XML and NoSQL DBMS: Migration and Benchmarking

Author:
Prakash Thapa

University of Konstanz

December 8, 2014

Abstract

XML and NoSQL database are two growing field in second generation database system, they share some similarities as well as they have some significant difference. This thesis focus on the comparative analysis of these two database system based on the Use cases and existing solution, we will discuss the data processing, query pattern and Information Retrieval(*IR*)

....

Zusammenfassung(German Abstract)

XML und NoSQL

Acknowledgments

.....

Contents

1 Introduction

1.1 Motivation

Ten years ago XML [1] was *de facto* data exchange format. But in recent years a big transformation has been going on in the world of data exchange. The more light weight, less bandwidth-consuming JSON [**?**] structure has been emerge as an alternative to the XML. Even though these two format are comparatively different regarding features and functionality and even though both have their own pros and cons, the rise of JSON as new predominant data exchange format is clearly visible. As a result, new *NoSQL* database systems came along and proved to be successful and the rate of new research papers in this area also increasing. ....

1.2 Contribution

The main contribution of this thesis is that it provides the necessary techniques and algorithms for migrating data from an XML database to NoSQL databases. More specifically, it will focus on document store databases like MongoDB, Couchbase and Rethinkdb as NoSQL and BaseX as XML Database. To approach this challenge, it is first necessary to understand the general architecture and data model of each of these databases as well as the way how they are queried. .....

1.3 Scope of Thesis

....

1.4 Overview

This thesis is divided into three main sections. Section 2 defines the techniques and necessary algorithms to convert XML to JSON. In Section 3 we will present the analyzed Systems and scope of work. Section 4 focuses on the performance tests and comparative analysis of each of the NoSQL databases with BaseX, based on the XMark benchmarking project.

2 Semi-structured data

2.0.1 XML and JSON

JSON and XML look conceptually similar as they both text based markup language, which are designed to represent data in human-readable form, exchangeable across multiple platforms and parseable by common programming language. When we look at them first, they appear to be quite similar, with difference only in their syntax. But it turns out that they are fundamentally incompatible, as we will see in the following section [2].

Document node

Anonymous values

Arrays

Identifiers

Attributes

Namespaces

ALSO definition of XML and XSD is needed here

A JSON document consists of two data structures:

- Objects, which are numbers of key value pairs
- Arrays, which are an ordered list of values content...

A JSON document consists of numbers of key-value pair which is unordered set of name/value named `Objects` and order list of values realized as `Array`.A JSON value can be Object, Array, number, string, `true`, `false` and `null`.

??? need to explain object and array

3 System/Environment

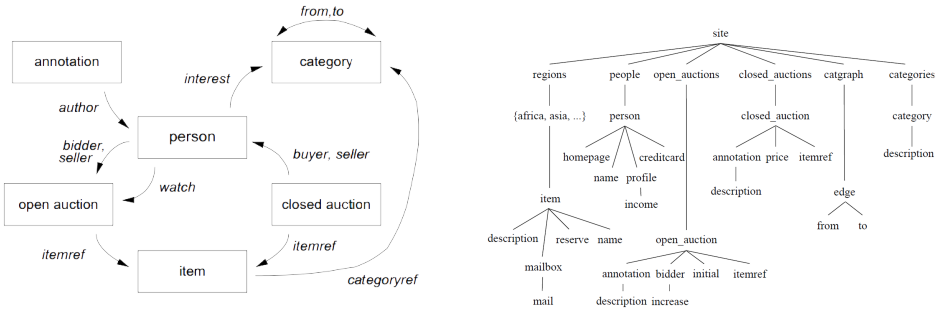
3.1 BaseX

3.2 MongoDB

3.3 Couchbase Server

3.4 Rethinkdb

3.5 Summary



(a) Reference in *XMark* (b) Reference in *XMark* dataset tree [3]

Figure 1: XMark data tree and reference

4 Performance/Experiments

4.1 XMark

The XML benchmarking project XMARK [3] is a single record with large and complex tree structure dataset which is one of the most popular and most commonly used XML Benchmarking project to date [4]. It uses a small executable tool called *xmlgen* that enables to create a synthetic XML dataset according to fixed DTD of an Internet auctions database. The xmlgen produces the single dataset of XML that is system independent and accurately scalable ranging from a minimal document to any arbitrary size limited by the capacity of the system. The textual part of resulting XML documents are constructed from 17,000 most frequently occurring words of Shakespeare's plays.

