RDFCube: 分散 RDF データベースのための三次元ハッシュ索引

RDFCube: a Three-dimensional Hash Index for Distributed RDF Databases

的野 晃整♡
サイド ミルザ パレビ♢　小島 功♡

Akiyoshi MATONO
SAID Mirza Pahlevi　Isao KOJIMA

本稿では，分散環境における RDF データに対して効率的に検索するための索引手法を提案する．本手法では，RDF の主語，述語，目的語に基づいた三次元構造のハッシュ空間上に RDF データを写像し，一定の大きさの領域内に写像されたデータが存在するか否かを示すビットを導入した．実際のデータの処理の前に，ビットの演算を行ない，不要なデータを除去することで，結合演算を含む問合せを効率的に処理することができる．

In this paper, we propose an indexing scheme to retrieve distributed RDF data efficiently. The proposed scheme maps RDF triples that compose RDF data into a three-dimensional hash space and introduces bit flags to indicate the existence of RDF triples in the hash space. Before processing the RDF triples, the scheme eliminates unneccessary triples by performing bit operations on the bit flags. This makes it possible to process RDF queries including those with join operations efficiently.

1. はじめに

Resource Description Framework (RDF) [5] はメタデータ記述のための枠組みであり，柔軟で高度な表現能力を提供するものとして大きな期待が寄せられている． RDF は様々な応用分野で広く利用され始めており，そのデータはネットワーク上に散在して増加している．そのため，分散した大量の RDF データに対して効率的に検索することが重要となってきた．

RDF データは，任意の利用者が任意の資源に対する情報を任意の場所で記述できるという特徴を持っている．これは，異なる場所に記述されたメタデータの間に意味的な関連を持つことができることを意味している．そのため，分散環境における RDF データに対する問合せ処理には，ネットワーク上の離れた場所にあるメタデータ同士を効率的に結合する処理 (結合演算) が求められる．

これまで分散した RDF データのための検索手法が幾つか提案されている [1, 2, 3]． しかしながら，これらでは結合演算を効率的に処理することが困難である．例えば，Edutella [2] は，非構造型 P2P ネットワークであるためにネットワーク上のすべてのノードに対してデータを送信し，不要なトラフィックが大量に発生する．拡張 Edutella [3] では，スキーマに基づいたルーティングと，スーパーピアを導入することでこの問題を解決しているが，スキーマをもたない RDF データを扱うことはできない．RDFPeers [1] は，分散ハッシュ表による RDF トリプル検索を提供しているが，RDF トリプルの結合演算を考慮した設計になっていない．

♡正会員　産業技術総合研究所グリッド研究センター
a.matono@aist.go.jp, kojima@ni.aist.go.jp
♢非会員　産業技術総合研究所グリッド研究センター
mirza@ni.aist.go.jp

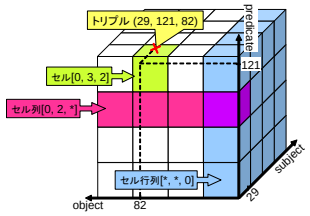

図 1: 三次元ハッシュ空間 (RDFCube)
Fig. 1 A three-dimensional hash space (RDFCube).

本稿では，分散した RDF データに対して，主に結合演算を含む問合せを効率的に処理することを目的とした索引手法を提案する．提案する索引は，分散 RDF データベースのための索引であるため，その索引自体も分散させる必要がある．提案索引は，構造型 P2P である分散ハッシュ表を利用した索引である．まず，RDF データを構成する主語，述語，目的語に基づいた三次元構造のハッシュ空間 (RDFCube) に RDF データを写像する．さらに，RDFCube 内の一定領域 (セル) 内に写像される RDF データが存在するか否かを示すビットを導入し，RDF データ自体の結合演算を行なう前に，ビットの演算を行なう．これによって，解となる RDF データのハッシュ値の範囲を絞り込むことができ，分散環境での結合演算の効率を向上させることができる．

2. 背景

2.1 Resource Description Framework (RDF)

RDF [5] は，資源間の二項関係を表現するための RDF トリプルと呼ばれる基本単位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，主語は資源を，目的語は資源あるいはリテラルを表現し，述語はそれらの間の関係を表現する．資源は一意に識別できる URI によって表現される．

