An Improved Indexing Method for Query Processing of Dataspaces

Xuguang Huang*, Dong-Wook Lee*, Soong-Sun Shin*, Sung-Ha Baek*, Hae-Young Bae*
* Dept. of Computer Science and Information Engineering, Inha University

Dataspaces are the collections of heterogeneous and partially unstructured data. It is difficult for the users to explore the data from various data sources using a single schema. And the queries supposed should be allowed to specify varying degrees of structure, spanning keyword queries to more structure-aware queries. Utilizing give the model of heterogeneous data and the definitions of two mainly types of query on dataspaces, in this paper we propose an improved method which can suppose the flexibly query more efficiently.

1. INTRODUCTION

Nowadays, in the real life, the users are faced with multiple data frequently, which include text document, statements, web pages and email etc. Most data management scenarios can deal with all the data above nicely by using relational DBMS, or any other single data model or system which can provide different search and query capability. However, there are two challenges [3] which are ubiquitous to users and developers to manage a set of loosely connected data sources as they must individually and repeatedly address low-level data management challenges across heterogeneous collections. One challenge is user-facing functions such as locating relevant data sources and providing search and query capability. Another challenge is on the administration side which includes enforcing rules, providing availability, recovery and access control.

The idea of dataspaces is to provide base functionality over all data sources. [2] A dataspaces should be a huge collection of heterogeneous and partially unstructured data, and model a rich collection of semantic relationships between them. Most data management scenarios can deal with all the data above nicely by using relational DBMS, or any other single data model or system which can provide different search and query capability. However, since there are many disparate data sources, the user can not query the data using a particular schema. So it is an important task to support that users are able to use varying degrees of structure in their queries, ranging from keyword queries to structure-aware queries.

This paper focuses the issues about indexing dataspaces and proposes an improved method efficiently to suppose for queries that combine keyword and structure. The rest of paper is organized as follows: Section 2 introduces the related works on indexing dataspaces. Section 3 represents the problem statement. Section 4 describes how to build an improve keyword inverted list and how the improved keyword inverted list works. Section 5 presents the results of experimental evaluation. Finally, section 6 concludes our work and discusses future work.

2. RELATED WORK

According to this novel querying requirements introduced in section 1, there are mainly two types of query [6]: predicate keyword queries and neighborhood keyword queries. A predicate keyword query contains the keywords and simple structural requirements which are specified by user. The keywords of a neighborhood keyword query are specified, and it not only returns the data which include the keyword but also the data related with the keywords. The process for XML [9] also can not fit the query above, because XML has tight organizational
structure which quite difference with relationships model in dataspaces.

A novel method proposed by [1] to suppose the queries combining keyword and structured using extended inverted list. This index method takes the structural aspects of the data such as attributes and associations between data items for inverted list labels. An extended inverted list is a two-dimensional table, where the i-th row represents indexed keyword and the j-th column represent instance. The cell (i-th, j-th) records the number of occurrences. When a keyword is attacked to an attribute tag or association tag, it means that the keyword is the value for that attribute or in that association. Then the corresponding results for that keyword query or neighborhood keyword query can be returned. However, combining the attributes and associations with their values as the labels of inverted list is not efficient in answering the flexibly queries, especially when the data set is quite large.

3. PROBLEM DEFINITION

In this section, first we introduce the heterogeneous data model in dataspaces. Then we give the definitions of the two types of query we can support. Base on this data model, we discuss the demerits in existing works and distribute the mainly idea of our method.

Model Heterogeneous Data

Dataspaces are large collections of heterogeneous data such as word, ppt, email, webpage, txt, XML and database. To an object of a data source in dataspaces, it can be described by a set of attributes and relationships whatever the data type is structured or partially structured or unstructured. A DSM schema to store various data was proposed by [4]. DSM uses a triple element (object ID, attribute, value) to store data which can adapt the variety data model. In this paper we represent an instance not only as the form (instance, association) but also as the form (instance, association, attribute, value) which denotes any kind of relationship between two participants. Assuming the association is directional, there might be two directions of association that can be named differently (e.g. author and authored).

