XML and NoSQL DBMS: Migration and Benchmarking

Author:
Prakash Thapa

University of Konstanz

January 13, 2015

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**
- 1.1 Motivation . . . 1
- 1.2 Contribution . . . 1
- 1.3 Scope of Thesis . . . 1
- 1.4 Overview . . . 1

2 Semi-structured data **2**
- 2.0.1 XML and JSON . . . 2
- 2.1 Mapping . . . 3
  - 2.1.1 Friendly . . . 3
  - 2.1.2 Unfriendly . . . 3
  - 2.1.3 Array and Object . . . 3
  - 2.1.4 Mapping approaches . . . 3
  - 2.1.5 Summary . . . 3
- 2.2 Translation from XML to JSON . . . 3
- 2.3 NoSQL database . . . 3
  - 2.3.1 Different Families of NoSQL Databases . . . 3
  - 2.3.2 Querying NoSQL database . . . 3
- 2.4 XML Database . . . 3
  - 2.4.1 XML Query Language . . . 3

3 System/Environment **4**
- 3.1 BaseX . . . 4
- 3.2 MongoDB . . . 4
  - 3.2.1 Data Model . . . 4
  - 3.2.2 Query Model . . . 5
- 3.3 Couchbase Server . . . 5
- 3.4 Rethinkdb . . . 6
- 3.5 Summary . . . 6

4 Performance/Experiments **7**
- 4.1 XMark . . . 7
  - 4.1.1 Dataset . . . 7
  - 4.1.2 Queries . . . 8
- 4.2 Evaluation of Test devices . . . 9
- 4.3 XMark data into NoSQL Database . . . 9
  - 4.3.1 MongoDB . . . 11
    - 4.3.1.1 XMARK in MongoDB . . . 11
    - 4.3.1.2 Queries . . . 11
  - 4.3.2 Couchbase Server . . . 11
    - 4.3.2.1 Queries . . . 12
  - 4.3.3 Rethinkdb . . . 12
- 4.4 Benchmarking . . . 12
- 4.5 Summary . . . 12

5 Discussion **13**

6 Conclusion **13**

7 Future Work **14**

1 Introduction

1.1 Motivation

Ten years ago XML [BPSM+08] 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

XML, by definition a textual markup language where data elements are ordered by nature: *string* is core data type from which richer data types e.g., integers, floats and even user-defined abstract data types are derived [SWK+02].

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 [Lee11].

Document node

Anonymous values

Arrays

Identifiers

Attributes

Namespaces

ALSO definition of XML and XSD is needed here

A JSON or JavaScript Object Notation is programming language model. It is minimal textual and a subset of JavaScript. JSON document consists of two data structures:

- Objects, an unordered collection of name-value pair that are encapsulated by curly braces { and }. The key is a string encapsulated in double quotes and must followed by a column(:) to it's value. The key must be unique for each object.
- Arrays, which are an ordered list of values

A JSON value can be Object, Array, number, string, `true`, `false` and `null`.

2.1 Mapping

2.1.1 Friendly

2.1.2 Unfriendly

2.1.3 Array and Object

2.1.4 Mapping approaches

2.1.5 Summary

2.2 Translation from XML to JSON

2.3 NoSQL database

2.3.1 Different Families of NoSQL Databases

According to [Sen15], there are more than 150 NoSQL databases. Based on data model, these systems are classified in four different categories:

Key/Value store The Data representation of key-value stores are based on attribute pair, the data model is expressed as collection of ¡*key,value*¿ tuples. The key is unique and the query operation on data can be uniquely performed by a key. There are not alternative way to access or modify data without key. DynamoDB, Riak, Redis are some examples of key-value stores.

Document store The document store database are based on semi-structured data model, where unique store a values that contains a tree like structure called *documents*. Document store encapsulate key-value pair in JSON or in JSON like documents[HJ11]. In document store database, data can be access by using key or specific value.

Column family store

Graph database Graph databases represent data as graph that means in the form of nodes and edges like in social network. They store entities and relationship between these entities. Neo4J and OrientDB are two example of these database.

2.3.2 Querying NoSQL database

2.4 XML Database

2.4.1 XML Query Language

(a) Reference document (b) Embedded document

Figure 1: Mongodb document structure

3 System/Environment

3.1 BaseX

3.2 MongoDB

Mongodb is a schemaless document oriented database developed by MongoDB Inc.(then 10gen Inc.) and Open Source Community. It is intended to be flexible data model, fast and Multi-datacenter scalable system written in C++.

3.2.1 Data Model

MongoDB uses the concept of *collections* and *documents* to model data. Collection contain any type of documents but generally it has documents with similar schema. Data has flexible schema where collections do not enforce document to structure rather requirements of our application. A collection is analogous to table of relational database or collection of XML database. Documents are modeled as a data structure following the JSON format which actually store as BSON [BSO], a binary variant of JSON. In Mongodb, there are two principle that allow application to represent documents and their relationship: *reference* and *embedded documents*.

need to change following image/json according to xmark data

Mongodb data model pg4

...??

