LINQ によるビューを用いた LOD に対する分散問合せ

熊本　和正† 　天笠　俊之†† 　北川　博之††

† 筑波大学システム情報工学研究科コンピュータサイエンス専攻　〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学システム情報工学域　〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †kumamoto@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

あらまし　コンピュータによる処理を目的としたデータを Web 上で公開，共有，および利用するための方法に Linked Open Data (LOD) がある．様々な種類のデータが LOD により公開されており， LOD を利用するアプリケーションの開発が強く期待されている．しかし複雑なグラフ構造である LOD への問合せは容易ではなく， SPARQL という問合せ言語の習得が必要になる．我々の先行研究では，これらの問題を解決するために，ビューを導入し，LINQ から LOD に対する問合せを容易にした．本研究では，先行研究を拡張し， LINQ から複数エンドポイントに対する分散問合せを実現した．本稿では更に，コストベースの問合せ最適化について検討，実験する．

キーワード　Linked Open Data, LOD, RDF, SPARQL, Federated Query, 分散問合せ, LINQ

1. はじめに

LOD [1] (Linked Open Data) はコンピュータ処理に適したデータを，ウェブ上で公開・共有するための方法である．LOD は データをオープン (誰でも利用，再配布が可能 [2]) にし，さらに異なる領域のデータを互いにリンクさせることで，データの相互運用性，再利用性を高める．国内外で LOD を公開する動きが近年高まっており，2013 年 6 月に行われた G8 首脳会議では，政府のデータを機械判読可能で，オープンなライセンスで公開することなどを原則とする「オープンデータ憲章 [3]」に，参加国首脳が合意した．

既に多くの LOD が公開されている．Data.gov (注1) や Data.uk.gov は国勢統計，地理空間情報などの情報を LOD として公開している．さらに，行政に限らず，様々な団体がデータを LOD として公開している．DBpedia (注2) は Wikipedia (注3) を元に作られた LOD であり，幅広いな領域の知識をカバーしている．世界中の地理情報に関するデータを公開する GeoNames (注4) や，オープンな音楽データを公開する MusicBrainz(注5) など様々な LOD が DBpedia にリンクしている．また，LOD の増加に伴い，問合せ用のインターフェイスとして，数多くの SPARQL エンドポイント (注6) も公開されており，これらを連携させて問合せることが，ますます重要になっている．

これに対して，本研究では，井上ら [4] が提案した，LINQ によるビューを用いた LOD に対する問合せ手法を拡張し，複数 SPARQL エンドポイントに対する分散問合せを可能にした [5]．井上らの手法は，公開対象の LOD をよく知る技術者が，それをどのような形式で公開したいかを JSON ビューの形式で記述する．このようにして提供された JSON ビューに対して，ユーザは LINQ 問合せ言語を利用して問合せを記述することで，LINQ 使用者は LOD に対する専門的な知識を必要とせずに LOD への問合せを行うことができる．これにより，LINQ を知っている開発者は学習のコストを必要とせず，LINQ を知らない開発者も，LINQ でなら SQL ライクな構文で簡単に問合せ処理を書くことが可能であるため，低い学習コストで問合せを実現できる．本研究は，このような LOD への問合せを容易にするシステムにおいて，さらに複数エンドポイントへの分散問合せを実行できるようにするものである．

一方，SPARQL エンドポイントのサーバの処理性能やレスポンスは，場所，利用状況や通信環境などによってばらつきがあり，処理時間が大きく変化する可能性がある．そのため，コストを考慮したクエリプランの作成は特に重要である．

そこで本稿では，JSON ビューを利用した複数エンドポイントへの分散問合せを行うにあたり，問合せ最適化のためのコストモデルを提案するとともに，クエリプランを作成する最適化アルゴリズムを提案する．本アルゴリズムでは，データサイズや，データの選択率などを考慮する従来の問合せ最適化手法に加え，ネットワークコストやビューを扱うためのコストをコストモデルに組み込むことで，クエリの実行プランをさらに最適化する．たとえば，レスポンスが非常に遅いエンドポイントの実行結果の結合処理を後に回し，実行結果を待つ間に他のビューの結合処理を行うことによって，クエリ全体の実行時間の短縮できる．

本稿の構成は以下の通りである，まず 2 節では，本論文に必要な基本的事項について解説する．3 節では，LINQ によるビューを用いた分散問合せシステムについて述べる．4 節では 3 節のシステムで利用する，コストモデルを用いた問合せ最適化手法について述べる．5 節では提案手法についての実験の予定について述べる．6 節では，関連研究について述べる．最後に 7 節で，まとめと今後の展望について述べる．

(注1)：Data.gov,http://www.data.gov/
(注2)：DBpedia, http://wiki.dbpedia.org/
(注3)：Wikipedia, http://wikipedia.org/
(注4)：GeoNames,http://www.geonames.org/
(注5)：MusicBrainz, https://musicbrainz.org/
(注6)：SPARQL Endpoints Status, http://sparqles.ai.wu.ac.at/

```
Prof:Prof1  Lab:ID    Lab:Lab1 .
Prof:Prof1  Prof:Name "K" .
Lab:Lab1  Lab:Name  "KDE" .
```