2.2 分散ハッシュ表

分散ハッシュ表 (Distributed Hash Table, DHT) とは，構造型 P2P ネットワークの手法の一つで，スケーラビリティや効率的な検索を提供することができる．分散ハッシュ表では，ノードがハッシュ空間に分散して構築されている．キーに対応した値を格納や検索するには，キーのハッシュ値を担当するノード (successor) を探索する必要がある．このキーの successor を探索する処理は参照 (lookup) と呼ばれる．分散ハッシュ表では，一般的にキーによる完全一致検索のみが可能である．

代表的な分散ハッシュ表として，Chord[4] がある． Chord では，仮想的なリング構造のハッシュ空間上に n 個のノードが分散している．各ノードは前のノードから自ノードまでのハッシュ値の範囲を担当し，その範囲に位置するキーと値のペアを保存する．各ノードは，自ノードからハッシュ値が $2^m (0 \leq m < \log hash_{max})$ の距離にあるノード集合の位置情報を保持しており，これを用いて目的のノードまでルーティングしながら探索する．任意のノードからノードまでの探索ホップ数は，$O(\log n)$ である．

3. 提案索引

3.1 三次元ハッシュ空間 RDFCube

本稿では，分散環境において効率的な RDF 検索を提供するために，*RDFCube* と呼ばれる三次元ハッシュ空間を提案する (図 1). RDFCube は，直交した三つの次元で構成されており，各次元は RDF トリプルの主語，述語，目的語に対応したハッシュ空間を表現している．

ハッシュ関数の最小値を 0，最大値を 2^m とし，RDFCube の各次元を 2^c に分割する ($0 < c < m$)．このとき，主語，述語，目的語

のそれぞれに対応したハッシュ空間上の区間，$[2^{m-c}i, 2^{m-c}(i+1))$, $[2^{m-c}j, 2^{m-c}(j+1))$, $[2^{m-c}k, 2^{m-c}(k+1))$ で囲まれた 3 次元空間を**セル**と呼び，セル座標 $[i, j, k]$ で一意に識別する ($0 \le i < 2^c$, $0 \le j < 2^c$, $0 \le k < 2^c$)． i，j，k を各空間上のセル値と呼ぶ．

いずれかの軸に平行な直線上に存在するすべてのセルの集合を**セル列**と呼ぶ．例えば，主語軸に平行なセル列は $[\{0, \ldots, 2^c - 1\}, j, k]$ と記述し，$[*, j, k]$ と略記する．同様に，いずれかの軸に直交する平面上のすべてのセルの集合を**セル行列**と呼ぶ．例えば，目的語軸に垂直なセル行列は $[\{0, \ldots, 2^c - 1\}, \{0, \ldots, 2^c - 1\}, k]$ と記述し，$[*, *, k]$ と略記する．図 1 にセル $[0, 3, 2]$，セル列 $[0, 2, *]$，セル行列 $[*, *, 0]$ を例示する．なお，セルの集合は，$2^{c2}s + 2^c p + o$ の降順に整列する順序集合である (s，p，o はそれぞれ主語，述語，目的語軸上のセル値)．

軸に直交するセル行列上のセルの座標は (u, v) と表わす．このとき，座標の軸の記述優先順位は，主語，述語，目的語の順に高い．例えば，述語軸に直交するセル行列上のセルの座標は，主語軸上のセル値 i，目的語軸上のセル値 k を用いて (i, k) と表わす．したがって，セル $[i, j, k]$ はセル行列 $[i, *, *]$ 上では座標 (j, k) で識別され，セル列 $[i, *, k]$ はセル行列 $[i, *, *]$ 上では座標の集合 $(*, k)$ で識別される．

RDF トリプルは，RDFCube 内の一つの座標に写像される．写像される座標は，その各要素のハッシュ値の組である．例えば，トリプル *aist*:rdfcube *dc*:creator *aist*:matono を RDFCube 上に写像するとして，仮にトリプルの各要素のハッシュ値が 29，121，82 とすると，与えられた RDF トリプルは， 図 1 に示すように RDFCube の $(29, 121, 82)$ の座標に写像される．また，RDFCube の各空間の区間を $[0, 128)$，分割数を 4 であると仮定すると，このトリプルが写像される座標を担当するセルは，$[0, 3, 2]$ である．

