

# XML and NoSQL DBMS: Migration and Benchmarking

Author:  
Prakash Thapa

University of Konstanz

March 3, 2015

## Abstract

As massive amount of dynamic data grow, database vendors search for the alternative of classical database management system such as RDBMS. XML and NoSQL databases are two non-relational database management system growing since the popularity of Big Data. This paper focus on the comparative analysis of these two new database system based on use cases and existing solutions. On the first part, our focus will be data migration from one system to another. We will also discuss the performance of both systems based on standard queries.

.....

## Zusammenfassung(German Abstract)

XML und NoSQL

## Acknowledgments

.....

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Contribution . . . . .	1
1.3	Scope of Thesis . . . . .	1
1.4	Overview . . . . .	1
<b>2</b>	<b>NoSQL and XML Databases</b>	<b>3</b>
2.1	XML Database . . . . .	3
2.1.1	BaseX . . . . .	3
2.2	NoSQL Database . . . . .	3
2.2.1	MongoDB . . . . .	4
	Data Model . . . . .	4
	Embedded . . . . .	4
	Reference . . . . .	4
	Documents . . . . .	4
	Data-types for Document Fields . . . . .	4
	Indexing . . . . .	5
	Query Model . . . . .	5
	System Architecture . . . . .	6
2.2.2	Couchbase Server . . . . .	6
	Metadata . . . . .	7
	Document key . . . . .	7
	Document design . . . . .	7
	Bucket and vBucket . . . . .	7
2.2.3	RethinkDB . . . . .	9
	Data Model . . . . .	9
	Query Model . . . . .	9
	Indexing . . . . .	9
2.3	Summary . . . . .	9
	. . . . .	9
<b>3</b>	<b>Semi-structured Data: XML and JSON</b>	<b>10</b>
3.1	Problem to translate from XML to JSON . . . . .	10
	Document node . . . . .	10
	Anonymous values . . . . .	10
	Arrays . . . . .	10
	Identifiers . . . . .	10
	Attributes . . . . .	11
	Namespaces . . . . .	11
	Others . . . . .	11
3.2	Mapping . . . . .	11
3.3	Migration from XML to JSON . . . . .	11
	Step 1 . . . . .	11
	Step 3 . . . . .	13

<b>4</b>	<b>Performance/Experiments</b>	<b>14</b>
4.1	XMark . . . . .	14
4.1.1	Dataset . . . . .	14
4.1.2	Queries . . . . .	16
4.2	Evaluation of Test devices . . . . .	17
4.3	XMark data into NoSQL Database . . . . .	17
4.3.1	XMARK in MongoDB . . . . .	19
4.3.1.1	Queries	19
4.3.2	Couchbase Server . . . . .	19
	Document design . . . . .	19
4.3.2.1	Queries	19
4.3.3	Rethinkdb . . . . .	19
	Data Model . . . . .	20
	Indexing . . . . .	20
4.4	Benchmarking . . . . .	21
4.5	Summary . . . . .	21
<b>5</b>	<b>Discussion</b>	<b>22</b>
<b>6</b>	<b>Conclusion</b>	<b>22</b>
<b>7</b>	<b>Future Work</b>	<b>23</b>

# 1 Introduction

## 1.1 Motivation

As the digital world growing very fast, the massive amount of data collected today in the field varying from business to scientific research, is becoming complex for storage, query and providing mechanism for failover. As the data collection grows so largely, the traditional data management tools e.g. relational database management systems(RDBMS) are struggling to handle such data effectively. The pre-design rigid schema structure of RDBMS made more complex for variety of data. High data velocity where massive read and write operations possible from different geographic location, storage of structured, semi-unstructured and unstructured data and gigantic volume, together sometimes refer as *Big Data* became a global phenomena which added more complexities on RDMS.

New database technologies NoSQL and XML Databases came in existence to overcome above mentioned problem of RDBMS. These systems are not just able to solve the issues of existing system, but also rise as alternative in their way. Both these new technologies are focus on varieties of data in large volume, dynamic schema and scalability.

There are some research analysis between NoSQL and RDBMS. Cattell (2011) analyze similarities between scalable SQL and NoSQL databases, Hadjigeorgiou et al. (2013) compare the performance between MySQL cluster as RDBMS and Mongoddb as NoSQL Database. There are also some research related to RDBMS with XML or XML Database Jiang et al. (2002) Shanmugasundaram et al. (1999). But there are not much more research in the field of NoSQL and XML database together. So this paper focus on these two new database systems. Migration of Data from XML database to NoSQL as well as performance of both system based on standard query will be examine in this paper. .... ..

## 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. The performance comparison of these two systems will be based XMARK dataset and its 20 standard queries (Schmidt et al., 2002). These 20 queries of XMARK dataset are being translated to each of NoSQL database for performance analysis.

## 1.3 Scope of Thesis

..... ..