リスト 1　RDF で記述された「教員」,「研究室」に関するデータセット (prefix は省略)

```
SELECT ?ProfName ?LabName
WHERE {  ?ID Prof:Name ?ProfName .
  ?ID Prof:ID ?LabID .
  ?LabID Lab:Name ?LabName .
  FILTER (str(?ProfName) = "K") }
```

リスト 2　「教員」,「研究室」に関するデータセットから「教員の名前と所属する研究室の名前」を問合せる SPARQL クエリ

2. 前提知識

2.1 RDF

RDF (Resource Description Framework) [6] はリソース自身を表す属性と，リソース間の関係を記述するフレームワークであり，W3C で標準化されている．

RDF では， URI (Universal Resource Identifier) によって識別されるものをすべてリソースとして扱う．リソースには文書，画像，人や場所，リソース間の関係も含まれる．RDF では，リソースに対するメタデータを，トリプル（主語 (Subject)，述語 (Predicate)，目的語 (Object)） によって記述する．

主語はメタデータを記述する対象のリソース，述語は主語に関する情報のプロパティを定義し，目的語は述語の対象となる値を格納する．主語と述語は URI で，目的語は URI またはリテラル（文字列，数値...）で表現する．リスト 1 は RDF で記述された「教員」,「研究室」に関するデータセットの例である．

2.2 SPARQL

RDF に対する問合せ言語として， SPARQL (SPARQL Protocol and RDF Query Language) [7] がある， SPARQL は，W3C によって標準化されており，近年広く用いられている．

SPARQL ではグラフパタンを記述することで，記述したグラフパタンにマッチする部分グラフを検索することができる．WHERE 句では検索したいグラフパタンをトリプル (主語，述語，目的語の三つ組) で定義し，FILTER 句では文字列などでの絞り込み等が可能である．SELECT 句では ?x の形式で変数を宣言する．

リスト 2 は 「教員」,「研究室」に関するデータセットから「教員の名前と所属する研究室の名前」を問合せる SPARQL クエリ の例である．

2.3 LINQ

LINQ (Language INtegrated Query) (注7) は Microsoft .NET Framework で提供されるデータ操作機能である．LINQ の利点として，以下のような点がある．(1) 簡潔で読みやすい（特に複数条件をフィルター処理する場合)，(2) 最小限のコードで強力なフィルタ処理，並び替え，グループ化などの機能を使用することができる．(3) 様々な情報源に対して統一的な構文で問合せ等の操作を行うことができるので，コードをほとんど変更することなく，他のデータソースに移植できる．LINQ は C#などの.NET 系のプログラミング言語にとどまらず，JavaScript をはじめ，さまざまなプログラミング言語向けの実装が存在する．リスト 3 は LINQ クエリの例である．

(注7)：https://msdn.microsoft.com/en-us/library/bb308959.aspx

```
var join = from prof in profs
  join lab in labs
  on prof["labID"] equals lab["ID"]
  where prof["Name"] == "K"
  select new { ProfName=prof["Name"],
  LabName=lab["Name"] };
```

リスト 3　「教員」,「研究室」に関するデータセットから「教員の名前と所属する研究室の名前」を問合せる LINQ クエリ

Algorithm 1: システムの処理全体の流れ

Input : Query: LINQ クエリ
Output: JSON: クエリ結果

begin
- 処理木 $QT \leftarrow Query$ をパースして処理木を構成する
- ビュークエリ $V \leftarrow QT$ に含まれるビュークエリ集合をビュー定義から抽出
- $GenerateSPARQLQueries(V)$ (Algorithm 2)
- $BestPlan \leftarrow FindBestPlan(V)$ (次節，Algorithm3)
- $QT' \leftarrow BestPlan$ を使って QT を書換え
- $JSON \leftarrow QT$ を実行
- return $JSON$

end

3. LINQ によるビューを用いた LOD に対する分散問合せシステム

本節では，LINQ によるビューを用いた LOD に対する分散問合せシステムについて述べる．このシステムは井上らによって提案された問合せシステム [4] をベースにしており，主に以下の 4 点が拡張されている．(1) 複数のビューを利用可能．(2) 複数 SPARQL エンドポイントへの問合せが可能．(3) 結合演算（JOIN 句）のサポート．(4) コストモデルによる問合せ最適化．コストモデルによる最適化については次節で詳しく説明する．

3.1 処理全体の流れ

システムの概要を図 1 に示す．本システムを導入することで，データ公開者など公開対象の LOD をよく知る技術者が，RDF データに対して JSON 形式のビューを提供することができる．一方アプリケーション作成者は，SPARQL エンドポイントに対して記述された JSON ビューに対して，RDF の構造

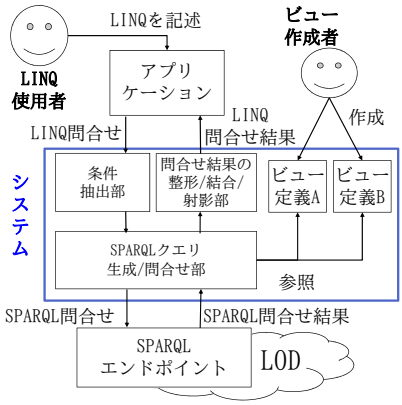

図 1　システムの概要図

```
{  "viewname" : "Profs",
  "sparql": ‘‘SELECT ?ID ?Name ?LabID
   WHERE { ?ID Prof:Name ?Name .
   ?ID Lab:ID ?LabID’’
  "jsonschema": {
    "type":"object",
      "properties":{
        "results": {
        ?ID:  { "type":"string" },
        ?Name:{ "type":"string" },
        ?LabID:{ "type":"string" }
  }}},
  "endpoint":
    "http://MyCentOSSrv:8890/sparql"}}
```

