情報処理学会論文誌　データベース　Vol. 2　No. 1　33–45 (Mar. 2009)

分散RDF問合せ処理時の転送量減少のためのブルームフィルタの拡張

的野 晃整[†1]　小島 功[†1]

本稿では，分散環境におけるRDF問合せ処理の効率化を目指し，ブルームフィルタを拡張して転送量を減少させる手法を提案する．RDFはメタデータ記述のための枠組みで，近年様々な応用分野に広く利用されており，各地でボトムアップに作成・管理されている．それらの分散したRDFデータに対して，横断的・包括的な問合せを行いたいという要求が高まっている．これまで，分散RDFデータ検索に関する研究はいくつか提案されているが，それらの多くは，トップダウンに配置したRDFデータに対する処理手法やトリプルパターンマッチングのような単純な検索に関する研究が主であった．我々が提案する手法のような，ボトムアップに作成されたRDFデータに対して，結合や和集合などの演算を含む高度な問合せ処理の効率化を目指した研究は，これまでほとんど行われていない．提案手法ではRDFトリプルに対応した3次元のブルームフィルタを用い，問合せ処理時にブルームフィルタ間でビット演算を行うことで，リモートRDFデータへアクセスする前に，そのデータが解に含まれているかどうかを判断することができるため，データ転送量を削減でき，処理時間の減少につながる．我々は，RDF問合せ言語SPARQLを処理するプロトタイプシステムを作成し，分散環境におけるRDFデータ問合せ処理の効率が向上することを実験によって確認した．

Distributed RDF Query Processing Based on an Extension of Bloom Filters

Akiyoshi Matono[†1] and Isao Kojima[†1]

In this paper, we propose an indexing scheme for distributed RDF query processing using the Bloom filters. RDF is a framework for describing metadata and, today, it is widely used in various fields. Generally, RDF data are created in a bottom-up manner, that is they are created by different people and stored in different locations. Therefore, RDF query processing for the distributed RDF data becomes an important issue. So far, several distibuted search approaches for RDF data have already been proposed. These approaches, however, were specially designed for RDF data in a top-down fashion, or for simple search using triple pattern matching. To the best of our knowledge, there are no studies for efficient distributed RDF query processing that includes binary operations such as join and union. In this paper, we propose an approach that uses an extended Bloom filter fitting for the RDF model. By using the filter combined with bit operations, we are able to know whether a particular data is included in the result set before actually accessing remote RDF data source. This can significantly reduce transfer volume among distributed RDF data sources, and thus the query processing time decreases. We have implemented the proposed approach and performed a performance evaluation. Our evaluation shows that the proposed approach can significantly improve query processing performance in a distributed environment.

1. はじめに

RDF（Resource Discription Framework）[18]に大きな期待が寄せられている．RDFはメタデータ記述のためにW3Cによって策定された枠組みで，主語，目的語，述語の3つ組（RDFトリプル）の集合によって資源に対する様々なメタデータを表現することができる．RDFを用いることで，利用者はあらゆる資源に対して，高度で柔軟なメタデータを特定のプラットフォームに依存しないフォーマットで記述することができる．そのため，多様な応用分野で広く利用されはじめており，たとえば，ユビキタスコンピューティングの分野[15]では，実世界に存在するモノや場所の状態や情報を表現するためにRDFを用いることが提案されている．このような背景から，RDFに基づいて記述されたデータ（以下，RDFデータ）は，大幅に増加しており，その傾向が今後も継続することは容易に予測できる．現在，このような膨大なRDFデータを効率的に検索することが重要となっている．

分散環境での管理方式には，RDFデータを各地で作成し，そこで管理・保守を行う（ボトムアップ）方式と作成されたRDFデータを配置する場所や構造まで，全体を一括して管理する（トップダウン）方式とに分類できる．P2P（Peer to Peer）のようなトップダウン方式を前提とした検索の効率化に関する研究は広く行われている[3),12),13),20)]が，一方でユビキタス環境[15]のようにボトムアップ方式のRDFデータ検索を効率化するような研究はあまり行われていない．しかしながら，セキュリティや一貫性の側面を考慮すると，RDFデータを各サイトで作成し，そのサイトでそのまま管理できるボトムアップ管理方式も非常

†1 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)

に重要である．本稿ではボトムアップ方式の分散環境を前提として RDF データの検索性能の向上を目指す．

RDF の問合せ処理では，関係モデルに比べ，自然結合が頻出してしまう．その理由としては，RDF では資源を URI を用いて識別するために，同一の URI を参照すると，それらは結合されることになる．また，RDF データは二項関係を表現するトリプルと呼ばれる単位の集合によって構成されており，一般的にトリプル単位に分割して格納される．そのため，多項関係を含んだ問合せが与えられた場合，トリプルどうしを結合して多項化を図る必要があるため，結合が頻出してしまう．RDF データの結合処理の効率化を目指した研究はある[4),7),16)] が，いずれも結合処理の結果を実体化して保持しておく手法で，問合せ処理の効率化を図ることは可能だが，結合処理自体を効率化することを目指した研究ではなく，またいずれも分散環境への適用は困難である．また，Stuckenschmidt ら[14)] は，ボトムアップ分散環境で階層化結合索引を構築する手法を提案しているが，結合処理自体の効率化のためではなく問合せ最適化に利用している．

本稿では，ボトムアップ方式の分散環境において，RDF 問合せ処理を効率的に実現するための手法を提案する．我々が提案する手法は，ブルームフィルタ[1)] を用いた手法で，RDF モデルに適用するため多次元化を図り，さらに分散環境へ適用するためにバインディングフィルタと呼ぶ，多次元化ブルームフィルタの転送と演算のためのデータモデルを導入した．この手法により，結合演算の対象となる集合サイズを減少させることができる．その理由は，バインディングフィルタ間での演算を行うことで，解となるデータを実際のデータを参照する前に予測できるため，結合不能タプル（dangling tuple）を削除することができるためである．結合対象の集合サイズを減少させることは，分散環境ではより影響力が大きい．その理由は，分散環境では，2 つの集合を結合する際には，いずれかの集合を他方のサーバに転送したのち，処理をしなければならないことから，通信帯域の節約を図るだけではなく，集合のサイズが大きく，メモリ上で扱いきれない場合，受信した集合をいったんハードディスクに書き込まなければならないため，処理時間に大きく影響する．また，Heine[9)] は，P2P 環境でブルームフィルタを用いて RDF 問合せの効率化を図る手法を提案しているが，結合のための効率化を目指したものではない．

本稿の構成を述べる．2 章で前提知識として，RDF とブルームフィルタの説明を簡単に行い，3 章では，提案手法として，分散 RDF データのためのブルームフィルタの拡張を述べる．4 章では，提案手法の性能評価を行い，5 章でまとめる．

2. 予 備 知 識

2.1 Resource Description Framework

Resource Description Framework（RDF）[18)] は，資源に対する構造的なメタデータを記述するための基礎を提供した枠組みである．RDF に基づいて表されたメタデータは，資源間の二項関係を表現することができる RDF トリプルと呼ばれる基本単位の集合によって構成されている．RDF トリプルは，主語（subject）と述語（predicate），目的語（object）で構成されており，主語は URI，述語は URI，目的語は URI あるいはリテラルによって表現される．RDF トリプルは，主語が示す資源と目的語が示す資源あるいはリテラルとを述語が示す関係で成り立っていることを意味している．トリプルの集合は，主語と目的語を頂点とし，述語が有向辺に対応したラベル付き有向グラフの構造を表現する．RDF データが構成するラベル付き有向グラフを RDF グラフと呼ぶ．

