2008－DD－66 (1)
2008／6／6

分散 RDF データベースシステムのための推論検索処理の実現

浜口 貴弘[†] 塚本 享治[†]
東京工科大学[†]

インターネット上の様々な所に点在する RDF に対して、検索を行う際に分散処理を実現する事で、より詳細な情報を、限られたリソースの上で実現するためのシステムを構築したので報告する。このシステムでは、RDF/XML のグラフとしての特徴と、XML としての特徴の両方を利用して検索する。その方法としては、まず検索時に入力された SPARQL 文を解析し、検索に必要なステートメントを抽出する。次にその抽出された RDF に対して、SPARQL 文と XQuery を併用した検索処理を行い、推論処理に利用するトリプルを抽出する。最後に、推論処理を行い、その後の再帰的な検索により、検索結果を表示する。

Realization of the reasoning search processing for the distributed RDF data base system

Takahiro HAMAGUCHI [†] Michiharu TSUKAMOTO [†]
Tokyo University of Technology [†]

I speak a study to build a system this study depends by realizing distributed processing when I search it for the RDF which the various places in the Internet are dotted with, and to realize detailed information on a thing of a limited resource.This is the search method that used both characteristic as the graph of RDF/XML and characteristics as the XML.For a method, I analyze an SPARQL sentence input at the time of a search and extract the statement that is necessary for a search with XQuery.For extracted RDF, I perform search processing by the SPARQL sentence.Finally it is a thing to perform the inference that I used minimum triple by a recursive search for.

1. はじめに

昨今、発展インターネットにより、世界中のあらゆる情報、多種多様なドキュメントを閲覧できるようになった。その反面、膨大で無尽蔵に増え続けるデータから本当に必要な情報を引き出すことが難しくなってきている。

そこで現在、インターネット上にある膨大なデータ全てをコンピュータにも理解できるように読み替えを行い、それ自体を巨大なデータベースとして再構築しようという流れがある。その１つが RDF[1]を用いた情報付加である。RDF を用いて付加された情報は、常にユニークな情報であり、様々なところに存在しているそれらの情報を繋げる事で、より詳しい情報を得る事が可能になる。

しかし RDF を用いて記述した情報を抽出する際、大量のメモリを必要とする傾向がある。このために１度に処理できる情報量が制限されてしまう。

本研究は一元管理よりもリソースの使用量が少ない分散システムを用いて大量のＲＤＦデータを処理し、情報抽出を行うことができるシステムの構築を行った。

2. RDF データ処理の現状と課題

現存する Web ページをセマンティック Web[2]の構想に則って情報を付加した場合や、各種情報を RDF で公開した場合、ページ容量そのものが膨大なものになる可能性が高い。またその場合、情報の粒度を細かくしていく上で、全てのデータを一元管理した場合生じる問題が多々ある。例えばデータの不整合や、データベースの物理的な容量面から見た細分化の限界等である。

またそれらを検索する上でも、一度データを集めて、その後検索処理を行うのが一般的であるが、それらの手法においても問題がある。全てのデータを転送する手法をとった場合、転送され、集められたデータを解析し、メモリ上に展開する際に大量の領域を必要とする。例えば、WordNet[3]のデータ約 140 万トリプルを解析して扱うためには、約 1084MB のメモリを必要とする。

さらに個別のスキーマで定義されている場合などを考慮し、集めてきたデータに対して推論処理を行う際、解析時の何十倍にもなるメモリを消費する。そうなったときにその巨大なデータを処理する基盤が現状整っていない。

3. アプローチ

ＲＤＦで表記される情報の最小単位は、２つの要素（主語・目的語）とその間の関係性（述語）からなるトリプルである。またトリプルは目的語の一部を例外として全てユニークな URI を持つ(図１)。

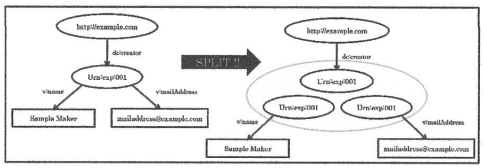

図１　RDF グラフのトリプルへの分割

そのことからトリプルの重複はなく、主語の重複もない。例え主語に対して重複要素があった場合にも、OWL[4]によって同義の物であると定義すれば、それらの情報を共用する事もできる。