リスト 4 「教員」データセットに対するビュー定義例

を意識することなく LINQ による問合せ記述を行うことが可能となる．システムは以下のことを行う．(1) LINQ クエリから抽出する問合せ条件を抽出し，ビュークエリから SPARQL クエリの生成を行う．(2) 最適なクエリプランを生成し，処理木を書換える．(3) SPARQL 問合せを発行し，処理木を実行する．SPARQL 問合せ結果を処理木のリーフノードとして処理し，LINQ 問合せ結果として返す．

全体の処理の流れをまとめたものを，Algorithm 1 に示す．各ビューの問合せに使用するクエリのことを，ビュークエリと呼称している．本節では，まず，システムが参照するビュー定義について説明する．その後，システムの処理について順に説明する．

3.2　ビュー定義

本システムでは，RDF データを JSON に見せる手段としてビューを利用する．ビュー定義には，グラフ構造から必要なデータが何かを SPARQL で記述した SPARQL テンプレート（ビュークエリ），問合せ対象のデータの構造を JSON スキーマで記述した JSON ビューをそれぞれ定義する．SPARQL エンドポイントのアドレスなどに関しても，ここに併せて記述する．リスト 4 は，ビュー定義の例である．

Algorithm 2: GenerateSPARQLQueries

Input : QT: 処理木
Output: V: 生成された SPARQL クエリ集合

```
1  begin
2      foreach QT の射影条件 do
3          対応する V_i の Select 句に 射影条件を追加
4      end
5      foreach QT の結合条件 (V_i.c_k == V_j.c_l) do
6          if V_i に c_k が含まれていない場合 then
7              対応する V_i の Select 句に c_k を追加
8          end
9          if V_j に c_l が含まれていない場合 then
10             対応する V_j の Select 句に c_l を追加
11         end
12     end
13     foreach QT の選択条件 do
14         対応する V_i の Where 句に 選択条件を追加
15     end
16     return V
17 end
```

3.3　システムの処理

3.3.1　問合せ条件の抽出および SPARQL クエリの生成

まず，図 1 中の条件抽出部は，問合せ条件の抽出のため，LINQ クエリから処理木を構築する．LINQ 問合せから処理木を構築する方法については本研究の対象から外れるためここでは詳しく議論しないが，LINQ プロバイダの構文解析器を利用することで処理木を得ることができるため，これを利用する．

処理木の構築が終了したら，ビュー定義に記述されている SPARQL テンプレート（ビュークエリ）を書換えることで，SPARQL クエリの生成を行う．まず，処理木に含まれるすべての射影条件を，対応するビュークエリの Select 句に加える．次に，処理木に含まれるすべての結合条件に指定されているキーを，射影条件の抽出で抽出されなかった射影条件であれば，対応するビュークエリの Select 句に加える．最後に，処理木に含まれるすべての選択条件を，FILTER キーワードとして対応するビュークエリの Where 句の末尾に書き加える．問合せ条件の抽出とクエリの書換え処理をまとめると，Algorithm 2 のようになる．

3.3.2　クエリプランの生成および処理木の書換え

ビュークエリの準備が終わったら，クエリプランの生成および処理木の書換えを行う．クエリプランの詳しい生成方法については次節で説明する．

3.3.3　処理木の実行と問合せ結果の生成

クエリプランによって書換えられた処理木を用いて，生成されたすべての SPARQL クエリを，並列でビュー定義に指定されている SPARQL エンドポイントに送信する．SPARQL エンドポイントは XML, JSON, CSV などの形式で問合せ結果を返却するので，返却されたデータは ビュー定義を元に整形する．

LINQ プロバイダはそれらのデータを処理木のリーフノードとして使用し，最終的な LINQ クエリの結果を生成する．こ

Algorithm 3: *FindBestPlan*

Input : V: ビュークエリ集合
Output: $bestplan[V]$

```
1  begin
2      統計情報 S ← 問合せ最適化に使用する統計情報を取得
3      配列 leastcost[] のすべての要素を ∞ で初期化
4      配列 bestplan[] のすべての要素を ∅ で初期化
5      配列 numrecords[] のすべての要素を 0 で初期化
6      foreach ビュークエリ V_i ∈ V do
7          sel ← V_i の選択条件から selectivity を計算
8          numrecords[{V_i}] ← sel × S_i.numrecords
9          bestplan[{V_i}] ← V_i
10         datasize ← (numrecords[{V_i}] × S_i.recordsize)
11         leastcost[{V_i}] ←
           S_i.time + β × datasize + datasize/S_i.bandwidth
12     end
13     FindBestPlanDP(V)
14     return bestplan[V]
15 end
```

のとき，リーフノードの結合相手の結果が既に問合せ終了していれば，その結合相手との結合演算を実行する．結合演算には，条件抽出部で抽出した結合条件が使用される．結合方法は，LINQ は結合条件に等結合を扱うため [8]，ハッシュ結合を採用すると高速になる．