図 1 に単純な RDF グラフの例を示す．この例は，**図 2** の 3 つの RDF トリプルによって構成されている[⋆1]．この例では，この記事（*aist*:`article`）の作成者（*dc*:`creator`）である的野（*aist*:`matono`）の名前（*foaf*:`name`）が "Matono" で，年齢（*foaf*:`age`）が "30" であることを示している．

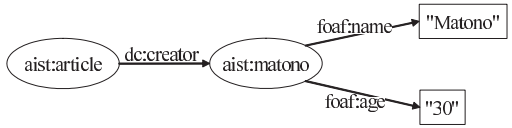

図 1　単純な RDF データ（有向グラフ）
Fig. 1　A simple RDF data (directed graph).

```
1  aist:article dc:creator aist:matono .
2  aist:matono  foaf:name  "Matono" .
3  aist:matono  foaf:age   "30" .
```

図 2　単純な RDF データ（Notaion 3）
Fig. 2　A simple RDF data (Notation 3).

⋆1 記述した構文は Notation 3 （http://www.w3.org/DesignIssues/Notation3.html）で，各資源の接頭辞の名前空間および，リテラルの型は省略する．以下すべての例も同様である．

```
1  SELECT ?y
2  FROM <http://graph>
3  WHERE{
4    aist:article dc:creator ?x .
5    ?x  foaf:name  ?y .
6  }
```

図 3　単純な SPARQL 問合せ
Fig. 3　A simple SPARQL query.

2.2　RDF 問合せ

RDF データ問合せの検索条件部分は，0 個以上 3 個以下の変数を含んだトリプル（以下，トリプルパターンと呼ぶ）の集合で構成されている．これは，RDF データ検索が部分グラフの発見にほぼ等しいためで，トリプルパターン集合が表現するグラフのパターンを条件として，それに一致する部分グラフを検索している．実際にこれまで提案されている RDF データ問合せ言語でも，本質的な処理は部分グラフの発見である．たとえば，SPARQL（SPARQL Protocol and RDF Query Language）[19] は，最も普及している RDF データ問合せ言語の 1 つである．SPARQL には SELECT 文だけでなく，CONSTRUCT や ASK など，4 つの問合せ文が定義されている．いずれの問合せ文でも，結果の出力方法が異なるだけで，部分グラフの集合を発見するまでの処理は等しい．なお，本稿での RDF 問合せは SPARQL によって行うものとする．

図 3 に図 1 で示したデータに対する SPARQL 問合せの例を示す．この例は，*aist*:`article` の作者（*dc*:`creator`）の名前 *foaf*:`name` を求める問合せである．SQL と同様に，SELECT 句は射影対象の変数を指定し，FROM 句は検索対象の RDF グラフを指定する．WHERE 句は目的の部分グラフの条件を示す部分で，この例では 2 つのトリプルパターンによって検索条件が記述されている．SPARQL では，変数は “?” または “$” で始まる文字列で記述されるため，この例では?`x` と ?`y` が変数で，残りの *aist*:`article` と *dc*:`creator`，*foaf*:`name` が定数である．この問合せを 図 1 の RDF データに上記問合せを発行した場合，変数 ?`x` に束縛される要素は *aist*:`matono` で，変数 ?`y` に束縛される要素は "Matono" となり，射影演算によって "Matono" のみが返される．

部分グラフの発見処理を細かく分けると，与えられた各トリプルパターンの定数に基づいて，一致するトリプル集合を検索する処理と，得られたトリプルの集合からグラフを構築する処理の 2 段階からなる．たとえば，図 3 の問合せでは，4 行目と 5 行目のトリプルパターンに一致するトリプルは，それぞれ図 2 の 1 行目と 2 行目で，それらからグラフを構成するには，変数 ?`x` に束縛された値が等しいかどうかの判定，すなわち結合演算の 2 段階の処理を行っている．このように，RDF 問合せでは，図 3 のような単純な問合せであっても，グラフ構築処理のために，暗黙的な和集合や結合などの演算が頻出してしまう．

RDF データの問合せ処理モデル，特に SPARQL モデルは，関係モデルとして扱うことが可能である[5]．たとえば，2 つのトリプル a と b が，a の目的語と b の主語が結合する場合，結果はトリプルではなく，5 つの要素からなるタプルである．また，RDF 問合せのほとんどの演算は，関係モデルの演算に置き換えることが可能であるため，トリプルのモデルとして扱うより，関係モデルのタプルとして扱う方がより自然である．

2.3　Bloom Filter

ブルームフィルタ（Bloom Filter）[1] とは，要素が集合に含まれるかどうかの判定のための空間効率の高い索引手法である．偽陽性（False Positive）による誤検出の可能性があるが，偽陰性（False Negative）はないなどの特徴を持つ．

ブルームフィルタのデータ構造は長さが m のビット配列で，あらかじめ出力する値の範囲が 0 から $m-1$ である k 個のハッシュ関数，すなわち配列の添え字を出力するハッシュ関数を定義しておく．

ブルームフィルタの操作は初期化，要素の追加，要素の検索の 3 つである．要素の削除はできない．ブルームフィルタの初期化は，ビット配列をすべて 0 することである．要素追加の手順は，追加する要素を k 個のハッシュ関数に入力し，k 個の添え字を得た後，それらが位置する配列のビットをすべて 1 にする．時間的計算量は，k の固定時間である．

要素検索の手順は，追加と同様に，検索する要素を k 個のハッシュ関数に入力し，k 個の添え字を得た後，それらが位置する配列のビットがすべて 1 であるかどうかを調べる．もし 1 つでも 0 である場合，その要素はこれまで追加されていないことを意味する．一方，すべてが 1 である場合，その要素が追加されたことがある可能性がある．しかしながら，それらのビットは他の要素を追加したときに偶然全部 1 になった（偽陽性）可能性もあるため，必ずしも検索要素が追加されたことがあるとは断言できない．このように，ブルームフィルタの検索とは要素が集合に含まれるかどうかの判定である．時間的計算量は，1 つでも 0 に遭遇した時点で処理を終了できるため，たかだか k である．

3. 提案手法

3.1 アーキテクチャ

本稿では，ボトムアップ方式の分散環境を前提としており，複数のデータベースサーバが散在し，それらをひとまとめに管理する問合せ処理サーバが 1 つ存在する．**図 4** は提案手法を実装するシステムのアーキテクチャと問合せ処理の流れを示した図である．データベースサーバは，複数の RDF グラフを格納できる RDF グラフデータベースを保持する．一方，問合せ処理サーバは，各 RDF グラフの 3 次元ブルームフィルタ（3.2 節）を格納するためのフィルタデータベースを保持する．本アークテクチャでは，問合せ処理が実行される前に，RDF グラフをデータベースサーバに格納し，3 次元ブルームフィルタを RDF グラフごとに作成（3.2.1 項）し，それらを問合せ処理サーバにアップロードする必要がある．3 次元ブルームフィルタのアップロードは，それが更新されたタイミングで行う．全体のアップロードではなく，変更があった部分ビット配列のみのアップロードでも可能である．