RDFCube の各セルは，自セル内に写像される RDF トリプルが存在するか否かを示す 2 値の状態を保持するビットを持つ．セルの集合 $\{c_1, c_2, \ldots, c_l\}$ のビットの状態を示す関数を $bits(\{c_1, c_2, \ldots, c_l\})$ とする．例えば，$bits(\{c_1, c_2\}) = \{1, 0\}$ は，セル c_1 の空間内に写像される RDF トリプルが存在し，セル c_2 の空間内に写像される RDF トリプルが存在しないことを示す．*bits* に与えられたセル集合がセル列であるとき，その値を**ビット列**と呼び，*bits* に与えられたセル集合がセル行列であるとき，その値を**ビット行列**と呼ぶ．

ビットは各セルに写像されるトリプルが存在するかどうかを表現するため，検索時に，トリプルが存在するセルを決定でき，さらに，ビット同士の演算を行なうことで，解を含むセルを決定することができる．セルはハッシュ値の範囲であるため，ハッシュ値の範囲に収まらないトリプルは解でないと判断できる．これにより，不要なトリプルを除去することができ，検索を効率化できる．なお，RDFCube の基本的な性質は 4. 章 で評価する．

3.2 RDFCube のための分散ハッシュ表

RDFCube ではセル集合とそのビット集合を用いて問合せ処理を効率化するが，トリプルが写像される空間全体を一つの Cube でモデル化するため，問合せ処理の過程で RDFCube データ全体を参照できる必要がある．特に非集中管理型の場合，RDFCube を一つのサイトに構築することが不可能なので，どのように分散環境で実現するかが重要な課題となる．この問題に対して，RDFCube を表現するために分散ハッシュ表を用い，ネットワーク全体で RDFCube を実装する手法を提案する． RDFCube のデータを保存する分散ハッシュ表を *RDFCube DHT* と呼ぶ．

RDFCube DHT は，キーとしてセル行列を用い，値としてそのビット行列を用いる．したがって，一つのセルのビットを格納するとき，そのセルを含む三つのセル行列をキーとして，そのビット行列を lookup によって検索し，更新する．逆に，あるセルのビットを検索するには，セルを含む三つのセル行列のいずれかをキーとして検索し，得られたビット行列から目的の 1 ビットを得る．一般的に，分散ハッシュ表は一つのキーに対して，複数の値を保存するが，RDFCube DHT は一つのキーに対し一つの値のみ保存する．すなわち，同一のキーに対して値を追加することはなく，更新するのみである．

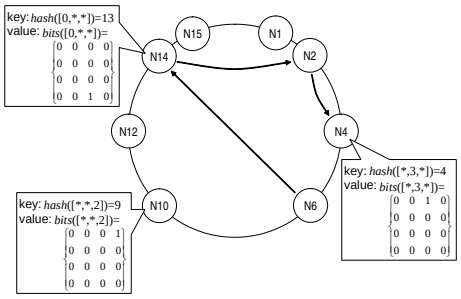

図 2: RDFCube を分散ハッシュ表へ保存
Fig. 2 Storing RDFCube onto a distributed hash table.

Algorithm 1: 索引の構築

```
Input: set of triples T
1 foreach t ∈ T do
2     cell [i, j, k] ←
3         getCell(hash(t.sub), hash(t.pre), hash(t.obj))
4     if onBit([i, *, *], (j, k)) is 0 then
5         onBit([*, j, *], (i, k))
6         onBit([*, *, k], (i, j))
```

例えば，セル $[0, 3, 2]$ 内に写像される RDF トリプルが存在するとすると，セル行列 $[0, *, *]$，$[*, 3, *]$，$[*, *, 2]$ をキーとし，座標 $(3, 2)$ $(0, 2)$ $(3, 2)$ のビットを 1 にしたビット行列 $bits([0, *, *])$, $bits([*, 3, *])$, $bits([*, *, 2])$ を値として格納する．この例を図 2 に示す．セル行列 $[0, *, *]$，$[*, 3, *]$，$[*, *, 2]$ のそれぞれのハッシュ値が 13，4，9 であるとき，それぞれの successor ノードは N14，N4，N10 である．したがって，ノード N14，N4，N10 に，ビット行列 $bits([0, *, *])$, $bits([*, 3, *])$, $bits([*, *, 2])$ がそれぞれ格納される．

