A Minor Prototype of Personal Dataspaces Management System

Tanvi Shree  
Asst.Prof IT Dept  
IMSEC Ghaziabad  
tnvshree@gmail.com

Upendra Mishra  
Asst.Prof IT Dept  
IMSEC Ghaziabad upendra.mishra13@gmail.com

ABSTRACT
A personal dataspaces includes all personal data of a user. This personal data is generally heterogeneous, rough and tumble in nature which is distributed over various data sources. So, a single system (Personal Data Space Management System) is needed to manage this data. In this paper, we have presented a prototype of Personal Dataspace Management System which is built on the vision presented in [1,2,3]. It represents data from different sources uniformly into triples that are similar to RDF. This paper presents the architecture of this system and advocates usage of MongoDB as a persistent storage.

Keywords
Dataspaces, personal dataspaces management system, triple model, triple store, MongoDB.

1. INTRODUCTION
An explosively increment in the amount of personal data and the limited time and capability of people has evolved PDSMS (personal dataspaces management system). It is built on the visions of dataspaces presented in [1,2,3], which advocates dataspaces as a new abstraction for information management i.e. it requires: (1) no full control on data, i.e., data may be accessed bypassing the interfaces of a PDSMS, (2) simple keyword search on all data available in a dataspaces without performing any semantic data integration, (3) rich querying able to mix structural, attribute, and content predicates, (4) pay-as-you-go integration capabilities, (5) the ability to define arbitrary logical views on all data, (6) durability and consistency guarantees to avoid loss of data assigned to a dataspaces, and (7) update capabilities.

A PDSMS stores, manages personal data, and provides a logical query mechanism. Personal data includes all data pertaining to a user on all his disks and on remote servers such as network drives, email and web servers. This data is represented by a heterogeneous mix of files, emails, bookmarks, music, pictures, calendar data, personal information streams and so on.

The main issue regarding designing of a PDSMS is “how to represent personal data”, i.e. data modeling. Various data models like idm, Triple model[5] etc. are present to represent data in a PDSMS.

A Triple model is a flexible and simple way to represent personal data. It is based on RDF but is more suitable to dataspaces applications. Information from heterogeneous data sources are decomposed into chunks of data unit with the help of wrappers. These chunks of data units are represented by independent data units called triples. A triple composes of Subject, Predicate, and Object.

To provide a system with persistent storage, triples are stored in triple store. Triple store is a database used for storing and querying triples. There are various triple stores available in market which are purposely built for triple store like sesame, mulgara etc. But existing databases like mySQL and Oracle are also being used as triple store.

This paper presents a minor prototype of PDSMS that uses MongoDB[16] as triple store. MongoDB is a noSQL, document oriented database. MongoDB use light weighted BSON (binary JSON) documents. In JSON, data is represented as key-value pair. It is schema-less, provides scalability to the model. MongoDB bridges the gap between key-value stores (which are fast and scalable) and relational database (which have rich functionality).

2. SYSTEM OVERVIEW
All In this section, we discuss the architecture of PDSMS that we have implemented. A PDSMS comprises of many components, but the components that are covered in this system are: data extraction, data modeling, data storage and query execution. The figure given below describes architecture of this system.

Figure 1: Architecture of the system

2.1 System Components
2.1.1 Data Source
Data sources of personal data are heterogeneous i.e. a similar domain data exists at physically separated sources or on a single data source but in different schemas.
2.1.2 Wrappers

Wrappers are constructed corresponding to each data model. It extracts useful information from datasource and decomposes it into smallest autonomous unit of information. Wrappers are constructed by encoding DRS (Decomposition rules set) into it. A DRS is defined as a set of rules built to decompose data into small data units for specific data source.

2.1.3 Dataspase using Triples

Please Dataspase is that space in which information from different data sources can be uniformly represented. Data from different sources co-exist instead of integrated. In our system, data from different sources are represented uniformly in dataspase in triples, all triples are loosely connected without reconciling the semantic heterogeneity.

A triple T is a 3-tuple (S, P, O), where S is Subject component, P is Predicate component, and O is Object component. We define each component of triple as follows:

S is an integer, which is the id of an item.
P is a 2-tuple (l, p), where l is a finite string that represents the label, and p is also a finite string that represents the data type.
O is an array of bytes, which stores the data.

![Figure 2: A Triple](image)

Following syntax is used to create a model and insert triples into it. Triples can be viewed as resources in jena.

```java
// create an empty graph
model = ModelFactory.createDefaultModel();
// create the resource
r = model.createResource();
// add the property
r.addProperty(RDFS.label, "11")
 .addLiteral(RDFS.label, 11);
// write out the graph
model.write( System.out);
}
```

![Figure 3: Create a Model and insert Triples using Jena](image)

A graphical representation of model can be generated using a tool “RDF gravity” which gives the following output:

![Figure 4: Graphical Representation of Triples](image)

2.1.4 MongoDB as a Triple Store

For persistent storage, triples are being stored in some database, which is known as triple store. In this system, we have stored triples in MongoDB which is a noSQL, document oriented database. MongoDB use light weighted BSON (binary JSON) documents. In JSON, data is represented as key-value pair.

Document-oriented key-value stores such as MongoDB are different than triple stores. The key-value store consists of two terms: keys and values, the triple store consists of three terms: subjects, predicates and objects. It is difficult to map a key-value pairs to RDF triples.