3 次元ブルームフィルタを使った RDF データ問合せ処理の流れは次のようになる．

1) 問合せ処理サーバが RDF データ問合せを受信する．
2) 問合せの構文解析の後，分散プラン木を作成する．プラン木とは，関係モデルでの問合せ処理の過程で生成されるものとほぼ同じで，結合や選択，和，射影などの集合演算の実行順序を表現するために，木構造の頂点に配置したデータで，葉から根に向って実行する．このプラン木を分散 RDF 用に拡張した．拡張した点としては，選択演算の代わりにトリプルパターンに一致するトリプル集合を取得する演算を追加し，分散環境のために演算として送信と受信を追加した点である．
   プランの最適化に関する議論は，本稿では扱わないため，任意のプランを処理することを前提とする．
3) 問合せを構成する各トリプルパターンを各データベースサーバの 3 次元ブルームフィルタに入力し，それぞれバインディングフィルタ（3.2.3 項）を作成する．バインディングフィルタとは，分散環境のために 3 次元ブルームフィルタから切り取った部分ビットデータである．
4) 各バインディングフィルタ間で問合せプランに従って演算を行う（3.2.4 項）．
5) 演算したバインディングフィルタを各 3 次元ブルームフィルタとを比較し，アクセスする必要があるか否かを判定し，必要なければプラン木を枝刈りする．
6) 分散プラン木にバインディングフィルタを付与する．
7) 分散プランを分割し，各部分プラン木をデータベースサーバに送信する．
8) プラン木に従って実行する．この過程では，実行結果を他のデータベースサーバに転送するため，データのやりとりが多く発生する．この転送の前に 9) の不要解の除去が行われる．
9) 実行結果を他のサーバに転送する前に，バインディングフィルタを使って解に含まれないものを除去する（3.2.5 項）．
10) 最終実行結果を問合せ処理サーバに送信する．この送信処理も分散プラン木に含まれる処理の 1 つであるため，送信するデータベースサーバは，分散プラン木によって決定する．
11) 解をクライアントに返す．

本提案手法の新規性は，ブルームフィルタの多次元化とそれを分散環境へ適用するためのバインディングフィルタの導入，およびバインディングフィルタ間の演算である．

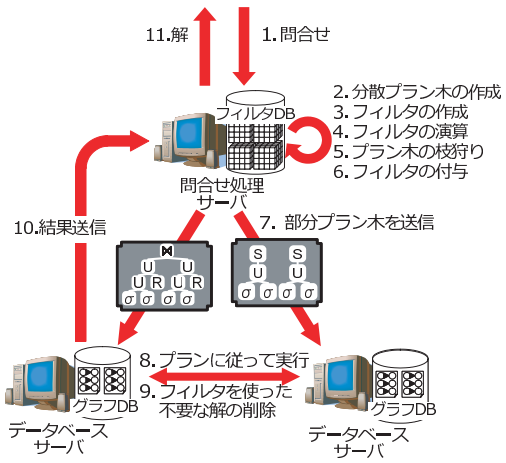

図 4　システムアーキテクチャと処理の流れ
Fig. 4　System architecture and process flow.

3.2 3 次元ブルームフィルタ

RDF データのブルームフィルタは，通常のブルームフィルタを RDF トリプルのモデルに基づいて 3 次元に拡張したものである．通常のブルームフィルタの特徴をそのまま継承しており，要素が集合に含まれるかどうかの判定のための索引手法で，偽陽性による誤検出

の可能性はあるが，偽陰性はない．

3 次元ブルームフィルタの構造は，3 次元のビット配列で，各次元の長さはそれぞれ，m_s，m_p，m_o である．各次元は RDF トリプルの主語，述語，目的語に対応している．また，あらかじめ出力する値の範囲を $[0, l)$ に指定できるハッシュ関数を k 個定義しておく．l は各次元の配列の長さである[⋆1]．本稿では出力する値の範囲が $[0, l)$ であるハッシュ関数を $hash_i^l$（$0 \leq i < k$，$0 < l$）と呼ぶ．実装としては，SHA（Secure Hash Algorithm）アルゴリズムのハッシュ関数を用いる場合，k 個の異なる初期値（たとえば，$0, 1, \ldots, k-1$）を入力要素に加算し，その結果を $[0, l)$ に収まるように変換する関数を介することで，擬似的に出力値の範囲が $[0, l)$ であるハッシュ関数を k 個定義できる．

3 次元ブルームフィルタの操作は，初期化，追加，検索である．初期化処理はすべてのビットを 0 にすることで，通常のブルームフィルタと同じである．追加に関する詳細は 3.2.1 項に，検索に関する詳細は 3.2.2 項に示す．

3.2.1　3 次元ブルームフィルタへの追加

トリプルの追加処理は，与えられたトリプルの各要素を k 個のハッシュ関数に入力し，k 個の 3 次元添え字を得た後，それらが位置する配列のビットをすべて 1 にする．Algorithm 1 に 3 次元ブルームフィルタにトリプルを追加するアルゴリズムを示す．入力されたトリプルを表現した変数 s，p，o は，それぞれ主語，述語，目的語に対応している．2 から 4 行目では，3 次元配列の各次元の長さを最大値とするハッシュ関数に s，p，o をそれぞれ渡し，配列の添え字を取得している．すなわち $s.hashsize = m_s$，$p.hashsize = m_p$，$o.hashsize = m_o$ である．5 行目は 3 次元配列 F 上の 3 次元添え字に対応するビットを 1 に設定している．時間的計算量は，$3k$ の固定時間である．

Algorithm 1：トリプルの追加

Input: 3D bloom filter F, triple s, p, o
Output: 3D bloom filter F

```
1 for i in 1 .. k-1 do
2 |   h_s ← hash_i^{s.hashsize}(s)
3 |   h_p ← hash_i^{p.hashsize}(p)
4 |   h_o ← hash_i^{p.hashsize}(o)
5 |   F[h_s][h_p][h_o] ← 1
6 return F
```

3.2.2　3 次元ブルームフィルタによる検索

3 次元ブルームフィルタを用いた検索処理は，通常のブルームフィルタでの検索のように，要素が存在するか否かの 2 値を返すのではない．それは，ブルームフィルタの多次元化のためと，分散環境での利用を想定しているためである．ブルームフィルタの多次元化により，検索対象の要素が単一の定数ではなく変数を含む複数の要素から構成される．具体的には，検索要素はトリプルではなくトリプルパターンが入力される．そのため，定数に対応するビットを得ることができるが，変数に対応するビットを得ることはできず，部分ビット配列を切り取ることになる．これは，3 次元ブルームフィルタに比べサイズが小さいフィルタであるため，分散環境での利用に適している．本稿では，このフィルタをバインディングフィルタと呼ぶ．

バインディングフィルタとは，マップ構造のデータで，キーとしてトリプルパターンの各要素が，値として k 個の 1 次元ビット配列が格納されたものである．キーは，定数でも変数でもよい．たとえば，以下のような構造をしている．

$$S = \{?x \rightarrow ([1\ 1\ 1\ 0], [0\ 0\ 0\ 1]),\\ p \rightarrow ([1], [1]),\\ ?y \rightarrow ([1\ 0\ 1\ 0], [1\ 0\ 0\ 1])\}$$