Reference Reference stores the relationships between data by including links and references from one document to another as in Figure 1(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 [Gao].

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 [Gao]. In addition to primary index, Mongodb supports various user defined indexes for field values of documents including single field index, multikey index, multidimensional index, geo-spatial index, text index and hash index. Single field, multidimensional and multikey index are organized using B-tree whereas geospatial index is implemented using quad trees.

To index a field that contains a array value, MongoDB provides special indexing called "Multikey". These *multikey* indexes allow MongoDB to return documents from queries using the value of array.

3.2.2 Query Model

Queries in MongoDB are expressed in a JSON like syntax and send to MongoDB as BSON objects by database drivers[Ore10]. MongoDB's query can be implemented over all documents inside collections, including embedded object and arrays. The Query model support following features:

1. Queries over documents, embedded subdocuments and arrays
2. Comparison operators
3. Conditional Operators
4. Logical Operators: AND and OR
5. Sorting
6. Group by
7. Aggregation per query

In addition to this MongoDB provide a features to for complex aggregation with the use Of MapReduce. The result of MapReduce either can be store as a collection or be removed after result has been return to client[Ore10].

3.3 Couchbase Server

Couchbase Server is NoSQL database that can be used both as a key-value store as well as document store system. As key-value store, it is able to store multiple data type such as strings, numbers, datetime, and booleans as well as arbitrary binary data. The key-value generally treated as opaque Binary Large Object(BLOB) and don't try to parse it. For document store, data need to be store in the valid JSON format.. 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.

document database system with flexible data model and easily scalable concept [Bro13]

Metadata For every value stored in database, Couchbase Server generate meta information that is associated with the document [OR14].

Expiration The Time to Live(TTL) also named expiration time is the life time of the document. By default it is 0 which indicates it will never expire, also can be set as Unix epoch time after which document is removed. e.g. The maximum number of seconds you can specify are the seconds in a month, namely 2592000(30 x 24 x 60 x 60). Couchbase Server will remove the item after given number of second it stores.

Check-and-Set(CAS) The CAS value is 64-bit integer that is updated by server when associated item is modified and is a method of optimistic locking [HD91]. It enables to update information only if unique identifier matches the identifier of the document that need to be update. CAS is used to manage concurrency when multiple client attempts to update the same document simultaneously.

Flags Flags are as 32 bit integer and are set of SDK specific use for SDK specific need. For example, format in which data is serialize or data type of the object being stored.

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 **??**.

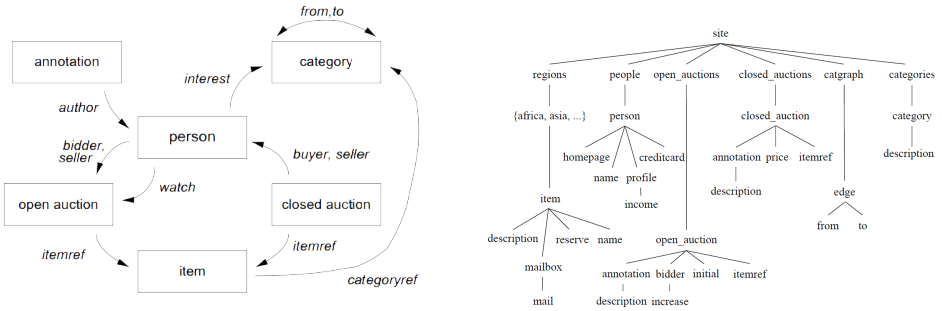
explain simple or complex

Document design

Bucket and vBucket

3.4 Rethinkdb

3.5 Summary



(a) Reference in *XMark*

(b) Reference in *XMark* dataset tree [SWK+02]

Figure 2: XMark data tree and reference

4 Performance/Experiments

4.1 XMark

The XML benchmarking project XMARK [SWK+02] is one of the most popular and most commonly used XML benchmarking project to date. It provides a small executable tool called *xmlgen* that allows for the creation of a synthetic XML dataset based on a fixed schema describing an Internet auctions database. Its generator can be used to build a single record with a large, hierarchic XML tree structure. A factor can be specified to scale the generated data, ranging from a few kilobytes to any arbitrary size limited by the capacity of the system. The textual part of resulting XML document is 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. 2(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. 2(b), `items` are the objects for sale or already sold. Each `item` caries a unique identifier and 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 a parent of an 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, amount of price etc. `people` are the information about `person` that are connected to buyer/seller of open_auctions, closed_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 [SWK+02].