4.1.1 Dataset

The main entities of XMark data come with two group. In first group `person`,`open_auction`, `closed_auction`, `item` and `category` are expressed through the reference as in Fig. 1(a) whereas second group entities `annotation` and `description` take after the natural language text and are document-centric element structure. `annotation` and `description` are embedded into sub-tree of group one entities.
As shown in Fig. 1(b), `items` are the objects that are for sale or already sold. Each `item` caries a unique identifier having properties like payment information(credit card, money order), `description` , a reference to the seller etc, all encoded as element. Each item is assign to world region like `asia`, `africa` etc. as parent of item. `open_auctions` are auctions in progress which contains bid history(increase/decrease over time) with reference to bidder and seller, current bid etc. `closed_auctions` are the auctions that are completed, which has reference to buyer, seller and item that is sold, price etc. `people` are the information about `person` that are connected to buyer/seller of open_auctions, cloded_auctions etc. `categories` are implemented to classify items which has a name and description. A `catgraph` links categories into a network. The full semantic of XMark dataset can be found in [3].

figure need to create

4.1.2 Queries

The XMark project contains 20 XQuery Queries [3] which focus on various aspect

change the style of queries and complete the queries

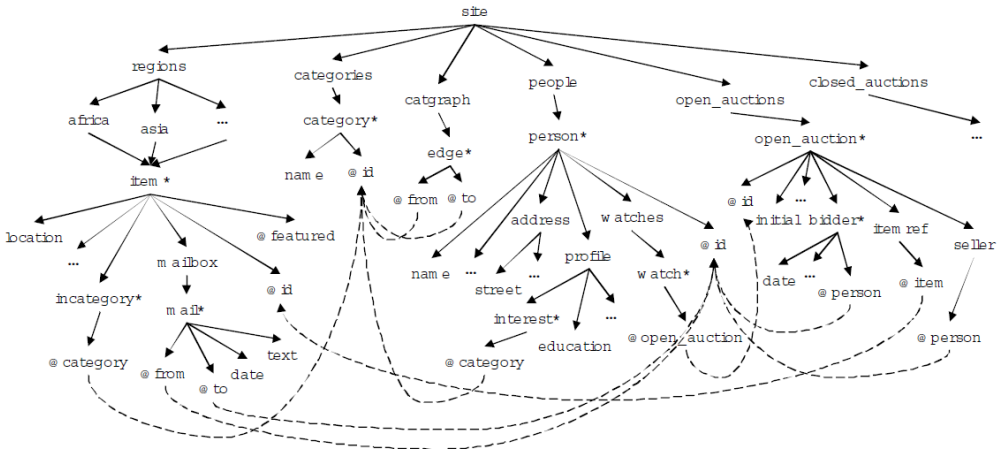


Figure 2: XMark ER-Diagram. Nodes, solid arrows, and dashed arrows represent schema elements (or attributes, with prefix '@'), structural links, and value links, respectively. Elements with suffix '*' are of SetOf type[6]

of language such as agrression, reference, ordering, wildcard expressions, joins, user defined functions etc.[4]

Q1. Return the name of the person with ID "person0".

```
let $auction := doc("xmark.xml") return
for $b in $auction/site/people/person[@id
    = "person0"]
return $b/name/text()
```

Q2. Return the initial increases of all open auctions.

```
let $auction := doc("xmark.xml") return
for $b in $auction/site/open_auctions/
    open_auction
return <increase>{ $b/bidder[1]/increase/
    text() }</increase>
```

Q3. Return the IDs of all open auctions whose current increase is at least twice as high as the initial increase.

```
let $auction := doc("xmark.xml") return
for $b in $auction/site/open_auctions/
    open_auction
where zero-or-one($b/bidder[1]/increase/
    text()) * 2 <=
$b/bidder[last()]/increase/text()
return <increase first="{$b/bidder[1]/
    increase/text()}"
last="{$b/bidder[last()]/increase/text()}"
    />
```

.....

4.2 Evaluation of Test devices

4.3 XMark data into NoSQL Database

A synthetic XMARK dataset consist of one(huge) record in tree structure [5]. However, As mentioned in 4.1, each subtree in schema, `items`, `people`, `open_auctions` , `closed_auctiosn`,`catgraph` and `categories` contains large number of instances in the database which are indexed. In most of NoSQL system, this scenario is different, each instance has it's own index structure, the dataset cannot be in just a huge block. As the data model of NoSQL do not match this single structure-encoded sequence, we breakdown it's tree structure into set of sub structure without losing the overall data and create index for each of them. Another thing, each NoSQL database has their own data model design, unlike most of the XML databases, which have more similar structure than NoSQL, we need to define model design for each of those database separately.