一方，$bits([0, 3, 2])$ を知るためには，RDFCube DHT に対して，セル行列 $[0, *, *]$，$[*, 3, *]$，$[*, *, 2]$ のいずれかをキーとして lookup による検索を行なう．図 2 では， $[*, 3, *]$ をキーとして ノード N6 から検索した時のルーティング経路を示してある．この結果得られたビット行列 $bits([*, 3, *])$ が，以下であったとすると，

$$bits\begin{pmatrix} [0,3,0] & [0,3,1] & \underline{[0,3,2]} & [0,3,3] \\ [1,3,0] & [1,3,1] & \overline{[1,3,2]} & [1,3,3] \\ [2,3,0] & [2,3,1] & [2,3,2] & [2,3,3] \\ [3,3,0] & [3,3,1] & [3,3,2] & [3,3,3] \end{pmatrix} = \begin{matrix} 0 & 0 & \underline{1} & 0 \\ 0 & 0 & \overline{0} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}$$

$bits([0, 3, 2])$ は 1 であることがわかる．したがって，セル $[0, 3, 2]$ の空間に写像されるトリプルが存在することが分かる．

3.3 索引の構築

RDFCube DHT を構築する手続きを Algorithm 1 に示す．このアルゴリズムは，入力をトリプルの集合として，与えられたトリプルが写像されるセルのビットを更新する手続きである．まず，2-3 行目で与えられたトリプルが写像されるセル $[i, j, k]$ を `getCell` 関数によって決定し，4-6 行目で，セル $[i, j, k]$ のビットを 1 に更新する．前節で述べたように，$bits([i, j, k])$ を 1 に更新するには，セル行列 $[i, *, *]$ $[*, j, *]$ $[*, *, k]$ をキーとして分散ハッシュ表に lookup を行ない，それぞれの値であるビット行列の座標 (j, k) (i, k) (i, j) のビットを 1 に更新する．

ビット行列上の座標 (j, k)，(i, k)，(i, j) のビットを 1 に更新するには，`onBit` 関数を用いる．`onBit` 関数は，RDFCube DHT に対して lookup を行なう関数で，第一引数のセル行列をキーに lookup を行ない，第二引数の座標を基に，値として保存されているビット行列を更新する．RDFCube DHT 自体はビット行列を値として格納するが，ネットワーク転送量を抑えるために，ビット行列全体ではなく，ビット行列上の更新対象の座標のみを転送する．また，返り値として，更新する前のビットが返される．

Algorithm 2: 索引の参照

Input: set of cells C
Output: set of bits B

```
1  B ← ∅
2  if C is a cell matrix then
3      B ← getBitMatrix(C)
4  else if C is a cell sequence then
5      lookup key key ← createKey(C)
6      set of coordinates S ← createCoordinatesSet(key, C)
7      B ← extractBitSeq(key, S)
8  else
9      foreach cell c ∈ C do
10         lookup key key ← createKey(c)
11         coordinates (x, y) ← createCoordinates(key, c)
12         B ← B ∪ extractBit(key, (x, y))
13 return B
```

また，4 行目に示すように，最初の onBit 関数の返り値によって，残りの二つの onBit 関数を実行するかどうかを判断する．これは，セル行列 $[i,*,*]$ 上の (j,k) のビット，セル行列 $[*,j,*]$ 上の (i,k) のビット，セル行列 $[*,*,k]$ 上の (i,j) のビットは，常に同じ状態であるため，そのうちの一つを確認した段階で，他の二つのビットの状態を判断することができるためである．

3.4 索引の参照

構築した索引を参照するアルゴリズムを Algorithm 2 に示す．このアルゴリズムは，入力として与えられたセルの集合に対応するビットの集合を，RDFCube DHT に対して lookup による検索処理を行い，取得する手続きである．与えられるセル集合によって処理が場合分けされており，セル行列である場合が 2-3 行目，セル列の場合 4-7 行目，それら以外の場合 8-12 行目である．

実際に lookup 検索を行なうのは，3 行目の getBitMatrix 関数，7 行目の extractBitSeq 関数，12 行目の extractBit 関数である．getBitMatrix 関数は，純粋に lookup 検索を行なう関数で，引数として RDFCube DHT のキーであるセル行列を渡し，返り値として RDFCube DHT の値であるビット行列が返される．extractBitSeq 関数は，RDFCube DHT に対して lookup による検索を行ない，その結果得られるビット行列から，目的の列あるいは行 (すなわちビット列) を抽出する関数である．そのため，引数は RDFCube DHT のキーであるセル行列 *key* と抽出対象のビット列の座標集合 S が渡される．5-6 行目は *key* と S を作成する処理である．extractBit 関数は，RDFCube DHT に対して lookup 検索を行ない，その結果得られるビット行列から，目的の座標のビットを抽出する関数である．そのため，引数は RDFCube DHT のキーであるセル行列 *key* と抽出対象のビットの座標 (x,y) が渡される．10-11 行目は *key* と (x,y) を作成する処理である．