## 1.4 Overview

This thesis is divided into three main sections. Section ?? defines the techniques and necessary algorithms to convert XML to JSON. In Section ?? 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 NoSQL and XML Databases

### 2.1 XML Database

XML Database or sometime written also Nativ XML Databases(NXDs) are document store databases that store XML data and all component of XML data based on a specified model called XDM. XML document is the fundamental unit of any of storage, so each document should be in a valid XML format. XQuery 1.0 and XQuery 3.0 are standards query languages recommended by W3C for NXDs. XQuery also includes XPath as sub-language. In addition to XQuery and XPath, XML databases also support XSLT, a method of transferring XML to other documents like HTML, plain text XSL Formatting Objects.

#### 2.1.1 BaseX

Basex is a Nativ XML database management system and XQuery Processor. BaseX uses tabular representation of XML data tree to store XML document (Grün, 2010). In this paper BaseX is represented as XML database.

### 2.2 NoSQL Database

NoSQL database "Not Only SQL" is distributed data management system for large and variable data. Three main focus of any NoSQL database are scalability, performance and availability (Hecht and Jablonski, 2011). Based on the Data model and design architecture, NoSQL databases are categories in 4 group:

- **Key/Value store** The Data representation of key-value stores are based on attribute pair, the data model is expressed as collection of  $\langle key, value \rangle$  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 databases** are based on semi-structured data model, where unique key store a values that contains a tree like structure called *documents*. Document store enclose key-value pair in JSON or in JSON like data format (Hecht and Jablonski, 2011). In document store database, data can be access by using key or specific value. Some of the examples of document store databases are MongoDB, Couchbase, RethinkDB, CouchDB etc.
- **Column family store** also known as wide-column database stores data tables as section of a column. Cassandra, BigTable, HBase are categories as wide column database.
- **Graph database** are based on graph theory. In these databases data are represented as graph, 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.

As there are many categories and varieties of database system, our focus this paper is limited to document store databases of NoSQL databases. More specifically, MongoDB, Couchbase and RethinkDB will represent NoSQL database in this paper. Therefore in following sections, these three NoSQL databases will be represented the word "NoSQL".

### 2.2.1 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++.

**Data Model** MongoDB database resides on MongoDB server that can host more than one database which are independent and store separately by MongoDB Server. A database contains one or more collection that contains documents. As MongoDB is schema-free system, Collection may 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 the application. Documents are modeled as a data structure following the JSON format which actually store as BSON BSON, a binary variant of JSON supports additional data types like ObjectId, timestamp, datetime etc. In MongoDB, there are two principle that allow application to represent documents and their relationship: *reference* and *embedded documents*.

**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-documents in the in the form of Array or/and Object Gao.

...??

**Reference** MongoDB doesn't support joins, so generally, related data are stored in a single document to remove the need for joins. In some cases, it makes sense to store related data in separate documents, typically in different collection or even database. Reference stores the relationships between data by including links and references from one document to another as in Figure 1a. These references can be created in two way.

- Manual reference The application handle the relationship between the related data in manual reference. The `_id` field of one document is referenced to another document.
- Database reference(`DBRefs`)

explain

The application can resolve these reference to access the related data. The normalize data model

MongoDB  
data  
model  
pg4

**Documents** A document is a abstraction and storable unit in MongoDB. It is the data structure represented in the form of JSON which is actually store in BSON.

**Data-types for Document Fields** MongoDB support following datatypes

- scalar types: **boolean**, **integer**, **double**
- character sequence type: **string**(encoded in UTF-8), **regular expression**
- **Object**: All BSON-Objects
- **ObjectId** : It is BSON type 12 byte long binary value to uniquely identify the document. Documents are stored in a collection with `_id` field which also act as primary key with the value of ObjectId. The object id datatype is composed of the following components:



```

{
    title : " MongoDB ",
    last_editor : "172.5.123.91" ,
    last_modified : new Date ("9/01/2015") ,
    body : " MongoDB is a..." ,
    categories : [ " Database ", " NoSQL ", "
        Document Database " ] ,
    reviewed : false
}

```

Code 1: MongoDB sample document

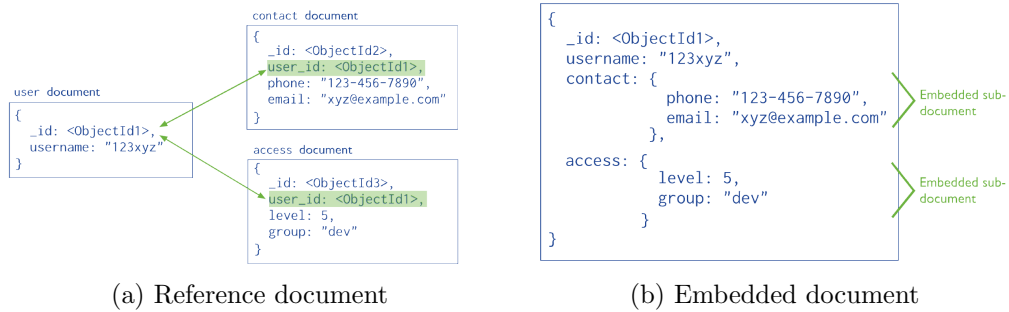


Figure 1: MongoDB document structure

- timestamp in seconds since epoch (first 4 bytes)
- id of the machine assigning the object id value (next 3 bytes)
- id of the MongoDB process (next 2 bytes)
- counter (last 3 bytes)

- null
- array
- date

**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 an array value, MongoDB provides special indexing called "Multikey". These *multikey* indexes allow MongoDB to return documents from queries using the value of array.

