Simple XQuery Evaluation Engine on XML Database and Indexed Documents

Renjie Weng

Stevens Institute of Technology

rweng@stevens.edu

Abstract

This project is a coding project. It is a B/S system based on Java Servlet to build the CGI interface and using AJAX to implement the XQuery and XPath Evaluation on XML files. The Persistence Storage (PS) is depended on XML documents, which is organized by an open source XML Database, Sedna, which organize the XML by B-tree structure. This project also provides a method to compare the difference of XPath evaluation performance between original Data Graph and Indexed XML documents. The original Data Graph is queried on default Java XPath API, and the Indexed XML documents are produced by a next-generation XML processing solutions named Ximplware.

This paper introduces my project will following steps:

I introduce my motivation to do this project in Section 1; then describe the project with a general view in Section 2; In Section 3, I'll analyze some critical codes in my projects; And in Section 4, I demonstrate the performance of two methods of XPath Evaluation. In Section 5, I'll introduce some of my own view of why I think XML will be further developed and researched in the future; At last, I sited a more widely used XML Database -- BaseX -- than Sedna.

1. Motivation

Persistence Storage (PS) using plain text is greatly improved in these years. XML, as a fully structured plain text format, has been developed and researched for many decades. XML has the features of highly readable and distributable. So, using XML as plain text storage is one of the best alternatives to traditional SQL based PS in many environments such as Distributed Systems, Cloud Computing, and Search Engines.

And from my personal experience, I'd like to use XML as Persistence Storage and Web Service call. Since XML can be distributed conveniently as a signal file, and the highly structured feature provides rich information exchange through SOAP based or other kind of Web Service call. It is not only efficient but also scalable to program on network applications with a framework using XML interactions.

The XML related technologies like XPath and XQuery are also located in my interests. XML can be a useful programming tool. Most configuration files in projects or services are XML formatted. So that programmer can get the main idea of the project or service swiftly and modify the XML configuration file to fulfill his/her own requirement, such as service port, reference libraries.

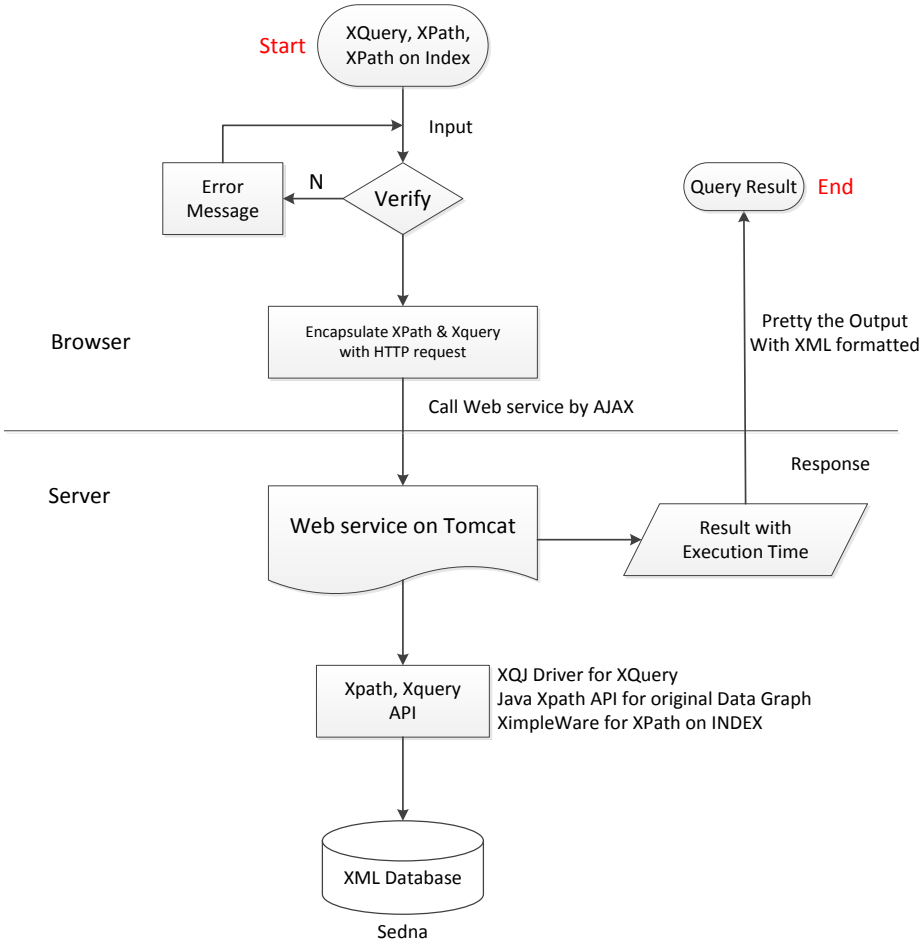
And the processing of structured files is also interesting to me. Even plain text may have regulations, such as string searching algorithms like Knuth-Morris-Pratt [1] and Boyer–Moore [2]. Then as highly structured document, XML must be more efficient and functional while processing. XPath, XQuery and XLST are the examples.