4. 索引の応用

RDFCube は，索引であるため実データとして RDF データを格納する分散データベースと併用して用いることを想定している．本節では，提案した RDFCube を分散 RDF データベースのための索引として応用する手法を述べる．本稿では，われわれが文献 [6] で提案した RDFCube のセル構造に基づいた手法 (4.1 節) と RDF トリプルを分散ハッシュ表上に格納する RDFPeers[1](4.2 節) の二つの分散 RDF データベースで RDFCube を索引として利用する場合について述べる．

4.1 RDFCube DB のための索引

本節で対象とする分散 RDF データベースは，文献 [6] で提案した RDFCube のモデルを利用した分散 RDF データベースである．本稿では，このデータベースを RDFCube DB と呼ぶ．RDFCube DB は，RDFCube 上で RDF トリプルが写像されるセルをキーとし，RDF トリプルをその値として分散ハッシュ表に格納する手法である．すなわち，RDFCube DB で提案索引を用いる場合は，RDFCube DB のための分散ハッシュ表と，索引のための分散ハッシュ表 RDFCube DHT の二つを用いる．

RDFCube DB への検索において提案索引を利用するには，1) まず与えられた問合せトリプルの主語，述語，目的語のうち，定数である要素のハッシュ値を求め，セル列あるいはセル行列を求める．これは，解のトリプル集合が写像されている可能性のあるセルの集合を決定していることに等しい．2) 次に，得られたセル集合 (セル列あるいはセル行列) に対応するビット集合 (ビット列あるいはビット行列) を Algorithm 2 によって取得する．3) さらに，結合演算を含む問合せの場合は，ビット集合同士で論理積を行なう．ビット集合同士の論理積を行なうことによって，すべてのセル集合で共通してトリプルが写像されているセルのビットのみが 1 になるため，解の候補が写像されているセルの集合を絞り込んでいることに等しい．最後に，4) その絞り込んだ結果のセルのビットが 1 であるセルをキーとして，RDFCube DB の分散ハッシュ表に lookup 検索を行う．

Listing 1: 結合演算を含む RDF 問合せ

```
1 ?x foaf:name "Matono" .
2 ?x foaf:age  "28" .
```

例を挙げて具体的に述べる．RDFCube の各次元の分割数を 4，ハッシュ値の最大値を 16 としたとき，Listing 1 に示した問合せトリプルの各要素のハッシュ値が $hash(foaf{:}\mathsf{name}) = 2$, $hash(\text{"Matono"}) = 6$, $hash(foaf{:}\mathsf{age}) = 14$, $hash(\text{"28"}) = 9$ であると仮定すると，1 行目の問合せトリプルが写像されるセル集合は $[\{0,1,2,3\},0,1]$ で，2 行目の問合せトリプルが写像されるセル集合は $[\{0,1,2,3\},3,2]$ である．次に，これらのセル集合のビット集合を Algorithm 2 で取得した結果，$bits([*,0,1]) = \{1,1,0,0\}$ と $bits([*,3,2]) = \{0,1,0,1\}$ であったとする．これらのビット列同士の論理積は，$\{0,1,0,0\}$ であるため，解となりえるトリプルが写像されるセルは，$[\{1\},0,1]$ と $[\{1\},3,2]$ である．最後に，$[1,0,1]$ と $[1,3,2]$ をキーに，それぞれ RDFCube DB の分散ハッシュ表に lookup 検索を行なう．

われわれは，RDFCube DB で提案索引を用いた場合と RDFPeers [1] との性能比較を行なった．RDFPeers とは，分散ハッシュ表を用いて RDF トリプルの格納および検索を実現する分散 RDF データベースで，主語，述語，目的語のそれぞれをキーとし，RDF トリプルをその値として分散ハッシュ表に格納する手法である．すなわち，一つのトリプルを格納するには 3 回の lookup が必要で，逆に，主語，述語，目的語のいずれかが決定できれば，それをキーとすることで，その要素を含むトリプルを検索できる．