4. コストベースの分散問合せ最適化

本節では，前節で提案した JSON ビューを用いた分散問合せシステムにおいて，最適化アルゴリズムの提案を行う．

4.1 手法のアイデア

本手法では，ネットワーク経由でアクセス可能な複数の SPARQL エンドポイントに対して，JSON ビューを経由した分散問合せを行う際の問合せ最適化について提案する．そのために，各 SPARQL エンドポイントについて，各種統計情報 (結合対象のデータサイズや結合演算のデータの選択率) に加えて，ネットワーク速度（帯域幅）を考慮したコストモデルを提案する．さらに，提案したコストモデルを最適化するために，動的計画法を用いて最適な問合せプランを探索する．

実行プランを生成する関数 *FindBestPlan* を，Algorithm 3 に示す．この関数の返り値は実行プラン $bestplan[V]$ である．*FindBestPlan* は大別して以下の 3 つのステップから成る．(1) コストの推定に使用するための配列を初期化する．(2) クエリが使用するすべてのビュークエリについて，各々のコストの推定を行う．(3) ビュークエリ集合から動的計画法を用いて結合順序を計算する．本節ではまずコストモデルについて説明してから，各ステップについて説明していく．

4.2 コストモデル

システムは，LINQ クエリの実行に必要なビューの個数，SPARQL クエリをビューの各エンドポイントに発行し，返ってきた結果をコストモデルによって最適化された結合順序で結合する．

表 1 統計情報

numrecords	ビュークエリの実行時間．この実行時間は純粋なクエリの実行時間であり，転送時間は含まない．
columnsizes	各列の平均データサイズ．コスト推定時に，射影する要素のみを足しあわせ，さらに SPARQL エンドポイントが返却するデータの形式に応じて (XML と JSON では大きさが違う) 区切り文字などを加味して計算したものを，1 レコードの大きさ *recordsize* として返却する．
valuecounts[]	各列の distinct な行数．結合演算のキーによる選択率の計算に使用する．
bandwidth	ビュークエリを実行する SPARQL エンドポイントとクライアントとの間の推定帯域幅．この値については，エンドポイントが提供するエスティメータのものではなく，ビュー作成時などにデータを事前にダウンロードしてその転送速度を測定することで取得しておく．

コストモデルでは，全体のコストを，転送時間を含むビュークエリの実行コストおよび，それらのビュークエリの結合コストとみなし，これを最小化する．Algorithm 3 の 11 行目が，ビュークエリのコストである．コスト式には，ビューの処理時間と，ネットワークの転送時間の 2 つのコストが含まれる．これらは，SPARQL エンドポイントから取得したり，事前に取得することで得られる統計情報を用いて計算する．

統計情報には，表 1 に示す情報を使用する．ビュークエリ実行結果の最大のレコード数は，ネットワーク転送時間の推定，ビューの整形時間の推定，結合時間の推定に使用する．レコードサイズは，ビューの整形時間の推定，ネットワーク転送時間の推定に使用する．valuecounts[] は，各列に重複しないレコードがいくつあるかを示し，結合時間の推定のとき，結合演算のデータの選択率を推定するため使用する．SPARQL エンドポイント上での実行時間の推定も，エンドポイントから得られる統計情報を使用する．クライアントと SPARQL エンドポイントとの推定帯域幅は，SPARQL エンドポイントのエスティメータからは得られないので，事前にクライアントから取得しておく．

4.3 配列の初期化

プランの計算のため，各ビュークエリの集合の最小コストを保存する *leastcost*[]，各ビュークエリ集合の最適な結合順序 *bestplan*[]，各ビュークエリ集合の生成する結果の行数 *numrecords*[] を使用する．ビュークエリ集合のべき集合の各要素 *Set* について，$leastcost[Set]$ を ∞，$bestplan[Set]$ を $\emptyset$，$numrecords[Set]$ を 0 として設定する．

これらの配列は，引数にビュー集合を取る．集合であること

Algorithm 4: *FindBestPlanDP*

```
Input  : V: ビュークエリ集合
1  begin
2      if leastcost[V] ≠ ∞ then
3          return;
4      end
5      foreach ビュークエリ V_i ∈ V do
6          FindBestPlanDP(V − {V_i})
7          FindBestPlanDP({V_i})
8          JoinCost ←
           numrecords[V − {V_i}], numrecords[{V_i}] から結合コ
           ストを推定
9          cost ← max(leastcost[V − {V_i}], leastcost[{V_i}]) +
           α × JoinCost
10         if cost < leastcost[V] then
11             leastcost[V] ← cost
12             bestplan[V] ← bestplan[V − {V_i}] ⋈
               bestplan[{V_i}]
13             numrecords[V] ←
               numrecords[V − {V_i}], numrecords[{V_i}] から結
               果の行数を推定
14         end
15     end
16 end
```

を示すために，ビュークエリ単体から成る集合は波括弧を付けて表現していることに注意されたい．たとえば，V はすべてのビュークエリの集合であり，V_i は V の一要素であるが，$\{V_i\}$ は，ビュークエリ V_i のみから成る集合である．