**Query Model** Queries in MongoDB are expressed in a JSON like syntax and send to MongoDB as BSON objects by database drivers Orend (2010). MongoDB's query can be implemented over all documents inside collections, including embedded object and arrays. The Query model support following features:

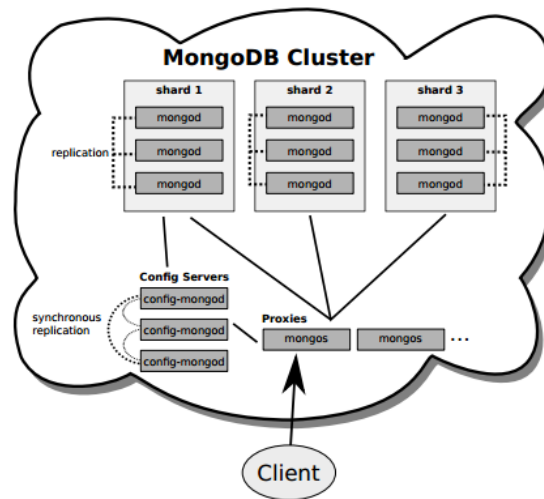


Figure 2: MongoDB Clusters

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 clientOrend (2010).

**System Architecture** MongoDB can be run in two mode. In stand-alone mode a single *mongod* demon is running in a single node without any distribution. The other mode is sharded mode, where various services of MongoDB are distributed to several nodes.

### 2.2.2 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 Mongoddb 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 Brown (2013)

**Metadata** For every value stored in database, Couchbase Server generate meta information that is associated with the document (Ostrovsky and Rodenski, 2014, p. 26).

**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 Halici and Dogac (1991). 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 information are stored 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. Unlike 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 Ostrovsky and Rodenski (2014).

explain  
simple  
or com-  
plex

## Document design

**Bucket and vBucket** Couchbase Server uses data bucket as a logical container of information. It provides a logical grouping of physical resources with in a cluster Lichtenberg and Ward (2013). A bucket is equivalent to a database in MongoDB or RDBMS. Unlike MongoDB, the Couchbase Server don't have concept of collections. Document don't have fixed schema, multiple documents with different schema can be stored in same bucket. One or more attributes are added to differentiate the various objects stored in a bucket and create indexes on them.

Each bucket is split into 1024 logical partition called vBuckets. A vBucket is treated as owner of subset of key a Couchbase Server. Every key belongs to a vBucket.

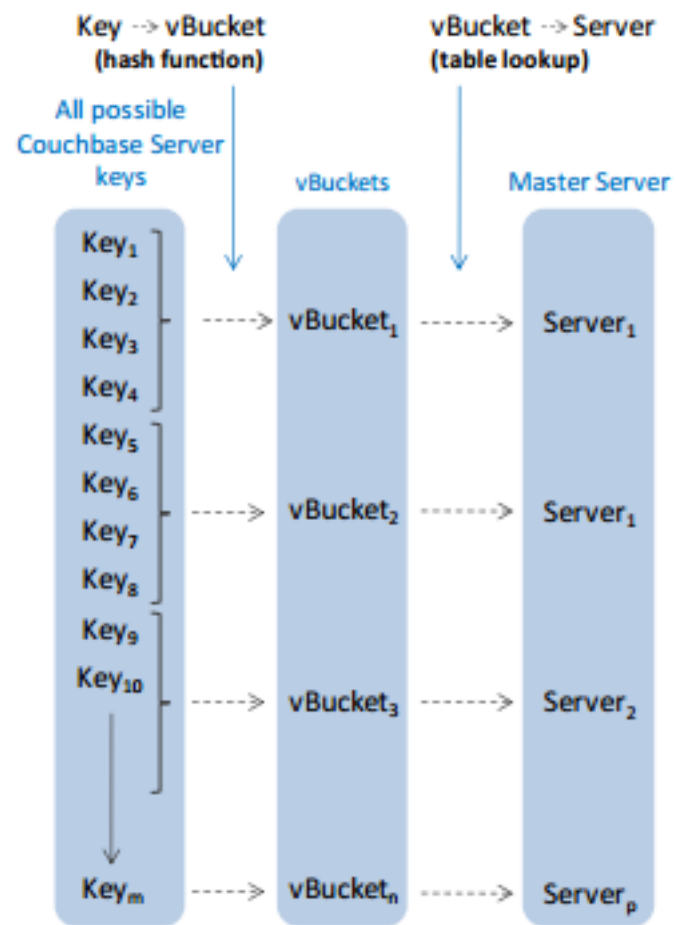


Figure 3: Couchbase Server vBucket

### 2.2.3 RethinkDB

RethinkDB Rethinkdb (2015 (accessed January 12, 2015) 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** Rethinkdb stores JSON documents with Binary on disk serialization. The data types supported by RethinkDB is same as JSON. There are two ways to model relationship between the documents in RethinkDB.