This should be elaborated, *language should be imporve*

???

The generalized concept of XMark data to NoSQL database is given here, but it will be slightly different each of these databases. All sub-structures *items*, *people*, *open_auctions*, *closed_auctions*, *catgraph* and *categories* are the basic for document fragmentation which store first group entities of XMark mentioned in 4.1 `item`, `person`,`open_auction`, `closed_auction` and `category` as individual documents respectively. In each of these document, one special field **doctype** is added to represent the value of the parent. for example, in case of person collection *doctype* will be *people*. This key value set will be also the part of document as shoown in Code 2. There is one exceptional case in for *items* which has **regions** as grandparent and name of different regions like *asia*, *europe* etc. as parent, the *doctype* will be *"items"* as others and one extra field need to be added to represent each region, so there will be one more field in case of *item*.

ref:code:nosql person0

numbers

Code 1: *doctype* and *regions* for item which has region name *asia*

```
1        "doctype":"items",
2        "regions":"asia"
```

numbers

Code 2: General NoSQL data representation of XMARk data

```
1        {
2                "id": "person0",
3                "doctype": "people",
4                "name": "Kasidit Treweek",
5                "emailaddress": "mailto:Treweek@cohera.
                    com",
6                "phone": "+0 (645) 43954155",
7                "homepage": "http://www.cohera.com/~
                    Treweek",
8                "creditcard": "9941 9701 2489 4716",
9                "profile": {
10                       "income": 20186.59,
11                       "interest": [{
12                               "category": "category251"
13                       }],
```

```
                    "education": "Graduate School",
                    "business": "No"
                }
                ....
        }
```

Code 3: XMARK data with of *person0*

```
<person id="person0">
<name>Kasidit Treweek</name>
<emailaddress>mailto:Treweek@cohera.com</emailaddress>
<phone>+0 (645) 43954155</phone>
<homepage>http://www.cohera.com/~Treweek</homepage>
<creditcard>9941 9701 2489 4716</creditcard>
<profile income="20186.59">
<interest category="category251" />
<education>Graduate School</education>
<business>No</business>
</profile>
...
</person>
```

(a) Reference document (b) Embedded document

Figure 3: Mongodb document structure

4.3.1 MongoDB

Mongodb uses the concept of *collections* and *documents* to model data. Collections are the grouping of documents which generally have similar schema. Data has flexible schema where collections do not enforce document to structure rather requirements of our application. Documents are modeled as a data structure following the JSON format which actually store as BSON [7], a binary variant of JSON. . In Mongodb, there are two principle that allow application to represent documents and their relationship: *reference* and *embedded documents*.

Reference Reference stores the relationships between data by including links and references from one document to another as in Figure 3(a). The application can resolve these reference to access the related data

Embedded Embedded documents captures relationships between the data by storing related data in a single document structure. The documents in this method are structured as sub-documentsin the in the form of Array or/and Object [8].

Indexing Each document in Mongodb is uniquely identified by a field *_id* which is a primary index. Hence the collection is sorted by *_id* by default [8].

4.3.1.1 XMARK in MongoDB

Monodbb's collections have a similar and related documents together that helps better indexing ultimately improve in performance. It will not worth to have single collection of whole XMark data as mentioned in 4.3. As we shown in Fig. 2. we create each collection of each substructure in such a way that we will not lose data as well as most representation of xmark data. For Mongodb, the idea(?????) of in 4.3 slightly change. Each *doctype* represented as a collection. so we will have six collections and doctype is already represented by by the collections we don't need key/value of doctype in our document. For *items*, *regions* contains the name of region of each item as usual.

Finally as we mentioned in Indexing, the *id* attribute of each these documents will be renamed to *_id* for default indexing. In case of `closed_auctions` and `catgraph`, system will automatically generate `_id` which is useless for our application. numbers

Code 4: Mongodb data representation of XMARk data

```
1 {
```

A collection is analogous to collection in XML database????

what is BSON, diff btn JSON/BSON

need to change following image/json according to xmark data

Mongodb data model pg4

...??

but note that this primary key index is not a clustered index in Database terms. I.e. the index entries only contains pointers to actual documents in the MongoDB data files. Documents are not physically stored in the order of _id on disks.?? ::ex-

```
2           "_id": "person0",
3           "name": "Kasidit Treweek",
4           "emailaddress": "mailto:Treweek@cohera.com",
5           "phone": "+0 (645) 43954155",
6           "homepage": "http://www.cohera.com/~Treweek",
7           "creditcard": "9941 9701 2489 4716",
8           "profile": {
9                   "income": 20186.59,
10                  "interest": [{
11                          "category": "category251"
12                  }],
13                  "education": "Graduate School",
14                  "business": "No"
15          }
16          ....
17  }
```

4.3.1.2 Queries

coming soon ...