EXAMPLE 1 In Fig. 1 there are five instances. A presents a book instance, B notes a publisher instance and C presents a person instance. These three instances are unstructured data. Instance D and instance E are two structured data elements in the RDBMS. For each instance, we can use the attribute values and associations with other instances to represent it. For example, to the B instance, we can represent it with (B, name, “YBM”) and (B, published, A).

Query types on Dataspaces

Dataspaces include heterogeneous data and we aim to suppose flexible queries on heterogeneous data sources with or without structure. The first type of query is called predicate keyword query, describes the desired instances by a set of predicate keywords, each specifying an attribute value or an associated instance. The following is the definition of predicate keyword query.

DEFINITION 1 A predicate key query contains a set of predicate keyword elements. Each of predicate keyword elements is of the form (P, K), where P is either an attribute or association and K is the keyword. A predicate keyword query can be represented as {(P1, K1), (P2, K2), … , (Pn, Kn)}.

To each of predicate keyword, any instances whose attribute or association P includes the keyword K will be return as a result. So to a predicate keyword element, there is a group of instances. The intersection of these groups which is a set of instances is the answer of this predicate keyword query.

EXAMPLE 2 The query “Liana’s book published by YBM” can be described with the following two predicates. This query can be represented by following predicate keyword elements:

(‘author’, ‘Liana’), (‘publisher’, ‘YBM’)

To the first predicate keyword element, users can get a set of books whose authors are ‘Liana’. According the second predicate keyword element, users can get another set of books that published by ‘YBM’. So the intersection of the two sets of books is the result of this query.

The second type of queries, called neighborhood keyword queries. Here we give the definition of this kind of query.

DEFINITION 2 A neighborhood keyword query contains a set of keywords, K1,...,Kn. The result is a set of instances which contains at least one of [K1,...,Kn] in their attributes or associations.

EXAMPLE 3 Consider the query “TOEFL”. Instance A is returned as a result because of its title contains “TOEFL”. Instance B and instance C are results as their associations include “TOEFL”.

(Fig.1) An example of modeling instances with their attributes and associations
Predicate keyword query and neighborhood keyword query is quite different from traditional structured queries in that the user can specify keywords instead of precise values. Keyword Inverted List (KIL) which is an extended inverted list proposed by [1] can answer these two types query easily, however, this extend inverted list commingle the attributes and associations and their values as inverted list label. When answer a query, it has to divide the predicate keywords from the mix label to match the keywords given by query, and it would be time consuming especially when the data sets is very large or there are several levels of sub-attributes.

In this paper, we propose an improved indexing method for dataspaces. This improved method builds an Improved Keyword Inverted List (IKIL) based on KIL to support robust indexing of loosely-coupled collections of heterogeneity data. Unlike KIL, IKIL represents the keywords and the structural aspects of data such as attributes and associations separately to avoid the time consuming split-matching process. So it can answer these two types of query immediately whatever the size of data set and the levels of attributes. As we show in the experiments, the improved method can answer the two kinds of query faster and more efficiently than the method proposed by [1]. The following section gives the details about building IKIL and how to suppose these queries with it.

4. INDEXING STRUCTURE

To suppose the queries introduced in section 2, we build an improved keyword inverted list (IKIL) based on KIL proposed by [1]. Inverted listed is widely used in keyword query. Now we quickly review how an IKIL indexes a set of instances with the triple model for dataspaces.

Different from the KIL, in this paper, the improved keyword inverted list is a two-dimensional table, where the rows represent the indexed keywords K and the columns represent associations or attributes P. The cell at i-th row and j-th column, denoted (K_i, P_j), is an array which includes a set of instances whose attribute or association P_j contains the keyword K_i. The keywords and attributes (or associations) are ordered in alphabetic order, and the cell is the address of the set of instances.