本研究では、インターネット上の複数箇所に存在する、RDF で定義されたデータに対し、RDF/XML のグラフとしての特徴と XML としての特徴の両方を利用した検索と推論を実現する。RDF 検索用クエリ言語としては、SPARQL[5]を用いる方法が一般的であるが、SPARQL のみでは推論処理に必要な部分のリソースを抽出する作業に対応しきれない。そのため、DOM と XQuery[6]を併用して情報抽出を実行する。

4. SPARQL 文による RDF の検索

4.1 単一システムにおける検索

ＲＤＦの検索方法として、SPARQL というクエリ言語が W3C[7]から提案されている。これは、上記した主語・述語・目的語を指定してトリプルを特定するクエリ言語で、下図２のようなパターンを作って対象となる意味ネットから情報抽出をおこなうものである。

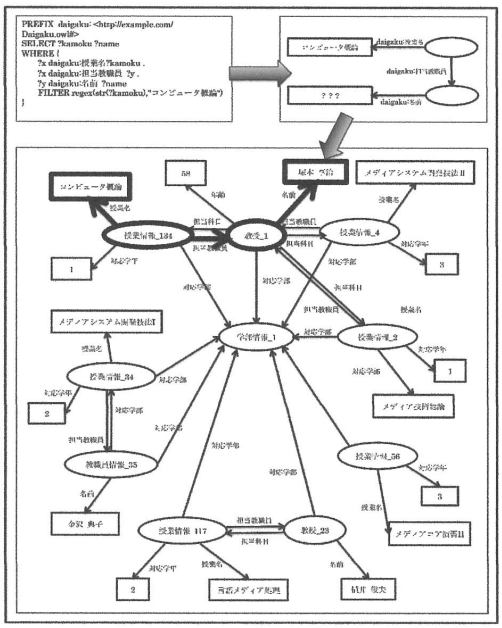

図２　SPARQL 文と検索対象のパターン

しかしRDFを分割し、保存した場合には、パターン自体がサーバーをまたいでしまう可能性が高くなる。このため、SPARQL文から作成したパターンの間にサーバーの境目がきた場合など、単一システムと同じ方法での抽出が難しくなる。

4.2 分散システムにおける検索

前節で述べた問題を解決するために本研究ではSPARQL文を解析し、クエリに含まれているトリプルを抽出。そのトリプルの単位抽出パターンを作成し、各データベースからデータの抽出を行った（図３）。

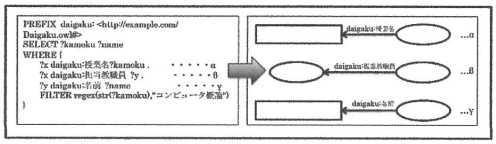

図３　トリプル単位でのパターン化

前章で述べたようにRDFの最小単位であるトリプルまでパターンを分割することで、抽出パターンが複数サーバーに及ぶことを回避し、さらに抽出した必要充分なトリプルを１カ所に集め、検索することで分散システムを実現した。

最終的にはデータを１か所に集め、検索をかけることになるが、SPARQL文から抽出した必要十分なトリプルのみを用いるため、リソースの使用量も最小限の利用のみでよくなる。

4.3 必要部分の推論処理

抽出したトリプルに対して推論処理を行うが、全てのトリプルに対して推論処理を行った際、その処理のために通常の十数倍のメモリ容量と膨大な時間が必要となる。具体例を挙げる場合、６章で実験に使っているRDFに対して、全体で推論を行った場合、メモリが90MB、時間にして約３時間を要した。それを回避するために抽出したリソースを基に、そのリソースの親クラス、その親のインスタンス、それらに関連するリソースの３種を抽出し、その限られたトリプルから推論処理を行った。また、それらの抽出範囲を自在に操作する事で、より自由度の高い部分推論のしくみができた。