1. In *Embedded arrays*, the related sub-documents are inserted with specific key in side a document as in `??`. The advantages of using embedded arrays are:
  - The Queries are tend to be simpler.
  - If a dataset is large and don't fit in RAM, data is loaded faster from disk as compare to tables.


Disadvantages of embedded arrays are:

- Everything should be loaded in memory for any operation.
2. In *multiple tables* approach, document with similar schema are stored in tables, like in relational database, connect by reference key to another table. Unlike embedded document, operation of a table don't required to load data from reference table. Table has also some disadvantages like liking between table is more complicated.

**Query Model** RethinkDB allows embedding JavaScript expressions anywhere as part of the query language. RethinkDB uses a pool of outofprocess V8 execution engines for isolation.

**Indexing** RethinkDB supports primary key, compound, secondary, and arbitrarily computed indexes stored as Btrees. Every RethinkDB query, including update operations, uses one and only one index.

## 2.3 Summary

All document s  NoSQL database store data in the form of JSON or JSON like format and XML is the basic unit of storage for XML database. for data migration, it is necessary to understand the properties of XML and JSON Data format.

### 3 Semi-structured Data: 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 user-defined abstract data types are derived (Schmidt et al., 2002). 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`.

#### 3.1 Problem to translate from XML to 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 (Lee, 2011).

**Document node** Each XML document have a document node. In contrast, the JSON does not have one. In practice, XML node is implicitly created in the model and does not have a textual representation.

**Anonymous values** JSON support the anonymous values. They only acquire names by being referenced in an object. for example 1 is valid JSON. An XML valid document should have a root element followed by non-mandatory child text or attributes.

Table 1: Anonymous values of JSON in XML

JSON		XML
numbers		<code>&lt;root&gt;Hello World&lt;/root&gt;</code>
<code>["Hello World"]</code>	or	<code>&lt;root value="Hello World"&gt;</code>

**Arrays** Arrays are native data types of JSON which does not exists in XML. There is no direct markup for array of Object from JSON to XML.

**Identifiers** XML is much more restrictive for identifiers as compare to JSON which allows any string to be an identifier. Translating from XML to JSON do not cause problem but in reverse case might lead to invalid XML. For example "Hello World" is a valid identifier in JSON but not valid attribute or element in XML as there exists a whitespace between two words.

**Attributes** JSON does not have the concept or representation for XML's attributes. When mapping from XML to JSON, attributes are translated to name object member along with other child elements. This information will be loss when mapping from JSON to XML.

**Namespaces** XML supports the concept of namespaces to provide uniqueness in element and attributes in XML document. Namespaces do not exists in JSON. Mapping QNames in XML with namespaces to JSON can lead ambiguous and duplicate names.

**Others** There are also some problem like processing instructions and comments which XML supports but not JSON. Other issues for example character set and encoding are not easily exchangeable.

## 3.2 Mapping

XML and JSON have different data types. Compare to JSON, XML more flexible data types. In Table 2 given types of data for XML and relevant JSON type.

Table 2: Translation of simple XML data types into JSON

XML Schema type	JSON type definition
<code>xs:string</code>	<pre>{   "type": "string" }</pre>
<code>xs:boolean</code>	<pre>{   "type": "boolean" }</pre>
<code>xs:float</code> <code>xs:double</code> <code>xs:decimal</code> <code>xs:integer</code> (All Other Numbers)	<pre>{   "type": "number" }</pre>
(Remaining all others)	<pre>{   "type": "string" }</pre>

## 3.3 Migration from XML to JSON

It takes different stages to convert XML data to JSON.

- Step1. XML to JSON friendly XML: Convert standard XML data to JSON friendly XML.
- Step2. Type of XML to JSON type: Define the type of of XML element whether it is array type or Object type or other scaler type.
- Step3. Final steps: Map XML to JSON object.

**Step 1** At first, all attributes of XML document are moved as ordered(first) child element of parent. There might more than one attributes which are inserted one

after other. XML node can contain both text node and element node as child, but JSON can have only key value pair. so each of text node which has also element node as siblings moved to "childtext" (name of the element) node. At the end, of first step, if a child node has another element as child node, this represent Object type of JSON. The type Object is marked each of those element. finally it is necessary to identify if a siblings of an element node(S) has same element name, If this condition exists, it is represented as the array in JSON. All the child nodes of all siblings are moved inside S node and if S is Object type it is replace with array type if not then array is added. In this algorithm, the loop of one action affect other, so they are looped independently.