4.4 各ビュークエリのコスト推定

クエリが使用するすべてのビュークエリについて，そのビュークエリのコスト $leastcost[\{V_i\}]$ を計算する．ビュークエリのコスト $leastcost[\{V_i\}]$ の推定は，統計情報を使用して行う．各要素は，以下のように設定する．

leastcost$[\{\mathbf{V_i}\}]$ ビュークエリの純粋な実行時間である $S_i.time$ と，結果の転送コストを計算するために $S_i.numrecords$，$S_i.recordsize$，$S_i.bandwidth$ を使用し，計算する．アルゴリズム中の $\beta \times datasize$ はビューを扱うためのコストを表す．ビューを扱うコストはデータサイズに比例すると考え，パラメータ β が乗じられる．

numrecords$[\{\mathbf{V_i}\}]$ $S_i.numrecords$，データの選択率 sel を乗じて推定された行数．

bestplan$[\{\mathbf{V_i}\}]$ V_i を設定する．

$leastcost[\{V_i\}]$ は，直感的には，ビュークエリが問合せを発行したマシンに到着し，結合演算が適用可能になるまでの時間（コスト）ということもできる．

4.5 実行プランの生成

ビュークエリのコスト推定が終了したら，ビュークエリ集合 V とその統計情報 S を関数 $FindBestPlanDP$ に与えて呼び出し，動的計画法を用いて結合順序の計算を行う．関数 $FindBestPlanDP$ を Algorithm 4 に示す．この関数は再帰的に呼び出される．

まず，与えられた ビュークエリ集合 V の $leastcost[V]$ が計算済みであれば関数を終了する．6〜7 行目は $V - \{V_i\}$ と $\{V_i\}$ について それぞれ再帰的に $FindBestPlanDP$ を呼び出す．与えられたビュークエリ集合 V のコスト $cost$ は，以下の式で求めることができる．

$$max(leastcost[V - \{V_i\}], leastcost[\{V_i\}]) + \alpha \times JoinCost$$

これは，二つのビュークエリ集合のコストの大きい方と，それらの結合コストが，そのビュークエリ集合のコストになることを示している．$JoinCost$ は，$numrecords[V - \{V_i\}]$ および $numrecords[\{V_i\}]$ から推定する．推定方法は結合アルゴリズムによって異なるが，たとえばネステッドループ結合なら numrecords 同士の積，ハッシュ結合なら numrecords 同士の和になる．$JoinCost$ は実際には結合に掛かる時間を推定しなければならないが，$JoinCost$ は時間に比例した値にすぎないため，結合 1 回に掛かる時間に相当するパラメータ α が乗じられる．なお，パラメータ α が適切に設定されている場合については，ネットワークコストを考慮していない場合に比べて必ず良いプランが提案できる．

もし計算した $cost$ すでに計算した $leastcost[V]$ より小さければ，$cost$ で $leastcost[V]$ を更新し，$bestplan[V]$ は ビュークエリ集合 $V - \{V_i\}$ および $\{V_i\}$ の結合であると設定する．結果行 $numrecords[V]$ の推定方法は本研究の趣旨から外れるため詳しくは議論しないが，古典的なリレーショナルデータベースシステム同様に，$T(R)T(S)/max(V(R,a), V(S,b))$ ... $T(R)$ はリレーションの行数，$V(R,a)$ は R 内のキー a の $valuecount$) などで推定する．

$FindBestPlanDP(V)$ を実行した結果，最終的に，$bestplan[V]$ に，結合順序が保存される．

5. 評価実験

本節では，提案手法の有効性を評価するための評価実験について述べる．実験では，SPARQL の仕様に含まれる SERVICE 句を用いた分散問合せ機能と性能比較を行うことによって提案手法の有効性を示す．

5.1 実験データセット

実験データセットとして Berlin SPARQL Benchmark [9] が開発する BSBM Tools (注8) を使用した．BSBM Tools は，コマースシステムに格納されているような架空の商品，商品の特徴，製造者などについて，スケーラブルなテスト用の合成デー

表 2 データの規模

製品数	トリプル数	データサイズ
1k	371,911	34MB
10k	3,534,773	310MB
50k	17,536,178	1.6GB
100k	34,872,182	3.1GB

(注8)：BSBM Tools, http://sourceforge.net/projects/bsbmtools

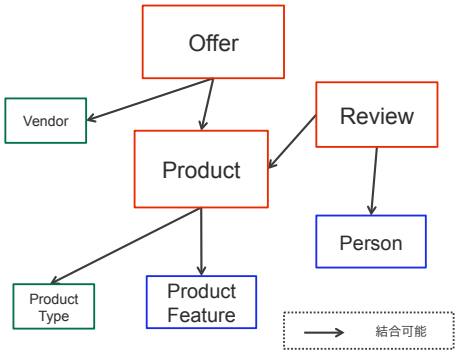

図 2　実験ビュー

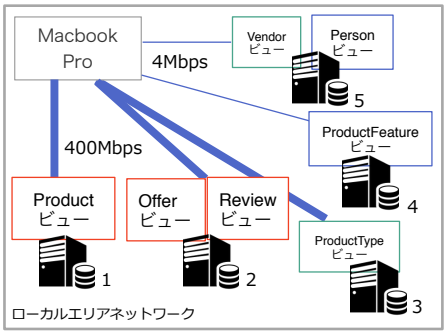

図 3　実 験 環 境

タを生成することができる．本実験では，製品数を 1,000 (1k と表記)， 10,000 (10k)， 50,000 (50k)， 100,000 (100k) にしてそれぞれ生成した．その際のデータセットのトリプル数とデータサイズを表 2 に示す．