After I have read the paper “Exploiting Local Similarity for Indexing Paths in Graph-Structured Data” which introduced a processing method named A(k)-Index, I think it is important for people or other programmers to have a web-based system which could help them to test the performance of different XML query processing algorithms, as well as to exam the theories.

Therefore, I decided to build a simple XQuery Evaluation Engine, and supplies interface to do query evaluation on both original Data Graph as well as Indexed XML Documents.

2. Overview

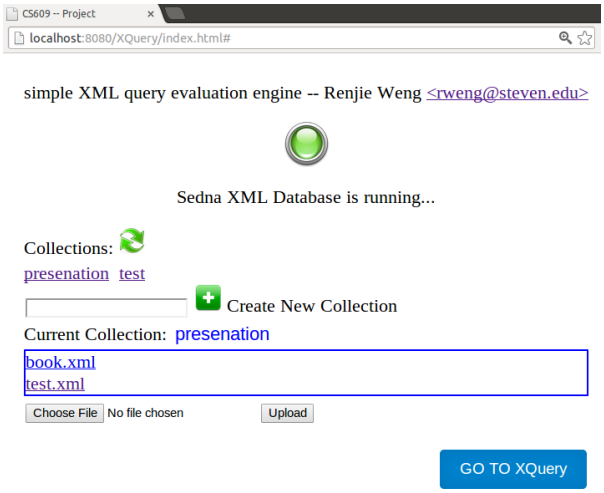
My project is building on Browser / Server architecture. The source code is accessible on Github: https://github.com/shohoku11wrj/XQuery-CGI-Engin .



My project is composed with two parts.

Part I. XML Documents Management Service:

This part includes Retrieve the State of Web service, Create new collections, List Documents in specified collection, Upload and Download documents.



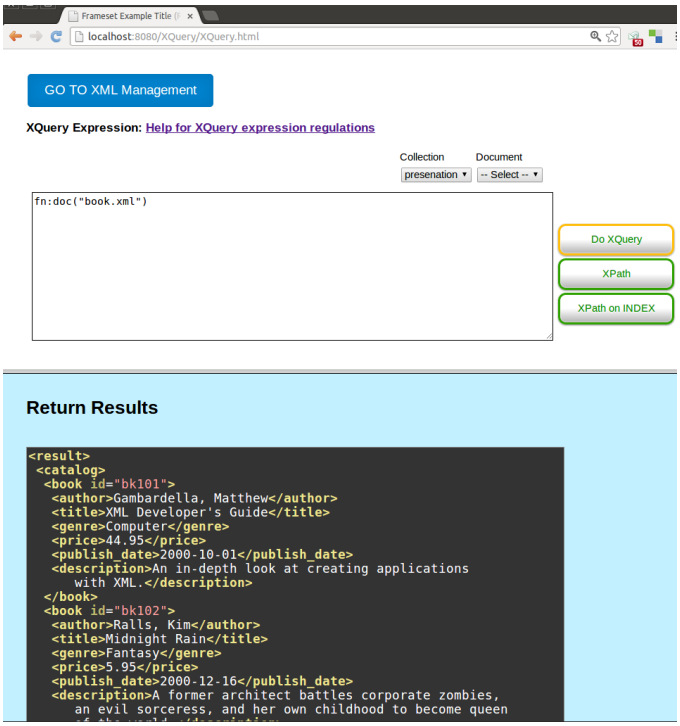
Part II. XQuery/ XPath Evaluate Service.

This part includes XQuery, XPath, and XPath on Index Evaluations. The query expressions should be verified at browser side at first before they are composed to HTTP requests;

Then the query requests are all sent with AJAX, which can remain the original query expressions without refreshed by page reloading;

When the server receives the request, it will execute the query expression according to three query types, and at the meantime, recording the evaluation time of querying, then response the result with execution time to the request;

At last, the browser will pretty the responses with XML formatted to clarify the execution time and query results.