4.3.2 Couchbase Server

Couchbase Server is NoSQL document database system with flexible data model and easily scalable concept [9]. Data in Couhbase Server are stored in logical unit called Buckets. Buckets are isolated to each other which also have their own RAM quota and replica settings. These buckets can be technically compare as `database` in Mongodb or other RDBMS. Couchbase recommends as few as possible number of buckets even it is possible to have up to ten bucket in a single cluster system.These buckets can contains theoretically any type of data(??). All data type other than JSON can be retrieve only by their key. So it is important to check meta type of data stored in a single document before retrieval. In contrast to Mongodb, Couchbase Server don't have concept of collection, documents are separated by the their `doctype`.

This section has to move to 3.2

Metadata For every value stored in database, Couchbase Server generate create metadata that is associated to with the document [10].

Expiration

Check-and-Set(CAS)

Flags

explain in detail

In addition of expiration, CAS and flags, three other meta store at the time of document creation, `Id`, the document's key is saved as part of metadata. `type` is the type of document whether it is JSON for all valid JSON documents and `Base64` for all other than JSON type is being saved as Base64 encoded string.

Document key Every value in Couchbase Server has simple or complex unique key. unlink Mongodb CB don't generate key automatically, it is up to the application creating the data to supply a unique string value up to 250 characters as key for each document **??**.

explain simple or complex

Document design For Couchbase Server, there is no need to change any structure of XMark NoSQL representation as mentioned in 4.3 as there is no concept of fragmentation like in *collections* of Mongodb or *tables* in Rethinkdb. The documents are identified by *doctype*. All of these documents are inserted in a single Bucket with *id* as key. For those documents that doesn't have id field, will be manually generated.

4.3.2.1 Queries

4.3.3 Rethinkdb

RethinkDB [**?**] is distributed database system to store JSON documents that uses efficient query languages named ReQl which automatically parallelize queries in multiple machines. ReQL is based on three main principle: it is completely embedded with programming language, ReQL queries can be passed as pipeline from one stage to another to get required result that means it is possible to use series of simple queries together to perform complex operation. Finally, all the queries are executed in server without any intermediate network round trip required the server and clients.

Data Model The relationships between documents can be can be created by embedded arrays of documents and linking document stores in multiple tables similar to MongoDB in 4.3.1. Rethinkdb stores JSON documents with binary on disk serialization. RethinkDB implicitly support JSON data types: `object`, `array`, `number`, `boolean` and `null`. RethinkDB allows embedded JavaScript expressions anywhere as part of query language

Indexing At the time of table creation, RethinkDB provide option of specifying the attributes that will serve as primary key, by default `id` will serve if primary key attribute is not specified.

4.4 Benchmarking

4.5 Summary

5 Discussion

6 Conclusion

7 Future Work

storing in the memory

References

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, et al. Extensible Markup Language (XML) 1.0 (Fifth Edition). http://www.w3.org/TR/xml, November 2008.

[2] David Lee. Jxon: an architecture for schema and annotation driven json/xml bidirectional transformations. In *Proceedings of Balisage: The Markup Conference*, 2011.

[3] Albrecht Schmidt, Florian Waas, Martin Kersten, Michael J Carey, Ioana Manolescu, and Ralph Busse. Xmark: A benchmark for xml data management. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 974–985. VLDB Endowment, 2002.

[4] Irena Mlỳnková. Xml benchmarking: Limitations and opportunities. Technical report.

[5] Haixun Wang, Sanghyun Park, Wei Fan, and Philip S. Yu. Vist: A dynamic index method for querying xml data by tree structures. In *In SIGMOD*, pages 110–121, 2003.

[6] Cong Yu and HV Jagadish. Schema summarization. In *Proceedings of the 32nd international conference on Very large data bases*, pages 319–330. VLDB Endowment, 2006.

[7] BSON. Bson specification.

[8] Xiaoming Gao. Investigation and comparison of distributed nosql database systems.

[9] Martin C Brown. *Developing with Couchbase Server.* " O'Reilly Media, Inc.", 2013.

[10] David Ostrovsky and Yaniv Rodenski. *Pro Couchbase Server.* Apress, 2014.

List of Figures

List of Tables

Listings