Berlin SPARQL Benchmark データセットのクラス定義に沿う形で，ビューをそれぞれ作成した．それぞれのビューの関係を表す RDF グラフを図 2 に示す．矢印で繋がっているものに関しては，URI 同士で結合条件を記述することができる．

5.2　実 験 環 境

実験は，ローカルネットワーク上のクライアントマシン 1 台とサーバマシン 5 台で行なった．実験用システムの環境，構成を図 3 に示す．クライアントマシンの実行環境として Java (JDK 1.8) を使用した．Java VM のオプションで，コンカレント GC を有効にするとともに，コンカレント GC の並列化を有効にしている．ヒープ領域のサイズは 4GB を指定している．各サーバマシンには，RDF ストア および SPARQL エンドポイントとして，Virtuoso Opensource (7.2.0) (注9) をインストールした．各マシンの OS，CPU，RAM，通信速度を以下に示す．通信速度は，tc コマンドを用いてネットワークインターフェイスに対して通信制限を擬似的に設定している．

クライアントマシン OS: Mac OS X 10.11, CPU: Core i5-5257U 2.7GHz, RAM: 8GB

サーバマシン 1 OS: CentOS 6.7, CPU: Core i7-4820K 3.70GHz, RAM: 64GB，通信速度: 400 Mbps

サーバマシン 2 OS: Ubuntu 14.04, CPU: Core i7-2600 3.40GHz, RAM: 64GB，通信速度: 400 Mbps

(注9)：Virtuoso Opensource, http://virtuoso.openlinksw.com/

```
var querystr = ' from offer in \$0
  join product in \$1
  on offer.ofprdct equals product.prdct
  join feature in \$2
  on product.prdctft equals feature.ft
  on offer.ofvndr equals vendor.vndr
  where product.value1 < 50
  where offer.ofdays > 6
  select [product.prdct, product.prdctlbl,
    product.value1, product.ptype,
    product.pdate,  product.pd,
    feature.ft, feature.ftlbl, feature.ftcmnt,
    offer.of, offer.ofdays, offer.ofdate,
    offer.ofvndr] ';
```

リスト 5　クエリ 1 (LINQ の Javascript での実装，JSINQ によるもの)

サーバマシン 3 OS: CentOS 6.7, CPU: Opteron Processor 6344 1.40GHz, RAM: 32GB，通信速度: 約 80Mbps

サーバマシン 4 OS: Ubuntu 14.04，CPU: Core-i5 M560 2.67GHz, RAM: 2GB，通信速度: 512 Kbps

サーバマシン 5 OS: Ubuntu 14.04，CPU: Core-i5 M560 2.67GHz, RAM: 4GB，通信速度: 512 Kbps

5.3　実験クエリ

実験に使用したクエリを以下に示す．分散問合せ処理機能を利用した問合せ処理（以下，SERVICE 句とも表記）用の SPARQL クエリについても示す．SPARQL クエリにはグラフの名前が指定されているが，実際はデータサイズの実験ごとに切り替えている．

クエリ 1 Product, Feature, Offer ビューに対する問合せ．実際のクエリをリスト 5 に示す．SPARQL クエリを付録のリスト 6 に示す．

クエリ 2 Product，Feature，ProductType，Offer，Vendor ビューに対する問合せ（クエリと SPARQL クエリは省略）．

クエリ 3 Product，Feature，ProductType，Review，Person ビューに対する問合せ（クエリと SPARQL クエリは省略）．

5.4　実 験 結 果

製品数 1k，10k，50k，100k のデータに対してクエリ 1〜3 を実行したときの実行時間をそれぞれ図 4，図 5，図 6 に示す．単位は秒で，初回の実行結果を捨て，5 回測定したものの平均である．1800 秒以上結果が帰ってこないものに関しては，DNF と表記している．クエリ 2, 3 に関しては，SERVICE 句の場合は，実行時間が大きすぎて終了しなかった．提案手法ではデータの規模に関わらず処理が終了しており，データの規模にしたがって，おおよそ線形に時間が増加している．

6.　関 連 研 究

6.1　LOD に対する分散問合せ最適化

RDF データベースに対する分散問合せの研究は数多く存在する [10] [11] ．Lynden ら [10] の研究 は，時間制限がある状況

```
SELECT ?prdct ?prdctlbl ?value1 ?ptype
       ?pdate ?pd ?ft ?ftlbl ?ftcmnt
       ?of ?ofdays ?ofdate ?ofvndr
WHERE {
  graph <p1000> {
    ?of rdf:type bsbm:Offer;
    bsbm:product ?prdct;
    bsbm:vendor ?ofvndr; dc:date ?ofdate;
    bsbm:deliveryDays ?ofdays.
    FILTER(?ofdays > 5)
  SERVICE <http://endpoint1:8890/sparql> {
    graph <p1000> {
      ?prdct rdf:type bsbm:Product;
      rdfs:label ?prdctlbl;
      bsbm:productFeature ?ft;
      rdf:type ?ptype;
      bsbm:productPropertyNumeric1 ?value1;
      dc:date ?pdate; bsbm:producer ?pd.
      FILTER(?value1 < 35) }}
  SERVICE <http://endpoint4:8890/sparql> {
    graph <p1000> {
      ?ft rdf:type bsbm:ProductFeature;
      rdfs:label ?ftlbl; rdfs:comment ?ftcmnt;
      dc:publisher ?ftpublisher;
      dc:date ?ftdate . }}}}
```