figure need to create

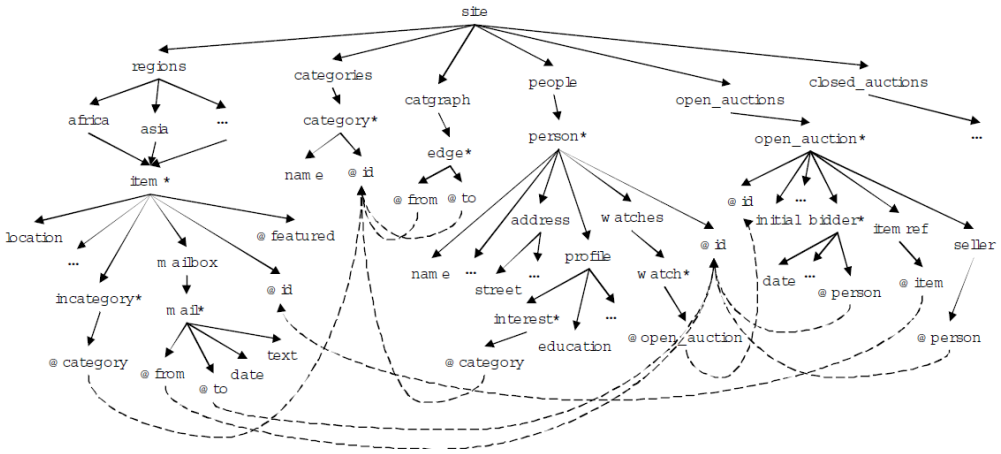


Figure 3: XMark ER-Diagram. Nodes, solid arrows, and dashed arrows represent schema elements (or attributes, with prefix '@'), structural links, and value links, respectively. Elements with suffix '*' are of SetOf type[YJ06]

4.1.2 Queries

The XMark project contains 20 XQuery Queries [SWK+02] which focus on various aspect of language such as agrression, reference, ordering, wildcard expressions, joins, user defined functions etc.[Mly]

change the style of queries and complete the queries

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 [WPFY03]. However, as already 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 substructure without losing the overall data and create index for each of them. Beside this, 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 databases separately.

???

The generalized concept of XMark data to NoSQL database is given here, but it might be slightly different each of them. 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 **type** is added to represent the value of the parent. for example, in case of person collection *type* will be *people*. This key value set will be also the part of document as shown in Code 2. There is one exceptional case for *items* which has **regions** as grandparent and name of different regions like *asia*, *europe* etc. as parent, the *type* 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        "type":"items",
2        "regions":"asia"
```

numbers

Code 2: General NoSQL data representation of XMARk data

```
1        {
2                "id": "person0",
3                "type": "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                       }],
14                       "education": "Graduate School",
```

```
15                              "business": "No"
16                      }
17              }
```

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>
```

4.3.1 MongoDB

4.3.1.1 XMARK in MongoDB

MongoDB's collections have 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. 3. 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 data model of in 4.3 slightly change. Each *type* represented as a collection. So we will have six collections and type is already represented by the collections. we don't need key/value of type 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

```
{
        "_id": "person0",
        "name": "Kasidit Treweek",
        "emailaddress": "mailto:Treweek@cohera.com",
        "phone": "+0 (645) 43954155",
        "homepage": "http://www.cohera.com/~Treweek",
        "creditcard": "9941 9701 2489 4716",
        "profile": {
                "income": 20186.59,
                "interest": [{
                        "category": "category251"
                }],
                "education": "Graduate School",
                "business": "No"
        }
}
```

4.3.1.2 Queries

coming soon ...

4.3.2 Couchbase Server

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

[BPSM+08] 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.

[Bro13] Martin C Brown. *Developing with Couchbase Server.* " O'Reilly Media, Inc.", 2013.

[BSO] BSON. Bson specification.

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

[HD91] Ugur Halici and Asuman Dogac. An optimistic locking technique for concurrency control in distributed databases. *Software Engineering, IEEE Transactions on*, 17(7):712–724, 1991.

[HJ11] Robin Hecht and S Jablonski. Nosql evaluation. In *International Conference on Cloud and Service Computing*, 2011.

[Lee11] David Lee. Jxon: an architecture for schema and annotation driven json/xml bidirectional transformations. In *Proceedings of Balisage: The Markup Conference*, 2011.

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

[OR14] David Ostrovsky and Yaniv Rodenski. *Pro Couchbase Server.* Apress, 2014.

[Ore10] Kai Orend. Analysis and classification of nosql databases and evaluation of their ability to replace an object-relational persistence layer. *Architecture. Citeseer*, page 100, 2010.

[Sen15] Mark Senn. *NOSQL Databases. @ONLINE Nosql-database.org*, 2015 (accessed January 10, 2015).

[SWK+02] 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.

[WPFY03] 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.

[YJ06] 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.

List of Figures

1 Mongodb document structure 4
2 XMark data tree and reference 7
3 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[YJ06] 8

List of Tables

Listings

1 *doctype* and *regions* for item which has region name *asia* 9
2 General NoSQL data representation of XMARk data 9
3 XMARK data with of *person0* 10
4 Mongodb data representation of XMARk data 11