5.　システム構築

5.1 システム概要

前章で述べた検索方法を基に、１台のトランザクションマネージャと３台のリソースマネージャからなるシステムを構築した。

5.2 RDFの格納処理

今回実験で利用したRDFデータとして、図４の構造を含むように格納した。

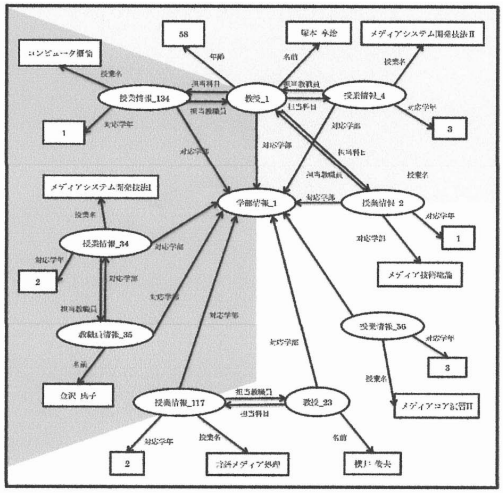

図４　格納したRDFが含む物とする構造

RDFを格納する処理としては、図５のような流れで処理が行われる。

① 利用者はまず、テキスト入力フォーム又はアップロードフォームからURLやファイルを送信する。

② 送信されたデータはセマンティックWebフレームワークであるJena[9]を用いてRDF/XMLの形式に変更される。

③ 出力されたRDF/XMLを『rdf:Description』要素の単位でリソースマネージャの数だけ分割し、それを引数としてリソースマネージャ側のリモートメソッドをXML-RPC[10]を用いて呼び出す。

④ リソースマネージャ側でJenaを用いてRDBへORマッピングを行う。

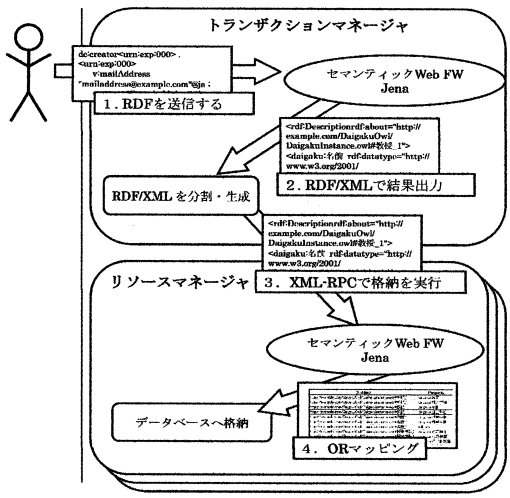

図５　格納処理の流れ図

5.3 SPARQL 文による抽出

前項で格納した RDF データに対して SPARQL 文を用いてで検索を行う場合は、図 6 のような流れで処理を行う。また利用する SPARQL 文は、リスト 7 の物を仮定する。

① 利用者は文章入力フォームに SPARQL 文を入力し、トランザクションマネージャに送信する。

② 送信されたデータは、リモートプロシージャコールによって各リソースマネージャに転送され、リソースマネージャ側で解析が行われる。

③ 解析された SPQRQL 文から検索対象のトリプルパターンを抽出し、そのパターン毎に該当要素を抽出する XQuery を生成する(リスト 8)。

④ RDB から取り出した RDF から RDF/XML 再構築し、③で生成した XQuery を用いてクエリ処理を行う。終了後に Jena を用いてエラーがないかチェックを行う。

⑤ 生成された RDF/XML は RPC の返り値としてトランザクションマネージャ側に返され、トランザクションマネージャ側で結合される。このときトランザクションマネージャ側の所有する意味ネットは、図 9 のような構造になっており、検索経路としても図中にあるような物になっている。

⑥ 最後に①で入力された SPARQL 文の Discribe 構文を用いて検索を行い、検索結果を取得する。

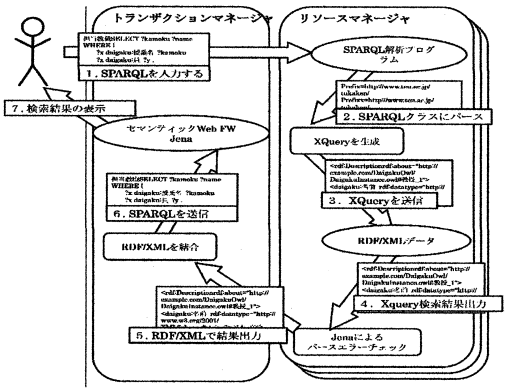

図 6　検索処理の流れ図

リスト 7　検索用 SPARQL 文

```
PREFIX  daigaku: <http://example.com/Daigaku.owl#>
SELECT ?kamoku ?name
WHERE {
      ?x daigaku:授業名?kamoku .        ・・・・・・・・・・・①
      ?x daigaku:担当教職員 ?y .        ・・・・・・・・・・・②
      ?y daigaku:名前  ?name            ・・・・・・・・・・・③
      FILTER regex(str(?kamoku),"コンピュータ概論") ・・・・④
}
```