リスト 6　クエリ 1 と等価な SPARQL クエリ (PREFIX は省略)

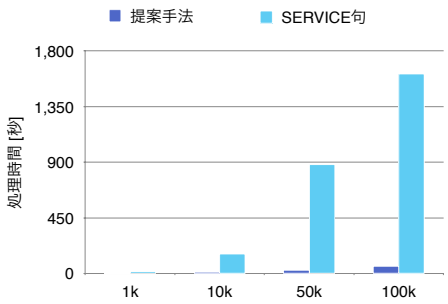

図 4　クエリ 1 の実験結果

下で，RDF データとパブリックな SPARQL エンドポイントの両方から情報を取得することで，SPARQL エンドポイントが時間内に結果を返せない状況下でも，より多くの結果を取得する研究である．クライアントマシンがプランニングを行い統合クエリの結合を行う点において本研究と類似するが，ビューを介した問合せではない点が本研究と異なる．[11] の研究は，クエリプランナとコストモデルを用いて問合せ最適化を行うが，RDF エンジンに対する最適化であり，問合せ側で最適化を行う本研究とは異なる．

6.2　リレーショナルデータベースに対する分散問合せ最適化

リレーショナルデータベースの分散問合せに関する研究も，古くから存在し，そのアルゴリズムは初期の分散問合せリレーショナルデータシステムにも組み込まれている [12] [13]. [12] は処理木のリーフノードがリレーションで，それら結合順序を推定する機構が組み込まれている点がで本研究と類似するが，対象がリレーショナル・データベースシステムである点が異なる．

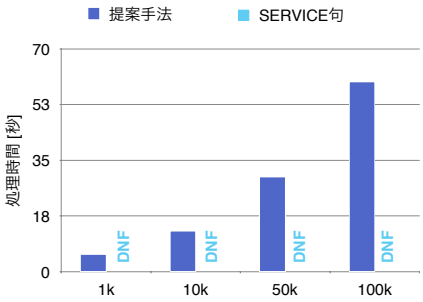

図 5　クエリ 2 の実験結果

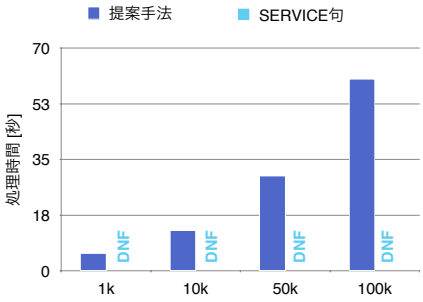

図 6　クエリ 3 の実験結果

6.3　LOD，RDF に対するビューの導入

井上らの提案し，本稿で拡張した手法は，RDF の集合に対してビューを導入しているが，RDF の集合に対するビューの導入に関する研究は LOD の概念が登場する前から行われている [14] [15] [16] [17]. [14] および [15] は，現在の仕様とは異なる古い SPARQL を対象に，前者はスキーマレベルのビュー定義を提案している．後者は RDB に格納された RDF に対して集約演算を独自に開発したものである．[17] はセンサ等から得た時系列 RDF データに高速に対して，アクセスするための一時的なビューの設計に関する研究であり，本手法が LOD を利用しようとするアプリケーションに対して，複雑性を除去して利用させる点で類似する．ただし SPARQL による問合せを想定している．

Le ら [18] や，Etcheverry ら [19] の研究は，LOD に対して SPARQL クエリを用いてビュー定義を行う点で本研究と類似するが，彼らの定義するビューは LOD への問合せを簡単化する目的の JSON ビューではなく，ビューの定義と研究の目的が異なる．

RDB に格納されたデータに対して RDF としてアクセスするためのビュー定義については数多くの研究が存在するが [20] [21]，これらの手法は仮想的な RDF 集合をビューとして作り出し，SPARQL クエリを SQL に書き換えて問合せを行う．対象がリレーショナルデータである点が，本研究とは異なる．

6.4 LINQ を用いた LOD へのアクセス

[22] は.NET 言語向けのライブラリ “LINQ to RDF(注10)” を用いて， Web 上の LOD に対して LINQ 問合せによる SPARQL 問合せを実現している．この手法では，アプリケーション開発者が LOD の内部構造を把握している必要がある．一方，本研究ではビューを定義することで LOD のリソース集合を抽象化し，アプリケーション開発者が特定のビューに対する LINQ 問合せを記述するだけで，LOD のグラフ構造に関する知識がなくても，LOD に対する問合せを行うことができる点が異なる．

7. おわりに

本研究では，LINQ によるビューを用いた LOD に対する問合せ手法の拡張として，LINQ から複数の SPARQL エンドポイントに対する分散問合せを実現し，コストベースの問合せ最適化手法を提案した．評価実験では，コストモデルを用いた分散問合せ処理が，SPARQL エンドポイントが行う SERVICE 句による分散問合せ処理に比べて，データの規模に関わらず，短い実行時間で処理が行えることを示した．