From example 1, the keywords we can collect are “Liana”, “YBM”, “Seoul”, “TOEFL”, “Female”, “Male”, “John”, “Incheon” and “2006”. The attributes and associations we can extract from example 1 are “address”, “author”, “authored”, “published”, “publisher”, “title”, “name”, “sex” and “year”. According these keywords and attributes (associations), we construct an IKIL as table 1 shows.

In this IKIL, the rows represent the keywords and the columns represent attributes or associations which all collected from instances A, B, C, D and E. The cell (K_i, P_j) is an array which includes a set of instances. For example, the cell (“TOEFL”, “Title”) denotes a set of instances including an instance A.

Consider the predicate keyword query “Liana’s book published by YBM” of example 1. This query includes two predicate keyword elements (author, “Liana”) and (publisher, ‘YBM’). To answer the first predicate keyword element (author, “Liana”) using the IKIL above, we can directly get the corresponding array of a set of instances whose attribute “author” contains the word “Liana”. Here, a set of instances {A} is returned as the result. Similarly, to answer the second predicate keyword element (publisher, ‘YBM’), a set of instances {A} as a result is returned according the IKIL above. Then the intersection {A} of the two sets of instances is the result of this predicate keyword query.

We now consider answering the neighborhood keyword query (“TOEFL”) using IKIL. At the row of keyword “TOEFL”, there are three sets of instances {C, D}, {B} and {A}. C, D is a set of instances which are the authors of “TOEFL”. B is a set of instances which publish the book “TOEFL” and A is a set of instances whose names contain the keyword “TOEFL”. So {C, D}, {B} and {A} are returned as the results of this neighborhood keyword “TOEFL”.

Our indexing method can give the results of two kinds of queries immediately requiring only one scanning IKIL. And most important, it uses simply keywords as the rows of inverted list instead of using the keywords combined with attributes or association in KIL which is more efficient and faster. The next section would introduce the experimental evaluation of IKIL comparing with other close methods.

5. EXPERIMENTAL EVALUATION

In this section, we show the experimental evaluation and compare it with other method especially with KIL. The data sources we extract mainly are text files, email and web histories, BIBTEX and LATEX files. And these data are
stored in RDF file, managed by Jena System [10]. We take three RDF files to make the experimental study. The following table shows the details of three data sets.

<table>
<thead>
<tr>
<th></th>
<th>RDF_1</th>
<th>RDF_2</th>
<th>RDF_3</th>
</tr>
</thead>
<tbody>
<tr>
<td>instances</td>
<td>30502</td>
<td>201380</td>
<td>1005028</td>
</tr>
<tr>
<td>attributes</td>
<td>119206</td>
<td>740562</td>
<td>3206102</td>
</tr>
<tr>
<td>associations</td>
<td>167194</td>
<td>892604</td>
<td>5704142</td>
</tr>
<tr>
<td>size</td>
<td>14.2MB</td>
<td>74.5MB</td>
<td>384.6MB</td>
</tr>
</tbody>
</table>

For each data set, we generated and executed 200 queries. A part of these queries are the predicate keyword queries which have one or two or several predicate keyword elements. The others are the neighborhood keyword queries which have one or several keywords. We mainly consider the time of constructing the inverted list and the time of index retrieve.

Fig. 2 and Fig. 3 show the average time of executing predicate keyword queries and neighborhood keyword queries. It is almost same between using KIL and IKIL when the data set is small. However, we can clear find that using IKIL is more efficient than using KIL when the data set is large as using KIL spent much longer time in locating the row area especially executing neighborhood keyword queries. We also can find that KIL is intense increase along with the data set growing. Nevertheless, the IKIL shows more stable increase.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an improve method to support flexible querying over dataspaces. We split the predicate keywords and instances’ attributes or associations with others as the rows and columns of an inverted list. And it is more efficient and faster than the existing method according a set of experiments. In the future, it is also a big challenge to extend our index to support the query which can return the ranked results according to their priority level.

REFERENCE