格納時の結果を図 3 に，検索時の結果を図 4 に示す．格納の効率は，セル数が小さいとき lookup 数で勝ることがわかる．これはビット濃度 (RDFCube 内で 1 であるビットの割合) が高くなるにつれて，ビットの更新を行なう必要がなくなるためである．また，データ転送量が常におよそ 3 分の 1 になる．これは RDFPeers の格納処理が 3 回の lookup を必要とするのに対し，RDFCube DB では 1 回で済む為である．一方，検索効率では，lookup 数では劣るが，転送量ではほとんどの場合で勝ることを確認した．これは RDFCube DB では，解のトリプルが写像されるセル列やセル行列を検索する際，セルの数だけ lookup による検索が必要になるためである．

また，最も効率的な問合せは，複数の問合せトリプルが並列に結合するような問合せ (Query3) であることが確認できる．これは，複数回の論理積ができるため，解が写像されるセル集合の候補を大幅に減少させることができるためである．

4.2 RDFPeers のための索引

前節では，提案索引を利用した RDFCube DB と RDFPeers との性能を比較した．RDFCube DB の問題点として，検索時の lookup 数の増加が見られた．これは，提案索引の問題ではなく，

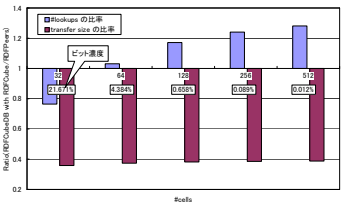

図 3: 格納時の処理比率 1
Fig. 3 Perf. ratio for storing 1.

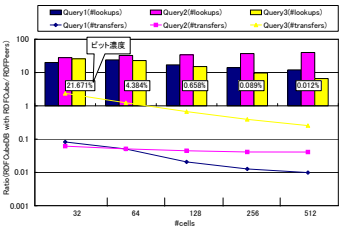

図 4: 検索時の処理比率 1
Fig. 4 Perf. ratio for search 1.

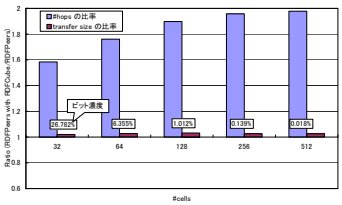

図 5: 格納時の処理比率 2
Fig. 5 Perf. ratio for storing 2.

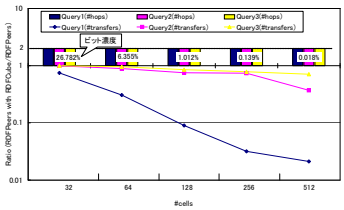

図 6: 検索時の処理比率 2
Fig. 6 Perf. ratio for search 2.

RDFCube DB 自体の問題で，セルをキーに値を格納するため，セルの数だけ lookup による検索が必要になるためである．本節では，提案索引を RDFPeers [1] の索引として利用することとした．これによって，検索時の lookup の回数を減少させることができる．なお，RDFPeers で提案索引を利用する場合も，RDFPeers のための分散ハッシュ表と提案索引のための RDFCube DHT の二つの分散ハッシュ表を用いることになる．

検索の流れを述べる．1) 与えられた問合せトリプルが写像されるセルの集合を決定し，2) それらのビット集合を取得する．3) そのビット集合の論理積を行なう．ここまでは，前節の場合とまったく同じである．最後に，4) 問合せトリプルの定数である一つの要素をキーとして RDFPeers の分散ハッシュ表に対して lookup 検索を行う．この処理は従来の RDFPeers の問合せ処理である．従来の RDFPeers 問合せ処理では，この値を解の候補としてそのまま問合せが発行されたノードに転送する．しかし，5) 本索引を用いる場合，これらの解候補から，論理積の結果のビットが 1 であるセル内に写像されるトリプルのみになるようにフィルタリングして，その結果をノードに転送する．これにより，結合演算を含む問合せの時に，解候補を転送前に減少させることができ，不要なデータ転送量を減少させることができる．