今後の課題としては，以下が挙げられる．(1) 実験クエリおよびデータセットの追加．(2) 問合せる複数のビューが同一エンドポイント内に存在する場合の対策．同じエンドポイントに複数のビューが存在するとき，ひとつの SPARQL クエリにまとめるなど．(3) 結果のキャッシュによる最適化．利用者が利用できるビューは固定されているので，ビュークエリの問合せ結果をキャッシュし，最新のデータとの差分だけを問合せることができれば，問合せ時間を短縮できるのではないかと思われる．

謝　辞

本研究の一部は，共同研究費（富士通研究所 CPE27151），文科省“ 実社会ビックデータ利活用のためのデータ統合・解析技術の研究開発 ”，および，科研費（25240014）による．

文　献

[1] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, Vol. 5, No. 3, pp. 1–22, 2009.

[2] Open definition 2.0 - open definition - defining open in open data, open content and open knowledge. `http://opendefinition.org/od/2.0/en/`.

[3] Cabinet Office, United Kingdom. Open Data Charter. June 2013. `https://www.gov.uk/government/publications/open-data-charter`.

[4] 井上寛之, 天笠俊之, 北川博之. LINQ を用いた Linked Open Data に対する問合せ. In *DEIM Forum 2014 D7-2*, 2014.

[5] Kazumasa Kumamoto, Toshiyuki Amagasa, and Hiroyuki Kitagawa. A system for querying RDF data using LINQ. In Leonard Barolli, Makoto Takizawa, Hui-Huang Hsu, Tomoya Enokido, and Fatos Xhafa, editors, *18th International Conference on Network-Based Information Systems, NBis 2015, Taipei, Taiwan, September 2-4, 2015*, pp. 452–457. IEEE Computer Society, 2015.

[6] Jeremy J. Carroll and Graham Klyne. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C recommendation, W3C, February 2004. `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`.

[7] Steven Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C Recommendation, W3C, March 2013. `http://www.w3.org/TR/sparql11-query/`.

[8] join clause (c# reference). `https://msdn.microsoft.com/library/bb311040(v=vs.110).aspx`.

[9] Christian Bizer and Andreas Schultz. The Berlin SPARQL Benchmark. *International Journal On Semantic Web and Information Systems*, 2009.

[10] Steven Lynden, Isao Kojima, Akiyoshi Matono, Akihito Nakamura, and Makoto Yui. A hybrid approach to linked data query processing with time constraints.

[11] Mohammad Hammoud, Dania Abed Rabbou, Reza Nouri, Seyed-Mehdi-Reza Beheshti, and Sherif Sakr. Dream: Distributed rdf engine with adaptive query planner and minimal communication. *Proc. VLDB Endow.*, Vol. 8, No. 6, pp. 654–665, February 2015.

[12] Guy M Lohman, C Mohan, Laura M Haas, Bruce G Lindsay, Patricia G Selinger, Paul F Wilms, and Dean Daniels. Query processing in r. *Research Report RJ*, Vol. 4272, , 1985.

[13] Philip A. Bernstein, Nathan Goodman, Eugene Wong, Christopher L. Reeve, and James B. Rothnie, Jr. Query processing in a system for distributed databases (sdd-1). *ACM Trans. Database Syst.*, Vol. 6, No. 4, pp. 602–625, December 1981.

[14] Huajun Chen, Zhaohui Wu, and Yuxin Mao. Rdf-based ontology view for relational schema mediation in semantic web. In *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems - Volume Part II*, KES'05, pp. 873–879, Berlin, Heidelberg, 2005. Springer-Verlag.

[15] Edward Hung, Yu Deng, and V. S. Subrahmanian. Rdf aggregate queries and views. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pp. 717–728, Washington, DC, USA, 2005. IEEE Computer Society.

[16] François Goasdoué, Konstantinos Karanasos, Julien Leblay, and Ioana Manolescu. View selection in semantic web databases. *CoRR*, Vol. abs/1110.6648, , 2011.

[17] Geetha Manjunath, R Badrinath, Craig Sayers, and Venugopal K. S. Temporal views over rdf data. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pp. 1131–1132, New York, NY, USA, 2008. ACM.

[18] Wangchao Le, Songyun Duan, Anastasios Kementsietsidis, Feifei Li, and Min Wang. Rewriting queries on sparql views. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pp. 655–664, New York, NY, USA, 2011. ACM.

[19] Lorena Etcheverry and Alejandro A. Vaisman. Views over RDF datasets: A state-of-the-art and open challenges. *CoRR*, Vol. abs/1211.0224, , 2012.

[20] Christian Bizer. D2rq - treating non-rdf databases as virtual rdf graphs. In *In Proceedings of the 3rd International Semantic Web Conference (ISWC2004)*, 2004.

[21] Richard Cyganiak, Souripriya Das, and Seema Sundara. R2RML: RDB to RDF Mapping Language. W3C recommendation, W3C, September 2012. `http://www.w3.org/TR/r2rml/`.

[22] Exploiting the RDF-based Linked Data Web using .NET via LINQ. `http://virtuoso.openlinksw.com/whitepapers/rdf%20linked%20data%20dotNET%20LINQ.html`.

(注10)：LINQ to RDF, https://code.google.com/p/linqtordf/