#query0
select (count(?x)as ?count)
where {
  ?x p ?y.
}
#query1
```

```
select (count(?x)as ?count)
where {
  ?x p ?y.
  filter not exists {
    worker0:ovr abs ?x.
  }
}
#query2
select (count(?x)as ?count)
where {
  ?x p ?y.
  filter not exists {
    worker0:ovr abs ?x.
  }
  filter not exists {
    worker1:ovr abs ?x.
  }
}
#query3
select (count(?x)as ?count)
where {
  ?x p ?y.
  worker3:exc abs ?x.
}
```

各クエリへの共通した処理として，元クエリの select 句に含まれる distinct が不要となることから，これを削除する．また，目的とするトリプルパターンと変数は各クエリ間で共通するため，where 句には元クエリのトリプルパターンを踏襲する．query0 では，重複に関係なくワーカ内の当該変数に当てはまる解の個数を集計すればよいことから，共通処理のみを施したクエリが変換後クエリとなる．query1 と query2 に関しては，重複部分のうち他のワーカが集計する領域を除外する必要があるため，where 句内にフィルター処理を表す filter not exists と，ovr を主語とする他ワーカの重複を表すトリプルを追加している．query3 では，自ワーカだけが持つデータが解の領域となるため，where 句に exc を主語とするトリプルを追加することで，解の範囲の絞り込みを行っている．

4. 評価実験

データ分割時におけるクエリ分散処理性能の測定するために，ニコニコデータセット [7] を使って検索結果が求まるまでの時間を測定した．ニコニコデータセットは，(株) ドワンゴおよび (有) 未来検索ブラジルから国立情報学研究所が提供を受けて研究者に公開しているデータセットである．データセット自体には，2007 年 3 月 6 日から 2016 年 8 月 31 日までの動画情報（投稿日，タイトル，タグ，コメントなど）の約 1,400 万動画分のデータが含まれているが，今回はこれらの動画情報のうち，約 9,900 動画を抽出したものと，約 67,300 動画を抽出したものの 2 種類を RDF 化して用いた．前者で約 6 千万トリプル，後者で約 4 億 4 千万トリプルの RDF データであり，便宜上それぞれをデータ A，データ B と呼ぶこととする．

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 のマシンをマスタ，CPU: Intel Core i7 930 2.8GHz，OS: CentOS 7.3.1611，Memory: 12GB のマシン 4 台をワーカとするマスタ/ワーカ環境である．

ワーカ数が 4 であることから，提案手法の評価時におけるデータ分割数も 4 とした．分割基準はデータ中で最も多く含まれている述語がコメントに関するものであることから，これが均一となるよう分割を行っているが，データ構造を加味し，動画単位での分割となるよう調整している．ディスク上のデータサイズはデータ A で約 8.59GB，データ B で約 55.58GB であったが，分割後はそれぞれ平均で約 2.85GB, 20.17GB と，およそ 1/3 程度に縮小した．

以上の条件において，次に示す 2 つのクエリを実行した．

- コメントに特定の文字列 ” w” を含む動画数
- コメントの種類数

1 つ目のクエリはクエリの select 句に distinct を含まないことから，元クエリへ要約情報を付加

表 1 コメントに特定文字列を含む動画数 (データ A)

	実行時間
分割配置	23.75 秒
複製配置	24.59 秒

表 2 コメントに特定文字列を含む動画数 (データ B)

	実行時間
分割配置	159.04 秒
複製配置	168.04 秒

表 3 コメントの種類数 (データ A)

	実行時間
分割配置	19.22 秒
複製配置	12.87 秒

表 4 コメントの種類数 (データ B)

	実行時間
分割配置	103.20 秒
複製配置	463.27 秒

して各ワーカでの検索を行う．特定の文字列を含むかどうかの判定については，正規表現パターンによるマッチを行う `regex` をフィルターに用いた．2 つ目のクエリは種類数を求めるために重複を除去する必要があることから，クエリの `select` 句に `distinct` が含まれる．この場合については 3.2 節に示したクエリ変換を行って各ワーカへのクエリを生成した．実行時間はワーカへクエリが送られてから，マスタが結果を統合し終わるまでの時間を測定した．クエリの変換自体は非常に短時間で完了するため，実行時間の測定対象とはしていない．

大きさの異なるデータ A, B のそれぞれに対し，上述の 2 つのクエリを実行したときの実行時間を，表 1 〜 表 4 に示す．値は 3 回実行したときの平均値としている．

実行結果についてを各クエリ毎にみていくと，特定の文字列を含む動画数を検索するクエリの実行結果である表 1，表 2 について，分割配置を行うことで最大 1.06 倍と僅かではあるものの，実行時間の短縮がみられた．各ワーカの持つデータセット内における検索範囲は複製配置の場合とほぼ同等でありながら，複製配置の方が絞り込みの効果を得やすいためこのような結果となったものと考えられる．コメントの種類数を検索するクエリについては，表 3 に示すようにデータサイズが小さい場合では複製配置の実行時間の方が短時間で解を求めることが出来る．一方で，データサイズを約 6 倍とした表 4 では，分割配置によって最大で 4.08 倍の高速化効果を得ることが出来た．データ内のコメントすべてをカウント処理する `query0` を含め，各クエリ毎の実行時間も複製配置した場合より短縮されていることから，ワーカのデータサイズが分割により小さくなったことが要因であると考えられる．しかしながら，ワーカ毎の実行時間について比較すると，クエリにフィルター処理を 2 つ含む `query2` の実行時間に律速されているため，今後の課題として各ワーカの処理量を更に平準化させる必要がある．

5. おわりに

本研究では，要約情報を用いた大規模 RDF の分散検索手法について，各ワーカにデータを分割配置することにより，検索処理の高速化と各ワーカ内に占めるデータセットの容量の削減を行った．また，重複除去を必要とするクエリに対しては，あらかじめ作成した分割情報を活用することでクエリ変換を行い，分散検索によって求める手法の提案を行った．データを分割配置する際の副次的な効果として，要約情報生成の分散処理が可能になる点や，データストアへのロード時間の短縮等が挙げられる．一方，今後の課題は以下のような点である．

- データ分割基準の最適化
- ワーカ間を跨ぐトリプルの検索への対応
- ネットワーク型に近いグラフ構造を持つデータでの実験
- 各ワーカの処理量の更なる平準化

実験で用いたニコニコデータにおいては，タグとコメントの共起情報を効率的に求めることができれば，既存の研究 [3] への応用が見込める．さらに，RDF の利点を生かし，複数の RDF データを統合した場合の，要約情報の利用やデータ分割方法等についても今後検討していく．

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] 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).

[3] Sakaji, H., Kohana, M., Kobayashi, A. and Sakai, H.: Estimation of Tags via Comments on Nico Nico Douga, *Network-Based Information Systems (NBiS), 2016 19th International Conference on*, IEEE, pp. 550–553 (2016).

[4] W3C: RDF Primer, http://www.w3.org/TR/rdf-primer.

[5] W3C: SPARQL 1.1 queryLanguage, https://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[6] 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).

[7] (株) ドワンゴ，(有) 未来検索ブラジル，国立情報学研究所：ニコニコデータセット，https://www.nii.ac.jp/dsc/idr/nico/nico.html.

[8] 金子　舟，高野保真，千代英一郎：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，第 58 回プログラミング・シンポジウム (2017).