リスト 8　各要素検索用 XQuery 文

```
{
  for $c in //rdf:Description
  let $n := $c/@rdf:about
  where $n [contains( . , '検索対象の主語')]
  return  $c
}
                                                    主語検索用XQuery
{
  for $c in //rdf:Description
  where fn:string-length($c// 検索対象の述語 /text() ) > 0
  return  $c
}                                                   述語検索用XQuery
{
  for $c in //rdf:Description
  let $n := $c//@rdf:resource
  where $n [contains( . , '検索対象の目的語')]
  return  $c
} (
  for $c in //rdf:Description
  let $n := $c//*
  where $n [contains( . , '検索対象の目的語'   )]
  return  $c
}                                                   目的語検索用XQuery
```

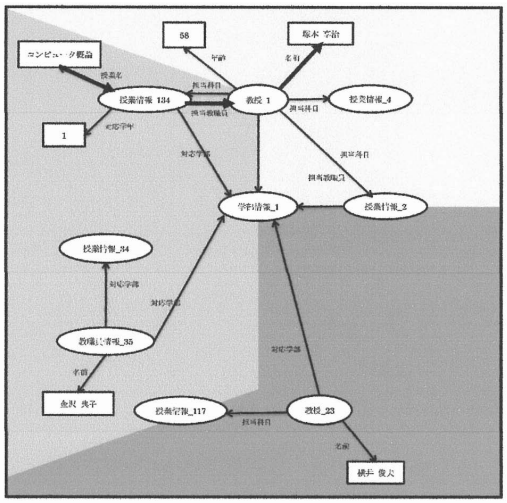

図 9　格納した RDF が含む物とする構造

5.4 推論のための関係調査と処理の実行

推論処理を全てに対して行った場合は、ソースに比べ数十倍とも言えるメモリ容量を必要とする。その為、必要な箇所のみに処理範囲を限定する必要がある。しかし前項で取得した検索結果に対し、単純に推論処理を行うだけでは対象となるインスタンスの数が不十分なため、再帰的な処理を行ってこれを実現する。

① 前項で摘出したリソースに対して、XPath を用いて親クラスの URL を取得する

② 取得した URL からスキーマを取得する

③ 取得したスキーマから、URL に該当する親クラスを抽出してスキーマモデルに追加する

④ 抽出した親クラスに対して、XQuery を利用してそのクラスが持つインスタンスを全て抽出する

⑤ 抽出したインスタンスと、元のリソースの両方に対して、関係している要素の URL を取得する

⑥ 上記の URL を基にリソース、リテラルする

⑦ 全てのインスタンスを１つにした RDF グラフを作成する

上記を１サイクルとして、推論深度(繰り返し回数)を入力することで抽出制度の操作を行う。その後抽出したそれらの RDF に対して下記の操作を行う。

① 抽出した RDF をトランザクションマネージャに集める

② 集められた RDF とスキーマを、各々結合する

③ 結合してできた成果物を基に推論処理を行う

④ 一番最初に与えられた SPARQL を基に関連情報を含む処理結果をユーザーに返す

上記の事を実装する事で、部分的な推論処理を行う。

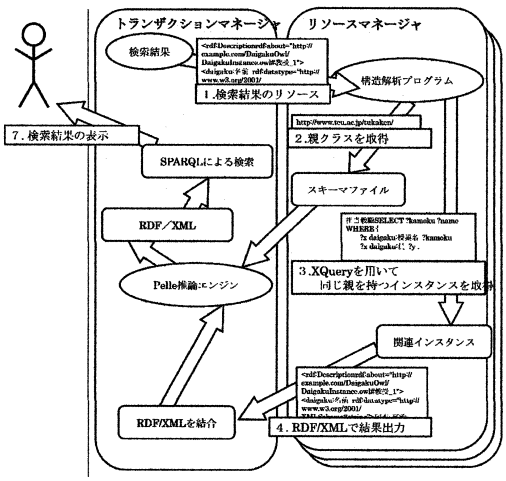

図 10　推論処理の流れ図

6. 性能評価と考察

6.1 システムの性能評価

実験環境として、トランザクションマネージャ１台とリソースマネージャ３台による実験を行った。システムのテストデータとして 1639 トリプルのインスタンスと 498 トリプルを持つスキーマの RDF を利用し、下記の処理と実験を行った。