バインディングフィルタは，分散環境において転送量の減少に利用できる．バインディングフィルタ間の演算によって，データベースサーバへアクセスする前に，正しい解のハッシュ値に絞り込むことができるため，各 3 次元フィルタと比較することで，対応するデータベースサーバへのアクセスの必要があるか否かを判断できる．また，バインディングフィルタは解集合のハッシュ値情報を保持しているため，データベースサーバから他のサーバに解候補を転送する前に，不要な解を除去でき，転送量を大幅に減少させることができる．さらに，2.2 節で述べたように，RDF 問合せ処理では，タプル構造の方がトリプル構造より適しており，このバインディングフィルタも，それに沿ってタプル構造をしている．

3 次元ブルームフィルタを使った検索はバインディングフィルタの作成，演算の後，データベースサーバ上での判定によって行う．それぞれの処理の詳細は次項以降で述べる．

3.2.3　バインディングフィルタの作成

Algorithm 2 にバインディングフィルタ作成のアルゴリズムを示す．このアルゴリズムでは，3 次元ブルームフィルタ F とトリプルパターン s，p，o を入力とし，バインディング

⋆1 3.2.4 項で述べるバインディングフィルタの演算の際，ビット配列は，m_s，m_p，m_o の組合せの最小公倍数に揃えるため，より正確には，l は，m_s，m_p，m_o，あるいは，それらの組合せの最小公倍数のいずれかである．

Algorithm 2：バインディングフィルタの作成

Input: 3D bloom filter F, triple pattern s, p, o
Output: Binding filter T

```
1  for i in 0 .. k-1 do
2      if s = constant then H_s ←
       {hash_i^{s.hashsize}(s)}
3      else H_s ← {0, ..., m_s}
4      if p = constant then H_p ←
       {hash_i^{p.hashsize}(p)}
5      else H_p ← {0, ..., m_p}
6      if o = constant then H_o ←
       {hash_i^{o.hashsize}(o)}
7      else H_o ← {0, ..., m_o}
8      A_s ← A_p ← A_o ← {0, 0, ..., 0}
9      for j_s in H_s do
10         for j_p in H_p do
11             for j_o in H_o do
12                 A_s[j_s] ← A_s[j_s] ∨ F[j_s][j_p][j_o]
13                 A_p[j_p] ← A_p[j_p] ∨ F[j_s][j_p][j_o]
14                 A_o[j_o] ← A_o[j_o] ∨ F[j_s][j_p][j_o]
15     T.put(s, i, A_s)
16     T.put(p, i, A_p)
17     T.put(o, i, A_o)
18 return T
```

フィルタが出力される．まず，2 行目から 7 行目で，各要素の存在を示す（変数の場合は，示す可能性のある）ビット配列の添え字の集合を H_s，H_p，H_o に代入する．次に，8 行目から 17 行目では，実際に各要素の存在を示す（可能性のある）ビット配列を取得する処理である．これらを k 回繰り返して最終的にバインディングフィルタ T を得る．時間的計算量は，トリプルパターンの変数の数を v（$0 \leq v \leq 3$），$m_s = m_p = m_o = m$ とすると，km^v である．そのため，変数の数が複数ある場合は計算時間は長くなる．

たとえば，$m_s = m_p = m_o = 4$ かつ $k = 2$ の 3 次元ブルームフィルタに，トリプルパターン $?x$，p，$?y$ を入力したとき，$i = 0$ かつ $hash_0^{p.hashsize}(p) = 2$ であると仮定すると，各要素の存在を示すビット配列の添え字は，$H_s = \{0, 1, 2, 3\}$，$H_p = \{2\}$，$H_o = \{0, 1, 2, 3\}$ となる．仮に，これらのビット値が

Algorithm 3：バインディングフィルタの等結合（JOIN）

Input: Binding filter T, Binding filter S
Output: Binding filter R

```
1  R ← ∅
2  for i in 0 .. k-1 do
3      for key in (T.keySet ∪ S.keySet) do
4          A_t ← T.get(key, i)
5          A_s ← S.get(key, i)
6          if key ∈ (T.keySet ∩ S.keySet) then
           R.put(key, i, A_t ∧ A_s);
7          if key ∈ (T.keySet − S.keySet) then
           R.put(key, i, A_t);
8          if key ∈ (S.keySet − T.keySet) then
           R.put(key, i, A_s);
9  return R
```

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

であるとき，各要素の存在を示すビット配列は，$A_s = [1\ 1\ 1\ 0]$，$A_p = [1]$，$A_o = [1\ 0\ 1\ 0]$ となる．A_s は各行に並ぶ成分の論理和，A_p はすべての成分の論理和，A_o は各列に並ぶ成分の論理和によって得られた値である．$k = 1$ の場合も同様の手順で行う．

3.2.4　バインディングフィルタの演算

バインディングフィルタ間での演算は，問合せ処理サーバ内でプラン木に基づいて行われる．すなわち，和集合や結合などの演算に対応した演算をバインディングフィルタ間で行う．これによって実際のデータでの演算を行う前に解のハッシュ値を知ることができるため，不要なデータの転送を抑制するためのフィルタとして利用できる．

Algorithm 3 にフィルタ間の等結合演算の手順を示す．このアルゴリズムは，2 つのバインディングフィルタが入力され，1 つのバインディングフィルタが返される．$T.keySet$ は T のキー集合を意味しており，3 行目では T と S のキー集合の和を繰り返している．4 行目と 5 行目でそれぞれのビット配列を取得するが，キーが存在しない場合は，0 に初期化された長さ 1 のビット配列を返すものとする．共通のキーの場合はビット配列の論理積を行い（7 行目），一方にしか存在しないキーの場合はビット配列の論理和を行う（8，9 行目）．論

Algorithm 4：バインディングフィルタの和（UNION）

```
Input: Binding filter T, Binding filter S
Output: Binding filter R
1 R ← ∅
2 for i in 0 .. k-1 do
3 |   for key in T.keySet do
4 |   |   A_t ← T.get(key, i)
5 |   |   A_s ← S.get(key, i)
6 |   |   R.put(key, i, A_t ∨ A_s);
7 return R
```

理和や論理積を行う際，ビット配列の長さが異なる場合は，それぞれの長さを最小公倍数に揃えた後，演算を行う．

たとえば，以下にバインディングフィルタの等結合の例を示す．

$$
\begin{aligned}
S = \{&?x \rightarrow ([1\ 1\ 1\ 0], [0\ 0\ 0\ 1]),\\
&p_1 \rightarrow ([1], [1]),\\
&?y \rightarrow ([1\ 0\ 1\ 0], [1\ 0\ 0\ 1])\}\\
T = \{&?x \rightarrow ([0\ 0\ 1\ 0], [1\ 0\ 0\ 1]),\\
&p_2 \rightarrow ([1], [1]),\\
&?z \rightarrow ([1\ 0\ 1\ 1], [0\ 1\ 0\ 1])\}\\
\texttt{JOIN}(S,T) = \{&?x \rightarrow ([0\ 0\ 1\ 0], [0\ 0\ 0\ 1]),\\
&p_1 \rightarrow ([1], [1]),\\
&?y \rightarrow ([1\ 0\ 1\ 0], [1\ 0\ 0\ 1])\}\\
&p_2 \rightarrow ([1], [1]),\\
&?z \rightarrow ([1\ 0\ 1\ 1], [0\ 1\ 0\ 1])\}
\end{aligned}
$$