---

**Algorithm 1:** Pseudocode to convert normal XML to JSON friendly XML

---

```

Initialize  $D = "XMLdocument"$ ;
for all descendant-or-self node of  $D$ ,  $X$  which has attributes  $A$  do
    move all attributes  $A$  to ordered child element of  $X$ 
end for
for all descendant-or-self node of  $D, X$  do
    if  $X$  contains Text node and element node Both then
        create new element "childtext"
        move text node to "childtext" element
    end if
end for
for all descendant node of  $D, X$  do
    for all child element  $C$  in  $X$  do
        if  $C$  has siblings  $S$  with same name then
            convert  $C$  as Array type
            move child of  $S$  into  $C$  as  $<_>$  element
        end if
        if  $C$  has child element then
            convert  $C$  as Object type
        end if
    end for
end for

```

---



---

**Algorithm 2:** convert data type of XML to JSON data type(Step 2)

---

```

Initialize  $D = "XMLdocument"$ ;
for all descendant-or-self node of  $D$  as  $C$  do
    if  $C$  has no attribute "type" then
        get content of  $C$ 
        identify content type according to Table 2. and add attribute to  $C$  "type"
    end if
end for

```

---

Table 3: XML to JSON friendly XML(STEP 1 and 2)

XML (Before Algorithm 1 and 2 )	AFTER
---------------------------------	-------



<pre> &lt;info&gt;   &lt;name&gt;     &lt;f&gt;a&lt;/f&gt;   &lt;/name&gt;   &lt;age&gt;24&lt;/age&gt;   &lt;ismarried&gt;false&lt;/ismarried&gt;   &lt;city name="Armonk" /&gt;   &lt;state&gt;NY&lt;/state&gt;   &lt;contact&gt;     home     &lt;p&gt;993-330&lt;/p&gt;     &lt;p&gt;993-331&lt;/p&gt;   &lt;/contact&gt; &lt;/info&gt; </pre>	<pre> &lt;info type="object"&gt;   &lt;name type="object"&gt;     &lt;f type="string"&gt;a&lt;/f&gt;   &lt;/name&gt;   &lt;age type="number"&gt;24&lt;/age&gt;   &lt;ismarried type="boolean"&gt;false&lt;/ismarried&gt;   &lt;city type="object"&gt;     &lt;name type="string"&gt;Armonk&lt;/name&gt;   &lt;/city&gt;   &lt;state type="string"&gt;NY&lt;/state&gt;   &lt;contact type="object"&gt;     &lt;childtext&gt;home&lt;/childtext&gt;     &lt;p type="array"&gt;       &lt;- type="number"&gt;993330&lt;/-&gt;       &lt;- type="string"&gt;993-331&lt;/-&gt;     &lt;/p&gt;   &lt;/contact&gt; &lt;/info&gt; </pre>
---	--

**Step 3** After Step 1. and 2., a complete JSON friendly XML is generated. All XML object and array are mapped to *<key, value>* pair of JSON and their respective data types.

Table 4: XML to JSON (STEP 3.)

XML(After step 2)	JSON
<pre> &lt;info type="object"&gt;   &lt;name type="object"&gt;     &lt;f type="string"&gt;a&lt;/f&gt;   &lt;/name&gt;   &lt;age type="number"&gt;24&lt;/age&gt;   &lt;ismarried type="boolean"&gt;false&lt;/ismarried&gt;   &lt;city type="object"&gt;     &lt;name type="string"&gt;Armonk&lt;/name&gt;   &lt;/city&gt;   &lt;state type="string"&gt;NY&lt;/state&gt;   &lt;contact type="object"&gt;     &lt;childtext type="string"&gt;home&lt;/childtext&gt;     &lt;p type="array"&gt;       &lt;- type="number"&gt;993330&lt;/-&gt;       &lt;- type="string"&gt;993-331&lt;/-&gt;     &lt;/p&gt;   &lt;/contact&gt; &lt;/info&gt; </pre>	<pre> numbers {   "info":{     "name":{       "f":"a"     },     "age":24,     "ismarried":false,     "city":{       "name":"Armonk"     },     "state":"NY",     "contact":{       "childtext":"home",       "p":[         993330,         "993-331"       ]     }   } } </pre>

## 4 Performance/Experiments

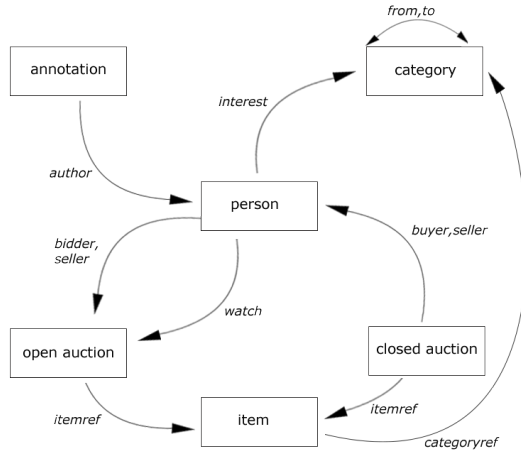
### 4.1 XMark

The XML benchmarking project XMARK (Schmidt et al., 2002) 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.