例えば，4.1 節で挙げた例と同じ条件とすると，解となりえるトリプルが写像されるセルの集合 $[1,0,1]$ と $[1,3,2]$ を得るところまでは，前節の場合と同じ手続きである．この後，RDFPeers の分散ハッシュ表に対して，1 行目は $hash(foaf\text{:}\mathsf{name}) = 2$ あるいは $hash(\text{“}\mathsf{Matono}\text{”}) = 6$ をキーとして lookup 検索を行い，値であるトリプル集合が決定される．提案索引を用いない RDFPeers の場合，これらのトリプル集合をそのまま返すが，提案索引を用いることで解候補を返す前にフィルタリングを行なう．具体的には，得られたトリプル集合のうち，セル $[1,0,1]$ に写像されるトリプルのみになるよう解候補を絞り込む．このフィルタリング処理は，lookup 検索の過程でアクセスしたリモートノード上で行なうため，転送前に解に含まれない不要なトリプルを除去でき，転送量を減少できる．同様に，2 行目は $hash(foaf\text{:}\mathsf{age})$ か $hash(\text{“}28\text{”})$ のいずれかをキーとして検索を行ない，得られたトリプル集合のうち $[1,3,2]$ に写像されるトリプルのみになるようフィルタリングを行ない，結果を転送する．

われわれは，RDFPeers で提案索引を用いる場合と用いない場合の比較実験を行なった．その結果を 図 5 と図 6 に示す．格納では，索引を構築した場合が構築しない場合より，コストが高くなることは明らかであるが，ホップ数は 2 倍以下に抑えることができ，データ転送量はほとんど増加しないことが分かった．また，検索では，索引を用いた場合にホップ数は常に 2 倍になる．RDFCube DB の場合で最悪のケースでおよそ 50 倍にもなっていたことを考えると大幅に抑えることができた．さらに，転送量を小さく抑えるという利点を維持していることが確認できる．

RDFCube DB を用いた場合と RDFPeers を用いた場合で，ノードへのアクセス数が大きく異なるのは，RDFCube DB はセルごとにデータを格納しているのに対し，RDFPeers は，RDFCube 上の一つの平面上に存在するすべてのトリプルを格納しているためである．

また，最も効率的な問合せは，複数の問合せトリプルが並列に結合するような問合せ (Query3) であることを確認した．これも前節の傾向を維持している．提案索引の利用は，RDF 問合せ内の一部の問合せトリプルのみに適用できるため，われわれは，より効率的な場合である並列結合部分のみで利用するなどの方法が効果的であると考える．

5. おわりに

本稿では，分散 RDF データベースのための索引手法を提案し，二つの分散 RDF データベースの索引として応用する事例を示した．提案索引は結合演算処理のための索引として利用でき，データの転送量を減少させることができる．一方，索引のデータ自体を分散させるため，その情報を取得するためのアクセスが必要になってしまう．今後の課題として，ノード自体が処理能力を持つ分散環境を想定した改良や，RDF スキーマを持つデータのスキーマ問合せの効率的な検索の提供などがある．

[謝辞]

実験に際し，同研究センターの首藤一幸氏に，多大なご助力を賜りました．ここに記して謝意を表します．

[文献]

[1] M. Cai and M. R. Frank. RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In S. I. Feldman, M. Uretsky, M. Najork, and C. E. Wills eds., *WWW*, pp. 650–657. ACM, 2004.

[2] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, and T. Risch. EDUTELLA: a P2P networking infrastructure based on RDF. In *WWW*, pp. 604–615. ACM, 2002.

[3] W. Nejdl, M. Wolpers, W. Siberski, C. Schmitz, M. T. Schlosser, I. Brunkhorst, and A. Löser. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In *WWW*, pp. 536–543. ACM, 2003.

[4] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pp. 149–160, 2001.

[5] World Wide Web Consortium. Resource Description Framework (RDF). `http://www.w3.org/RDF/`, 2004. W3C Recommendation 10 February 2004.

[6] 的野, サイド ミルザ, 小島. P2P 環境における RDF データのための三次元索引に基づいた検索. In *DBWeb*, IPSJ Symposium Series, pp. 9–16. 情報処理学会, November 2005.

的野 晃整　Akiyoshi MATONO
産業技術総合研究所グリッド研究センター特別研究員．博士 (工学)．RDF 検索の研究に従事．情報処理学会，電子情報通信学会，ACM，日本データベース学会，各会員．

サイド ミルザ パレビ　SAID Mirza Pahlevi
産業技術総合研究所グリッド研究センター特別研究員．博士 (工学)．データグリッド及びセマンティックグリッドの研究に従事．

小島 功　Isao KOJIMA
産業技術総合研究所グリッド研究センターデータグリッドチームチーム長．データグリッドの研究に従事．情報処理学会，ACM，各会員．GGF(Global Grid Forum) および OASIS メンバ．