Algorithm 4 にフィルタ間の和演算の手順を示す．このアルゴリズムも，2 つのバインディングフィルタが入力され，1 つのバインディングフィルタが返される．入力される 2 つのバインディングフィルタ T と S は，キーの集合が等しいという前提がある．4 行目と 5 行目でそれぞれのビット配列を取得し，6 行目でビット配列の論理和を行い，それを R に格納している．Algorithm 3 と同様に，論理和を行うにはビット配列の長さが異なる場合は，それぞれの長さを最小公倍数に揃える必要がある．

Algorithm 5 にフィルタ間の射影演算の手順を示す．このアルゴリズムは，1 つのバインディングフィルタと属性の集合が入力され，1 つのバインディングフィルタが返される．T のキー集合と属性集合に共通するキー（3 行目）の値を取得し，R に代入する（5 行目）．

Algorithm 5：バインディングフィルタの射影（PROJECT）

```
Input: Binding filter T, Attribute set
       {a_1, a_2, ..., a_n}
Output: Binding filter R
1 R ← ∅
2 for i in 0 .. k-1 do
3 |   for key in T.keySet ∩ {a_1, a_2, ..., a_n} do
4 |   |   A ← T.get(key, i)
5 |   |   R.put(key, i, A);
6 return R
```

なお，関係代数にある他の演算は，ブルームフィルタでは削除ができないという理由や，SPARQL 問合せ処理では行われない演算であるという理由から，本稿では特に議論しない．

3.2.5　バインディングフィルタによる判定

問合せ処理ノードで作成・演算したバインディングフィルタは，分散プラン木に付与され，各データベースサーバに転送される．各データベースサーバでは，プラン木に沿って処理する過程で，トリプルパターンに一致するトリプル集合を扱う．それらに対して結合や和集合などの演算が行うことでグラフが構築されるが，他のデータベースサーバのトリプル集合との間で演算を行うには，必ずトリプル集合の転送が発生する．この転送を行う前にバインディングフィルタを適用することで，不要な解の除去を行う．

バインディングフィルタを用いた判定アルゴリズムを Algorithm 6 に示す．入力はバインディングフィルタとバインディングセットで，出力はブーリアンである．入力のバインディングセットとは，関係表における 1 つのタプルのような属性と値のマップである．結果が真（true）である場合は入力したバインディングセットが解に含まれる可能性があることを示しており，偽（false）である場合は解に含まれないことがを示している．偶然，真を返す偽陽性があることに注意する必要がある．アルゴリズムは，T と B に共通するキー（3 行目）に対応する値をそれぞれ取得し（4，5 行目），6，7 行目で $hash_i^{key.hashsize}(v)$ 関数によって配列の添字を取得し，そのビットが 1 であるかどうかを判定している．それらすべてが 1 である場合，解である可能性がある．時間的計算量は，T と B に共通するキーの数を n とすると，0 に遭遇したときに処理を終了できる（8 行目）ので，たかだか kn である．

Algorithm 6：バインディンフィルタによる判定

Input: Binding filter T, Binding Set B
Output: Boolean

```
1 b ← 1
2 for i in 0 .. k-1 do
3     for key in T.keySet ∩ B.keySet do
4         v ← B.get(key)
5         A ← T.get(key, i)
6         pos ← hash_i^{key.hashsize}(v)
7         b ← b ∧ A[pos]
8     if b = 0 then return false
9 return true
```

3.2.6　偽陽性発生率

3 次元ブルームフィルタが誤検出する確率（偽陽性発生率）は，各次元の配列長 m_s，m_p，m_o と k，要素数 n に依存している．$m_s \times m_p \times m_o = m$ とし，n 要素を追加したとき，あるビットが 0 である確率は $(1-1/m)^{kn}$ である．そのため，偽陽性発生率は

$$\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^k \approx \left(1-e^{-kn/m}\right)^k$$

となる．したがって，偽陽性発生率は m が大きいほど低下し，n が大きいほど上昇する．m と n が決定しているとき，偽陽性発生率を最小にする k は $m/n \ln 2 \approx 0.7 \ln 2$ である．このときの偽陽性発生率は $(1/2)^k \approx 0.6185^{m/n}$ となる．たとえば，偽陽性発生率を 0.1 以下に保つためには，$k=2$ で要素数の 6 倍の長さの配列を用意する必要がある．また，偽陽性発生率を 0.01 以下に保つためには，$k=5$ で要素数の 10 倍の長さの配列を用意する必要がある．そのため，データ規模に合わせた k と m を設定すれば，偽陽性発生率を目標値以下に抑えることが可能となる．

ブルームフィルタの演算による偽陽性発生率について述べる．まず集合 S をそれを格納したブルームフィルタに写像する関数を $BF(S)$ とする．また，$max(i,j)$ は i と j の大きい値に写像する関数であるとする．文献 10) によると，集合 A と B において，$BF(A)$ と $BF(B)$ の論理和後の偽陽性発生率は，$BF(A)$ あるいは $BF(B)$ のいずれかの偽陽性発生率，すなわち以下より大きくなる．

$$max\left(\left(1-e^{-k|A|/m}\right)^k, \left(1-e^{-k|B|/m}\right)^k\right)$$

また，$BF(A)$ と $BF(B)$ の論理積後の偽陽性発生率は，$BF(A) \cap BF(B)$ の偽陽性発生率である以下より小さくなる．

$$\left(1-e^{-k|A-(A\cap B)|/m}\right)\left(1-e^{-k|B-(A\cap B)|/m}\right)$$

提案手法では，3 次元ブルームフィルタからバインディングフィルタを作成し，それらに対して演算を行う．バインディンフフィルタの作成では，トリプルパターンが複数の変数を含む場合，論理和を行い，各変数に対応する 1 次元のビット配列を得る．また，バインディングフィルタの演算では，結合演算では論理積を行い，和演算では論理和を行う．これらから，バインディングフィルタの演算による偽陽性発生率も，通常のブルームフィルタの演算と同様であるため，問合せの偽陽性発生率は，トリプルパターンが含む変数が多いほど高く，結合が多いほど低く，和が多いほど高くなる．

4. 性能評価

提案手法の性能を実験を通して評価を行う．評価実験は，OWL [17] 問合せのためのベンチマークである LUBM [8] を用いて，3 次元ブルームフィルタを用いた場合と，用いない場合との問合せ処理時間と転送量を比較した．実験環境は，デフォルト設定で生成した LUBM データの各ファイルを 1 グラフとして，データベースサーバに 1 グラフずつ格納した．実験では，計算機 4 台（CPU: Xeon 3 GHz，Memory: 4 GByte，RedHat 8）を用いて，1 台の計算機上で問合せ処理サーバと評価データ収集用サーバを起動し，残りの 3 台でデータベースサーバをそれぞれ 5 プロセス，計 15 プロセス起動した．問合せの種類，3 次元ブルームフィルタの各次元の配列の長さ，ハッシュ関数の数，サーバ数の 4 つである．