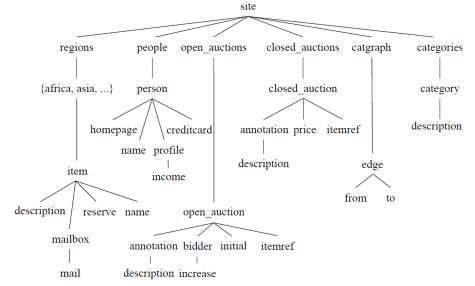
#### 4.1.1 Dataset

The main entities of XMark data come in two groups. `person`, `open_auction`, `closed_auction`, `item` and `category` are categories as first group. The second group entities `annotation` and `description` are natural language text and document-centric element structure. The relationship between the entities in group one are expressed in reference and second group entities are embedded into subtree of group one’s entities. As shown in Fig. 4b, Xmark dataset has following properties:

- **people** are the information about **person** that are connected to buyer and seller of open-auctions, closed-auctions etc. **categories** are implemented to classify items which has a name and description. Each person has an unique id to reference to another entities like open and closed auctions.
- **items** are the objects for sale or already sold. Each **item** carries a unique identifier and has properties like name, payment information(credit card, money order), **description**, a reference to the seller etc., all are encoded as element. Each item belongs to the world’s regions **africa**, **asia**, **australia**, **europa**, **namerica** and **samerica**. as a parent of an item element.
- **open\_auctions** are currently in progress auctions which contain bid history(increase/decrease over time) with reference to bidder and seller, current bid, the time interval within which the bid is accepted, the status of transaction, a reference to the item which is being sold etc.
- **closed\_auctions** are the auctions that are successfully completed bidding. They have properties like buyer and seller information which are reference to *person*, a reference to an item that is sold, amount of price, quantity of items sold, date of transaction, type of transaction.
- *categories* are used to classify the items. Each category has an unique identifier which is used to reference in item, a name and a description.
- A **catgraph** links categories into a network. The full semantic of XMark dataset can be found in Schmidt et al. (2002).



(a) Reference in *XMark*



(b) Reference in *XMark* dataset tree (Schmidt et al., 2002)

Figure 4: XMark data tree and reference

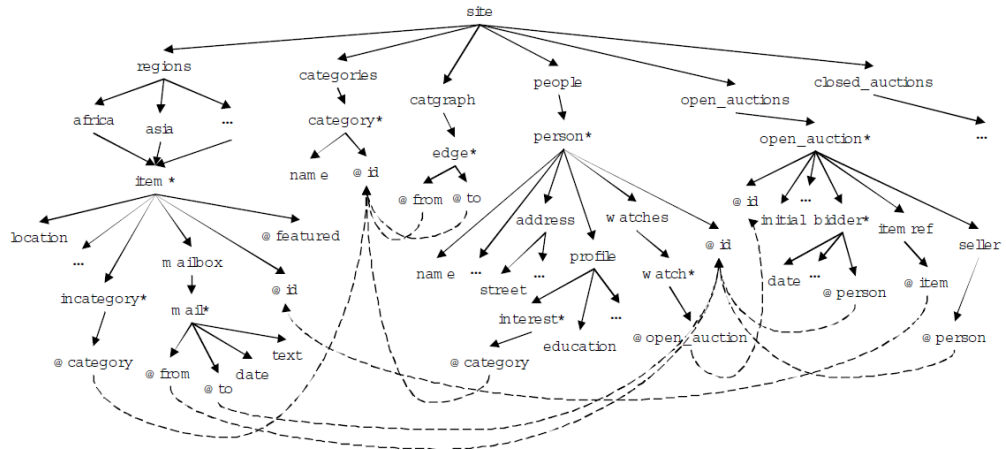


Figure 5: XMark ER-Diagram. Nodes, solid arrows, and dashed arrows represent schema elements (or attributes, with prefix '@'), structural links, and value links, respectively. Elements with suffix '\*' are of SetOf type (Yu and Jagadish, 2006)