処理１　RDF のデータベースへの格納
処理２　SPARQL の検索処理

実験I　RPC を使用せず、Java クラスのみで検索を実行する
実験II　単一コンピュータの中で XML-RPC を用いて検索する
実験III　前節で述べた分散環境を用いて検索する
実験IV　実験Ⅰ～Ⅲで求めたモデルに対して、推論深度を指定して部分推論を実行する

実験結果としては下記のリスト７、リスト８のようになった。

リスト７　実験Ⅰ～Ⅲ　結果一覧

所要時間（単位：秒）／最大使用メモリ（単位：MB）	処理１	処理２
実験Ⅰ	10.24	63.32
	86.8	76.3
実験Ⅱ	25.02	118.45
	66.5	86.4
実験Ⅲ	825.33	840.14
	66.5・63.4	57.7・48.3

リスト８　実験IV　結果一覧

推論深度	1	2	3	4	5
総トリプル数	437	1036	1732	132420	160973
平均使用メモリ(単位:MB)	47.82	53.60	60.04	57.40	60.89
平均処理時間(単位:秒)	9.15	11.41	14.31	12.90	15.42

6.2 考察と課題

(1) 本研究で構築したシステムで処理毎に型変換を頻繁に行った。主な変換としては String 型と Document 型、String 型と Model 型、または String 型と Stream 型であった。各ライブラリ間の共用フォーマットとして String 型を中間形式として用いたが、大容量ファイルを扱う際には使用するメモリや処理速度等の面で不安が残る結果となった。

(2) 本研究で構築したシステムは、クエリ量や検索条件によって処理速度が大きく変化する。クエリ量が増えた場合、まずそのクエリの解析に時間がかかる。次にそのクエリが求めているステートメントに対して、各サーバーが持っているステートメントの中でどのくらいの量が該当するかによっても、トランザクションマネージャに返送するデータ量が変化する。そこからネットワークを流れるパケット量が変化し、その結果全体の処理速度に影響が出る。これらについて、与えるクエリと返されるデータ量の関連性について探求することで、今後データ検索の速度を上げることができる。

6.3 おわりに

Jena というフレームワークを使ったため、検索や OR マッピング、表記の違いを吸収する等のことができた。しかし速度面においては Jena の性能を十分に生かしきることができなかった。これは、今回開発したシステムの内部プロセスの複雑さから来るものであり、今後は RDB をさらに有効に活用し、メモリ・速度の両方の面でより効率よく処理する必要がある。

7. 参考文献

[1] W3C "Resource Description Framework" , http://www.w3.org/RDF/

[2] Semantic Web . http://www.w3.org/2001/sw/

[3] WordNet,http://wordnet.princeton.edu/

[4] W3C "OWL Web Ontology Language" , http://www.w3.org/TR/owl-features/

[5] W3C "SPARQL" , http://www.w3.org/TR/rdf-sparql-query/

[6] W3C "XQuery" , http://www.w3.org/TR/xquery/

[7] W3C, http://www.w3.org/

[8] W3C "RDF Vocabulary Description Language", http://www.w3.org/TR/rdf-schema/

[9] Jena SemanticWeb Framework , http://jena.sourceforge.net/

[10] XML-RPC , http://www.xmlrpc.com/

[11] Pellet, http://pellet.owldl.com/

[12] W3C "XML Schema", http://www.w3.org/XML/Schema

[13] W3C "XPath" , http://www.w3.org/TR/xpath20/

[14] W3C "XSLT" , http://www.w3.org/TR/xslt

[15] RedStone XML-RPC Library , http://xmlrpc.sourceforge.net/

[16] RDQL, http://www.w3.org/Submission/RDQL/

[17] Protege 3.2 , http://protege.stanford.edu/

[18] UML Modeling Tool JUDE, http://jude.change-vision.com/jude-web/

[19] Mark Little, Jon Maron, Greg Pavlik : Java Transaction Processing: Design and Implementation

[20] Mark Richards : Java Transaction Design Strategies

[21] Paul R. Reed Jr. : Developing Applications with JAVA and UML