実装では，Sesame [2] の Native Store をグラフデータベースとして利用したが，1 つのトリプルパターンに一致するトリプルの集合を返すインタフェースが提供されていれば，データベースの実装には依存しない．そのため，ラッパを介することで，異なるトリプルストアを統合した問合せを処理できる．

また，プロトタイプでは 3 次元ブルームフィルタの実装として，主語と目的語のためのビット配列の長さはパラメータ設定によって決定し，述語のためのビット配列は述語の種類数によって決定するように実装した．すなわち，述語のためのビット配列の長さは，述語の種類数と等しくなる．これは，述語の種類数が，主語や目的語の種類数に比べ，非常に小さ

いために，主語や目的語のためのビット配列のように，大きな空間を用意してもほとんどが利用されないためである．そのため，本章でのパラメータの配列の長さは，主語と目的語のためのビット配列の長さである．

索引構築の際の転送量は，ビット配列の大きさに比例して増大することは明らかであるため，本評価では割愛する．ブルームフィルタは空間効率が良いことが特徴であるため，その転送量は検索の際の転送量に比べ，十分小さいといえる．特に，本実装では述語のためのビット配列長を述語数と等しくしているため，無駄がなく，より空間効率は良い．

OWL とは，RDF の 3 つ組というモデルで，オントロジを記述するための語彙を定義した仕様である．そのため，LUBM が提供する 14 種類の問合せのうち，12 種類が OWL や RDF スキーマで定義された語彙を利用した推論が必要で，そのままでは問合せが表現する意味が大きく異なってしまう．たとえば，クラス Person のインスタンスを求める問合せは，推論を行わない場合は，クラス Person の直接のインスタンスが解集合であるが，推論を行うと，クラス Student やクラス Professor など Person を含め 21 クラスのインスタンスが解集合となる．このように継承や推移律，対象律など語彙の意味を解釈した処理が必要になる．

このような推論を実現する素朴な方法としては，データに推論展開を行う方法（Sesame, Jena [11]）と，問合せに推論展開を行う方法（Oracle [4]）の 2 種類がある．どちらの方法であっても，公平な評価を行うことができるが，本稿では LUBM で定義された問合せを手動で推論展開させることで，推論機能の実装が不要になるため，後者を採用した．推論展開した問合せは非常に複雑化しており，たとえば，Query 13 は，本来 2 つのトリプルパターンから構成される問合せであるが，推論展開によって 26 ものトリプルパターンに膨張する．

図 5 に各問合せの処理時間を比較した図を示す．縦軸は処理時間 (ms) で，横軸は問合せである．また，**図 6** に各問合せの転送量を比較した図を示す．縦軸は転送量 (kByte) で，横軸は問合せである．いずれも縦軸は対数目盛である．比較対象のパラメータは，3 次元ブルームフィルタを用いない場合（WITHOUT）と，配列長が 64 でハッシュの数が 1 の場合（array=64，hash=1），配列長が 256 でハッシュの数が 2 の場合（array=256，hash=1），配列長が 1024 でハッシュの数が 3 の場合（array=1024，hash=3）の 4 種類である．いずれもデータベースサーバ数は 5 である．

提案手法を用いた場合に，転送量を低減させることができたのは，図 6 からは判断しにくい Query 9，11 を含め，11 問合せであった．一方，提案手法を用いたうち 1 つでも，処理時間を低減させることができたのは 8 問合せであった．このことから，必ずしも転送量の

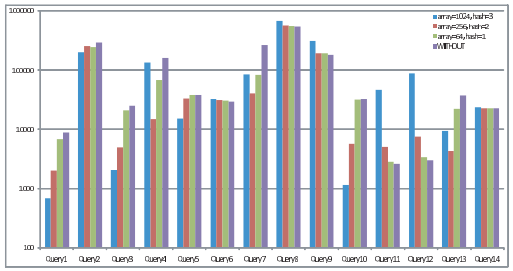

図 5　各問合せの処理時間（DB サーバ数 15）
Fig. 5　The processing times of each query using 15 DB servers.

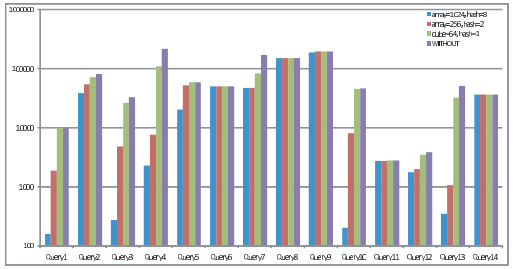

図 6　各問合せの転送量（DB サーバ数 15）
Fig. 6　The amount of data transferred among 15 DB servers for each query.

低減が処理時間の低減につながるとは限らないことが分かる．しかしながら，提案手法を用いて逆に転送量が増えた問合せは必ず処理時間も増加しているため，転送量減少が処理時間低減の必要条件であると判断できる．また，Query 4，7，13 のように配列長とハッシュ数を増加させると，転送量以外の要因で，途中から処理時間が増加している．これらから，バインディングフィルタの作成と演算が時間を要する処理であると判断できる．

各問合せは次のように分類できる．

A)　転送量および処理時間がともに減少した問合せ：Query 1，2，3，4，5，7，10，13

B)　転送量および処理時間がともに増加した問合せ：Query 6，8，14

C)　転送量は低減したにもかかわらず，処理時間が増加した問合せ：Query 9，11，12

A) は，提案手法の効果があった問合せで，最も効果が高かった場合，最大 228 分の 1 に転送量を減少させ，最大 28 分の 1 に処理時間を減少させることができた．これらのうち，

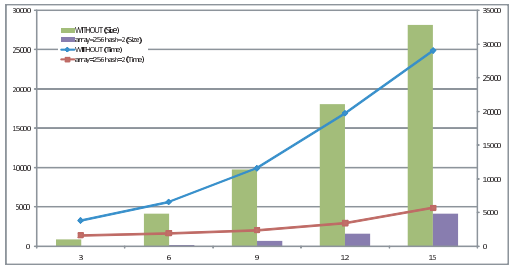

図 7　サーバ数の増加に対する転送量と処理時間
（Query 3，配列長 256，ハッシュ数 2）

Fig. 7　The transferred size and the processing times for Query 3 while the number of DB servers increases.

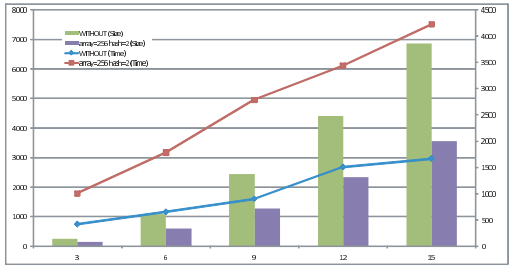

図 8　サーバ数の増加に対する転送量と処理時間
（Query 12，配列長 256，ハッシュ数 2）

Fig. 8　The transferred size and the processing times for Query 12 while the number of DB servers increases.