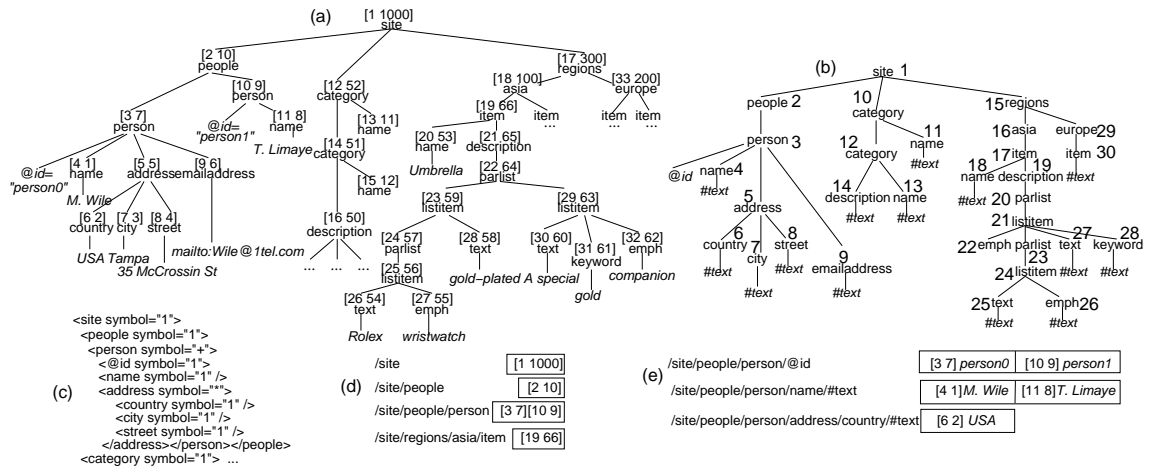


Figure 6: XMark document snippet, its path summary, and some path-partitioned storage structures.

#### 4.1.2 Queries

The XMark project contains 20 XQuery Queries which focus on various aspect of language such as aggression, reference, ordering, wildcard expressions, joins, user defined functions etc.(Mlynková, 2008). In this section these queries has been reprinted.

change the style of queries and complete the queries

## 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 Wang et al. (2003). However, as already mentioned in 4.1, each subtree in schema, *items*, *people*, *open\_auctions*, *closed\_auctions*, *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. 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 . 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*.

```
{
    "type": "items",
    "regions": "asia"
}
```

Code 17: *type* and *regions* for item which has region name *asia*

```
<people>
  <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>
</people>
```

Code 18: XMARK data with of *person0* in XML format

```
{
    "id": "person0",
```

ref:code:jbsn-  
nosql-  
person0

```

    "type": "people",
    "name": "Kasidit Treweek",
    "emailaddress": "mailto:Treweek@cohera.com",
    "phone": "+0 (645) 43954155",
    "homepage": "http://www.cohera.com/~Treweek",
    "creditcard": "9941 970124894716",
    "profile": {
        "income": 20186.59,
        "interest": [{
            "category": "category251"
        }],
        "education": "Graduate School",
        "business": "No"
    }
}

```

Code 19: XMARK data with of *person0* in JSON format

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 see in Fig. 5, 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.

```
{
  "_id": "person0",
  "name": "Kasidit Treweek",
  "emailaddress": "mailto:Treweek@cohera.com",
  "phone": "+0 (645) 43954155",
  "homepage": "http://www.cohera.com/~Treweek",
  "creditcard": "9941 970124894716",
  "profile": {
    "income": 20186.59,
    "interest": [{
      "category": "category251"
    }],
    "education": "Graduate School",
    "business": "No"
  }
}
```

Code 20: Mongoddb data representation of XMARK data

#### 4.3.1.1 Queries

#### 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 *type*. 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

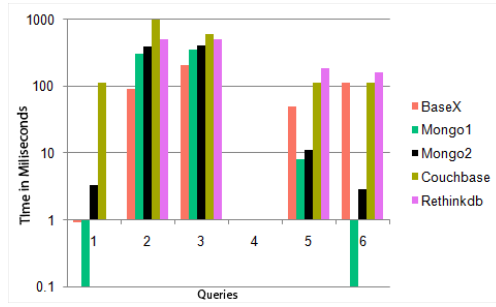
RethinkDB Rethinkdb (2015 (accessed January 12, 2015) is distributed database system to store JSON documents that uses efficient query languages named ReQL which automatically parallelize queries in multiple machines. ReQL is based on three main principle: it is completely embedded with programming language,

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.

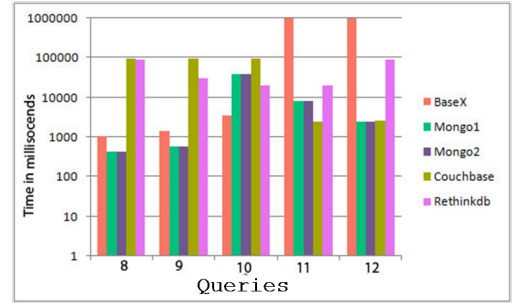
**Data Model** The relationships between documents can be can be created by embedded arrays of documents and linking document stores in multiple tables similar to MongoDB in 4.3.1. Rethinkdb stores JSON documents with binary on disk serialization. RethinkDB implicitly support JSON data types: **object**, **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.