In this system, we have stored triple inside a document, each document contain a triple as shown below

```
{ subject: "_:a",
  Predicate: ":http://xmlns.com/foaf/0.1/name",
  Object: "Dominik Tomaszuk"
}
```

![Figure 5: Representing a Triple in MongoDB](image)

These documents are residing inside a single collection. All triple of a graph in jena are mapped to documents of a collection.

2.1.5 Query Converter from SPARQL to MongoDB

It is proposed to introduce a triple store based on document-oriented MongoDB. RDF has different query languages. The official and most widely used is the SPARQL [25] which is explained in section.

MongoDB uses own query syntax and do not support SPARQL. Mongo Query Language is an object-oriented imperative language. SPARQL is a domain-specific declarative language. Therefore, it is difficult to replace the SPARQL to MongoDB Query Language. Therefore, we need to design and implement a tool for SPARQL to MongoDB
query language conversion. The converter receives a SPARQL query which is then parsed and analyzed and converted into Mongo query language. After executing the mongo query, we get rows back from the database. In the end this result is return as an answer to the initial SPARQL query. The working of the converter is shown in figure

![Diagram of converter process](image)

### Figure 6: SPARQL to MongoDB Converter

#### 2.1.5.1 SPARQL Processor
This component follows the Facade design pattern. It hides the complexity of the whole conversion process by providing an `queryProcessorMongoDB()` method that accepts a SPARQL query in string format as an argument and returns an XML result (also in string format). The SPARQL Processor passes on the query to the next component.

#### 2.1.5.2 SPARQL Parser
This component is responsible for parsing the SPARQL query. The component delegates this task to the ARQ module [4] which belongs to Jena Framework[5]. Although Jena and ARQ constitute a complete system supporting RDF storage and SPARQL execution, I use only their SPARQL parsing function.

#### 2.1.5.3 SPARQL Analyzer
This component takes the SPARQL query parsed earlier and traverses its object representation. During this process a new data structure is built to represent the query. Only those elements of the SPARQL syntax which I currently support are reflected in the new data structure. These elements include SELECT, WHERE (triples patterns only, no FILTER nor OPTIONAL). The new query representation is better suited to translation into MongoDB query language than the original one coming from the SPARQL Parser and ARQ.

#### 2.1.5.4 MongoDB query language Generator
This component is responsible for producing a MongoDB query based on the SPARQL query and on selected RDF to DB mapping.

#### 2.1.5.5 SPARQL Query Browser
This Query browser provides a GUI to the user where one can enter query in SPARQL and gets result back. It hides complexity of Triple store from user.

### 3. ASSESSING PERFORMANCE
For persistent storage, we use MongoDB as triple store. To assess performance of MongoDB, MongoDB is compared to MySQL i.e. triples are also stored in MySQL. Jena support storage of triples in MySQL and also contains SPARQL to SQL converter.

I take one database “Addressbook” from MySQL and two folders from File system for simplicity. Total 130 triples are generated which are stored in MongoDB and MySQL databases. The table below shows the load time (time taken to write the data to MongoDB and MySQL).

<table>
<thead>
<tr>
<th>Rounds</th>
<th>MongoDB(ms)</th>
<th>MySQL(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>1828</td>
<td>6297</td>
</tr>
<tr>
<td>2.</td>
<td>1234</td>
<td>5594</td>
</tr>
<tr>
<td>3.</td>
<td>750</td>
<td>5609</td>
</tr>
<tr>
<td>4.</td>
<td>890</td>
<td>6437</td>
</tr>
<tr>
<td>5.</td>
<td>750</td>
<td>6282</td>
</tr>
</tbody>
</table>

For As we can see, MongoDB loads about 164 triples in one second, whereas MySQL loads about 6 to 7 triples in one second. Once I had these datasets loaded I then set up a number of SPARQL endpoints and ran a SPARQL query on triples stored in MongoDB, MySQL and a RDF file stored in system and achieved the following results:

<table>
<thead>
<tr>
<th>Round</th>
<th>MySQL</th>
<th>In-memory</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>1688</td>
<td>1578</td>
<td>1469</td>
</tr>
<tr>
<td>2.</td>
<td>1687</td>
<td>1547</td>
<td>1297</td>
</tr>
<tr>
<td>3.</td>
<td>1718</td>
<td>1578</td>
<td>1312</td>
</tr>
<tr>
<td>4.</td>
<td>1719</td>
<td>1546</td>
<td>1297</td>
</tr>
<tr>
<td>5.</td>
<td>1703</td>
<td>1578</td>
<td>1281</td>
</tr>
</tbody>
</table>

MongoDB takes least time than MySQL and File to execute a query. The reason behind the better performance of MongoDB from MySQL is that MongoDB does not require joins like in case of relational database MySQL.
4. CONCLUSION

In this paper, we have tried to build a prototype of Personal Dataspace Management Systems (PDSMS). It is built on the bases of vision presented personal dataspace [1, 2, 3]. This dissertation has advocated the working of MongoDB as a triple store. By including MongoDB as triple store, its advantages of schema-less, document oriented database and using light weighted JSON document have been optimally exploited. As we can see, performance of MongoDB is better than MySQL in both load time and query execution. Therefore, MongoDB can be used as triple store. This system can be amended by including features like data integration and pay-as-you-go.

5. REFERENCES


[35] A. Muys,“ Building an Enterprise Scale Database for RDF Data".