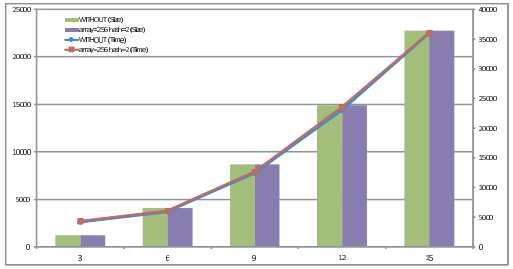

図 9　サーバ数の増加に対する転送量と処理時間
（Query 14，配列長 256，ハッシュ数 2）

Fig. 9　The transferred size and the processing times for Query 14 while the number of DB servers increases.

バインディングフィルタ処理の影響を受けている問合せは，Query 2，4，7，13 である．B) は，提案手法の効果がいっさいなかった問合せで，Query 6 と 14 は結合演算が存在しない問合せである．Query 8 は結合演算が存在する問合せではあるが，絞り込みが効くトリプルパターンがいっさいなく，転送量を減少させることができなかった．C) は転送量の面では効果があったが，処理時間への効果はなかった問合せである．これもバインディング処理のためであるため，処理時間低減のためにはバインディング処理コストの低減が課題となる．

バインディング処理の影響を受けた問合せに共通して見られる特徴は，トリプルパターンに 2 つの変数を含む場合で，これは，2 次元のビット配列から 1 次元のバインディングフィルタを作成する処理が発生するためで，この手法の改善が処理時間低減につながると考えられる．

前述の分類に基づいて，それぞれの特徴が最も顕著であると判断できる問合せを選択し，それらを以降の実験で用いる．選択した問合せは，A) から Qeury 3，B) から Query 14，C) から Query 12 とした．

図 7 と**図 8**，**図 9** に，Query 3，12，14 でのサーバ数の増加に対する転送量と処理時間の変化をそれぞれ示した．パラメータは配列長が 256，ハッシュ数が 2 である．横軸はサーバ数，左の縦軸が問合せ処理時間 (ms) で，右の縦軸が転送量 (kByte) である．いずれも折れ線グラフが処理時間を，棒グラフが転送量を表している．なお，実験では各サーバのデータサイズは一定であるため，本実験はデータ量増加の傾向を見る実験であるともいえる．

転送量はいずれの場合でも，サーバ数の増加に対して指数関数的に増加する傾向があることが分かる．このことから，サーバ増加時にで問合せ処理を行う場合，転送量の増加が問題となると予測できる．そのような状況下では，Query 14 のように転送量を減少できない場合でも，ほとんど処理時間に影響を与えていないため，Query 12 のようなバインディングフィルタ処理が悪影響を与える問合せを除いて，転送量を減少させることができる提案手法が有効である．

そのためには，バインディングフィルタ処理が悪影響を与える問合せを問合せ最適化の段階で予測できる仕組みが必要で，これは与えられた問合せから判断可能で，実現困難な問題ではない．また，バインディングフィルタ処理の高速化の余地も十分あると考えており，今後の課題としたい．

図 10 と**図 11**，**図 12** に，Query 3，12，14 での配列長あるいはハッシュ数の増加に対

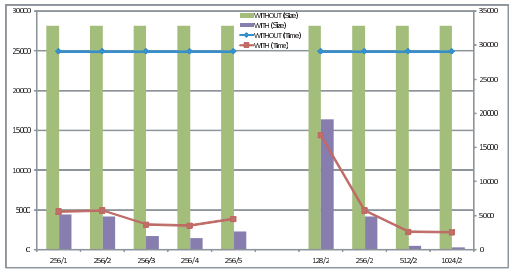

図 10　ハッシュ数あるいは配列長の増加に対する転送量と処理時間
（Query 3，サーバ数 15）

Fig. 10　The transferred size and the processing times for Query 3 while the array length or the hash increases.

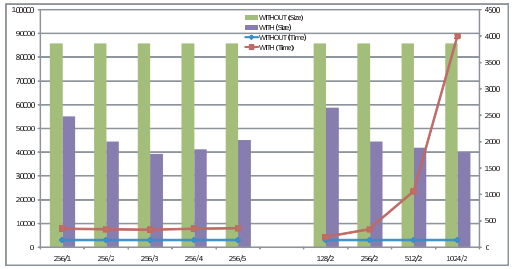

図 11　ハッシュ数あるいは配列長の増加に対する転送量と処理時間
（Query 12，サーバ数 15）

Fig. 11　The transferred size and the processing times for Query 12 while the array length or the hash increases.

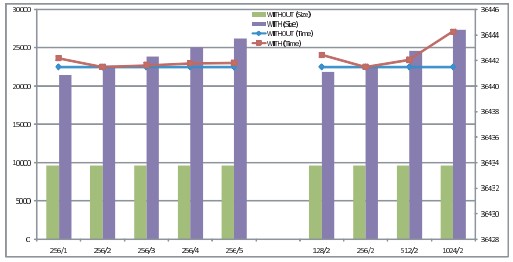

図 12　ハッシュ数あるいは配列長の増加に対する転送量と処理時間
（Query 14，サーバ数 15）

Fig. 12　The transferred size and the processing times for Query 14 while the array length or the hash increases.

する転送量と処理時間の変化をそれぞれ示した．パラメータはサーバ数が 15 である．横軸は前半 5 つがハッシュ数を増加させた場合で，後半は配列長を増加させた場合である．左の縦軸が問合せ処理時間 (ms) を，右の縦軸が転送量 (kByte) を表している．いずれも折れ線グラフが処理時間を，棒グラフが転送量を表している．

提案索引を用いない場合，配列長やハッシュ数に依存しないため，いずれも一定である．ハッシュ数の増加によって，Query 3，12 でいったん減少した転送量が途中から増加している．これは偽陽性が発生したことが原因と考えられる．Query 14 でも，転送量が増加しているように見えるが，これは純粋にブルームフィルタの増加分で数 kByte であるため無視できる．処理時間は，Query 12，14 はほぼ一定であるが，Query 3 は転送量に沿って変化している．このことから，Query 3 はバインディングフィルタ処理の影響をほとんど受けていないことが判断できる．配列長の増加によって，Query 3，12 では転送量を減少させることができた．Query 14 は先ほどと同様で無視程度である．処理時間は，やはり Query 3 は転送量に沿って減少しており，バインディングフィルタ処理の影響はないが，Query 12 は転送量の減少に反して増加している．このことから，Query 12 は転送量の減少がほとんど影響しない問合せであるといえる．また，Query 14 も転送量はほとんど変化していないにもかかわらず，わずかではあるが処理時間は増加している．これもバインディングフィルタ処理の影響であると考えられる．

3.2.6 項でデータ規模に基づいて偽陽性発生率が低くなるよう配列長やハッシュ数を設定すれば，どのようなデータ規模であっても提案手法の性能を発揮できると述べた．今回の実験では，データ規模としては，およそ 10 万トリプルを 15 サーバ上に分散して行った．ネットワーク帯域やデータセットなどにも依存するが，この程度の規模であれば，配列長は 256 のとき，ハッシュ数は 3 のときに，提案手法の性能を最も発揮できたと判断できる．

5. ま と め

本稿では，ボトムアップ方式の分散環境における RDF 問合せ処理の効率化を目指し，RDF モデルに適用するためにブルームフィルタを 3 次元に拡張した．提案した 3 次元ブルームフィルタを用いた検索では，バインディングフィルタと呼ぶ，分散環境に適したサイズの小さい部分フィルタを 3 次元ブルームフィルタから切り取る手法を提案した．作成されたバインディングフィルタは，それらの間での等結合，和集合および射影の演算を行うことで，リモートにあるデータベースサーバへアクセスすることなく，解の存在を知ることができる．これによって不要なアクセスを削除することができるが，本稿の評価実験では，アクセス数