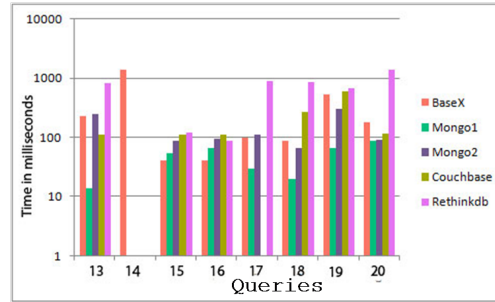




(a) XMark Queries No. 1 to 7



(b) Join Queries (No. 8 to 12)



(c) Queries 13 to 20

Figure 7: XMark Queries time in XML and NoSQL databases(Mongo1: Mongoddb shell, mongo2:through application)

#### 4.4 Benchmarking

#### 4.5 Summary

**5 Discussion**

**6 Conclusion**

storing in the memory

## References

- Martin C Brown. *Developing with Couchbase Server*. ” O’Reilly Media, Inc.”, 2013.
- BSON. Bson specification. URL <http://bsonspec.org/>.
- Rick Cattell. Scalable sql and nosql data stores. *ACM SIGMOD Record*, 39(4): 12–27, 2011.
- Xiaoming Gao. Investigation and comparison of distributed nosql database systems.
- Christian Grün. BaseX – The XML Database for Processing, Querying and Visualizing large XML data. <http://basex.org>, October 2010.
- Christoforos Hadjigeorgiou et al. RDBMS vs NoSQL: Performance and scaling comparison. *The University of Edinburgh*, 2013.
- 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.
- Robin Hecht and S Jablonski. Nosql evaluation. In *International Conference on Cloud and Service Computing*, 2011.
- Haifeng Jiang, Hongjun Lu, Wei Wang, and Jeffrey Xu Yu. XParent: An efficient RDBMS-based XML database system. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 335–336. IEEE, 2002.
- Susan Malaika Michael Maximilien Rich Salz Jerome Simeon John Boyer, Sandy Gao. Experiences with json and xml transformations. IBM, 2011.
- David Lee. Jxon: an architecture for schema and annotation driven json/xml bidirectional transformations. In *Proceedings of Balisage: The Markup Conference*, 2011.
- Kyle Lichtenberg and Miles Ward. Nosql database in the cloud: Couchbase server 2.0 on aws. 2013.
- Irena Mlýnková. Xml benchmarking: Limitations and opportunities. Technical report, Technical Report, Department of Software Engineering, Charles University, Czech Republic, 2008.
- 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.
- David Ostrovsky and Yaniv Rodenski. *Pro Couchbase Server*. Apress, 2014.
- Rethinkdb. *An open-source distributed database built with love - RethinkDB*, 2015 (accessed January 12, 2015). URL <http://www.rethinkdb.com/>.
- 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.

- Mark Senn. *NOSQL Databases*. @ONLINE *Nosql-database.org*, 2015 (accessed January 10, 2015). URL <http://www.nosql-database.org/>.
- Jayavel Shanmugasundaram, Kristin Tufte, Chun Zhang, Gang He, David J DeWitt, and Jeffrey F Naughton. Relational databases for querying XML documents: Limitations and opportunities. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 302–314. Morgan Kaufmann Publishers Inc., 1999.
- 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.
- 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.

<b>List</b>	<b>of</b>	<b>Figures</b>
1	MongoDB document structure . . . . .	5
2	MongoDB Clusters . . . . .	6
3	Couchbase Server vBucket . . . . .	8
4	XMark data tree and reference . . . . .	15
5	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(Yu and Jagadish, 2006) . . . . .	15
6	XMark document snippet, its path summary, and some path-partitioned storage structures. . . . .	16
7	XMark Queries time in XML and NoSQL databases(Mongo1: MongoDB shell, mongo2:through application) . . . . .	21

# List of Tables

1	Anonymous values of JSON in XML . . . . .	10
2	Translation of simple XML data types into JSON . . . . .	11
3	XML to JSON friendly XML(STEP 1 and 2) . . . . .	12
4	XML to JSON (STEP 3.) . . . . .	13

## Listings

1	MongoDB sample document . . . . .	5
2	. . . . .	10
3	. . . . .	10
4	. . . . .	10
5	. . . . .	11
6	. . . . .	11
7	. . . . .	11
8	. . . . .	11
9	. . . . .	11
10	. . . . .	11
11	. . . . .	11
12	. . . . .	11
13	. . . . .	13
14	. . . . .	13
15	. . . . .	13
16	. . . . .	13
17	<i>type</i> and <i>regions</i> for item which has region name <i>asia</i> . . . . .	17
18	XMARK data with of <i>person0</i> in XML format . . . . .	17
19	XMARK data with of <i>person0</i> in JSON format . . . . .	17
20	Mongoddb data representation of XMARK data . . . . .	19