に関する評価は行っていないため，今後その評価を行う必要がある．

また，分散環境における問合せ処理では，あるデータベースサーバで演算した結果を他のサーバに転送し，そこで演算を行うことが一般的である．その転送の前に，バインディングフィルタによって解判定を行うことで，解に含まれないデータを削除することができるため，転送量の減少につながる．データ規模がある数 GByte を超える程度になると，受信したデータを主記憶上に保存できないため，ディスクに書き込むことになる．ディスク IO のコストが高いため，転送量の増減が処理時間に影響を与える．

転送量と処理時間の実験評価によって，大半の問合せの転送量を減少させることができ，最大 228 分の 1 に転送量を減少させ，最大 28 分の 1 に処理時間を減少させることができた．しかし，バインディグフィルタの処理コストが高く，処理時間が増加する問合せもあった．そバインディングフィルタの処理に時間を要する問合せは，問合せ最適化で提案手法を用いないよう回避することは可能であると考えおり，またバインディングフィルタ処理自体の改善の余地もあると考えている．これらは今後の課題としたい．

また，その他の課題としては，ユビキタスなどの実際の環境を想定した，本手法の応用を考えている．ユビキタス環境は，RDF データのボトムアップ方式分散環境であるため，容易に適用は可能であるが，識別子に意味があることや，メタデータの管理主体が存在するなど，ユビキタス環境固有の前提条件が存在するため，それらに合わせた拡張が必要になると考えている．また，ブルームフィルタは追加する度に偽陽性発生率が向上してしまう問題がある．これを改善するためのブルームフィルタの拡張を考えている．さらに，任意次元のブルームフィルタへの拡張も今後取り組んでいく．これによって，RDF のような二項関係だけでなく，多項関係を表現する関係モデルも扱うことができるようになる．

謝辞　本研究は科研費 19700109 の助成を受けたものである．

参 考 文 献

1) Bloom, B.H.: Space/Time Trade-offs in Hash Coding with Allowable Errors, *Comm. ACM*, Vol.13, No.7, pp.422–426 (1970).
2) Broekstra, J., Kampman, A. and van Harmelen, F.: Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, *International Semantic Web Conference*, Horrocks, I. and Hendler, J.A. (Eds.), Lecture Notes in Computer Science, Vol.2342, pp.54–68, Springer (2002).
3) Cai, M. and Frank, M.R.: RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network, Feldman, et al. (Eds.) [6], pp.650–657 (2004).
4) Chong, E.I., Das, S., Eadon, G. and Srinivasan, J.: An Efficient SQL-based RDF Querying Scheme, *VLDB*, Böhm, K., Jensen, C.S., Haas, L.M., Kersten, M.L., Larson, P.-Å. and Ooi, B.C. (Eds.), pp.1216–1227, ACM (2005).
5) Cyganiak, R.: A relational algebra for SPARQL, Technical Report HPL-2005-170, Digital Media Systems Laboratory, HP Laboratories Bristol (2005).
6) Feldman, S.I., Uretsky, M., Najork, M. and Wills, C.E. (Eds.): *Proc. 13th international conference on World Wide Web, WWW 2004*, New York, NY, USA, May 17-20, 2004, ACM (2004).
7) Groppe, S., Groppe, J. and Linnemann, V.: Using an index of precomputed joins in order to speed up SPARQL processing, *ICEIS* (*1*), Cardoso, J., Cordeiro, J. and Filipe, J. (Eds.), pp.13–20 (2007).
8) Guo, Y., Pan, Z. and Heflin, J.: LUBM: A benchmark for OWL knowledge base systems, *J. Web Sem.*, Vol.3, No.2-3, pp.158–182 (2005).
9) Heine, F.: Scalable p2p based RDF querying, *Infoscale*, Jia, X. (Ed.), *ACM International Conference Proceeding Series*, Vol.152, p.17, ACM (2006).
10) Hua, Y. and Xiao, B.: A Multi-attribute Data Structure with Parallel Bloom Filters for Network Services, *High Performance Computing — HiPC 2006*, Lecture Notes in Computer Science, Vol.4297, pp.277–288, Springer Berlin/Heidelberg (2006).
11) McBride, B.: Jena: Implementing the RDF Model and Syntax Specification, *SemWeb* (2001).
12) Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M.T., Brunkhorst, I. and Löser, A.: Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks., *Proc. 12th International World Wide Web Conference, WWW2003*, Budapest, Hungary, 20-24 May 2003, pp.536–543, ACM (2003).
13) Sidirourgos, L., Kokkinidis, G. and Dalamagas, T.: Efficient Query Routing in RDF/S schema-based P2P Systems., *4th Hellenic Data Management Symposium* (*HDMS'05*) (2005).
14) Stuckenschmidt, H., Vdovjak, R., Houben, G.-J. and Broekstra, J.: Index structures and algorithms for querying distributed RDF repositories, Feldman, et al. (Eds.) [6], pp.631–639 (2004).
15) T-Engine Forum：ユビキタス ID アーキテクチャ (2006).
http://www.t-engine.org/japanese/archives/UID-CO00002-0.00.24.pdf
910-S002-0.00.24 / UID-CO00002-0.00.24.
16) Wilkinson, K.: Jena Property Table Implementation, Technical Report HPL-2006-140, HP Laboratories Palo Alto (2006).
17) World Wide Web Consortium: OWL Web Ontology Language (2004).
http://www.w3.org/2001/sw/WebOnt/. W3C Recommendation 10 February 2004.

18) World Wide Web Consortium: Resource Description Framework (RDF) (2004). http://www.w3.org/RDF/. W3C Recommendation 10 February 2004.
19) World Wide Web Consortium: SPARQL Query Language for RDF (2007). http://www.w3.org/TR/2007/PR-rdf-sparql-query-20071112/. W3C Proposed Recommendation 12 November 2007.
20) 的野晃整，サイドミルザパレビ，小島　功：P2P 環境における三次元ハッシュ索引を用いた分散 RDF データベース問合せ処理，情報処理学会論文誌：データベース，Vol.47, No.SIG 8 (TOD 30), pp.121–133 (2006).

(平成 20 年 9 月 20 日受付)
(平成 20 年 12 月 31 日採録)

（担当編集委員　浦本 直彦）

的野　晃整（正会員）

2000 年岡山県立大学情報通信工学科卒業．2002 年同大学大学院情報系工学研究科博士前期課程修了．2005 年奈良先端科学技術大学院大学情報科学研究科博士後期課程修了．博士（工学）．同年産業技術総合研究所にて特別研究員を経て，2007 年同研究所入所．現在同研究所情報技術研究部門所属．RDF データ検索の研究に従事．ACM 会員．

小島　　功（正会員）

1958 年生まれ．1984 年京都大学大学院工学研究科情報工学専攻修士課程修了．同年電子技術総合研究所入所．現在，産業技術総合研究所情報技術研究部門サービスウェア研究グループグループ長．データグリッドの研究に従事．ACM 会員．OGF（Open Grid Forum）および OASIS メンバ．