There are also some important concepts and technologies I'd like to explain using in my project:

The Server side is implemented by Java Servlet as a Dynamic Web Site. It is running on Apache Tomcat 7.0.

The XML documents are managed by a XML Database. In my project, I used one named Sedna [1].

An XML database a data persistence software system that allows data to be stored in XML format. These data can then be queried, exported and serialized into the desired format. XML databases are usually associated with document-oriented databases.

There are two major classes of XML database exist:

1. XML-enabled: these may either map XML to traditional database structures (such as a relational database), accepting XML as input and rendering XML as output, or more recently support native XML types within the traditional database. This term implies that the database processes the XML itself (as opposed to relying on middleware).

2. Native XML (NXD): the internal model of such databases depends on XML and uses XML documents as the fundamental unit of storage, which are, however, not necessarily stored in the form of text files.

Sedna is in the second class. It is a free native XML database which provides a full range of core database services - persistent storage, ACID transactions, security, indices, hot backup. Flexible XML processing facilities include W3C XQuery implementation, tight integration of XQuery with full-text search facilities and a node-level update language.

The management of XML documents is based on the B-tree architecture of Sedna, which gathers related XML documents in one same collection, and each XQuery can be evaluated on the specified collection.

I developed the CGI based management service in my project.

The XQuery Evaluation is implemented in server side by XQJ (XQuery API for Java) library [2];

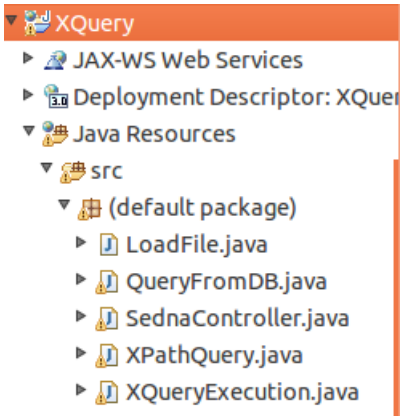
The XPath Evaluation on original Data Graph is implemented in server side by The Java XPath API ;

The XPath Evaluation on Indexed XML documents is implemented in server side by XimpleWare [3].

In the following section, I'll explain some critical codes of How To use these APIs in my project [4].

As the XML has an important feature -- Readable, so it essential to keep the output of XML query results formatted. In order to fulfill this feature, I used javascipt and Cascading Style Sheets (CSS) to pretty the output, which will also be generally explained afterwards.

3. Critical Part Analyze



To have a glance at the Servlets:

SednaController.java is used to management the XML documents in Sedna Database;

QueryFromDB.java is used to evaluate XQuery expressions;

XPathQuery.java is used to both XPath on original Data Graph and Indexed XML documents.

Part I. Management

Since the Sedna only provide C++ APIs, in order to manage it with Java, I execute the command line of Sedna from Java code:

```
    private String ListCollections() {
        String result="";
        try {
            pr = rt.exec(SEDNA_BIN+"/se_rc -sm-list");
        } catch (IOException e) {
            e.printStackTrace();
        }
        BufferedReader stdInput = new BufferedReader(new
                 InputStreamReader(pr.getInputStream()));

        BufferedReader stdError = new BufferedReader(new
                 InputStreamReader(pr.getErrorStream()));


        try {
            // read the output from the command
            while ((s = stdInput.readLine()) != null) {
                if(result.equals("")){
                    result=s;
                }
                else {
                    result = result+","+s;
                }
            }
            // read any errors from the attempted command
            while ((s = stdError.readLine()) != null) {
                result = s;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        return result;
    }
```

This snips is in SednaController.java, used to list all running collections of Sedna Database.

The line pr = rt.exec(SEDNA_BIN+"/se_rc -sm-list"); is used to execute the command supplied by Sedna.

Part II. XQuery / XPath Evaluation

1. XQuery Evaluation:

```
out.append("<result>");
XQSequence xqs = xqe.executeQuery(expression); // execution method 1: tp outputStream
if (xqs.next()) {
    xqs.writeSequence(out, null);
}
else {
    // This category consists:
    // 1. no query result
    // 2. plain string query , repsonse plain string
    out.println("No data found corresponding to this query.");
}
out.append("</result>");
```

This snips is in QueryFromDB.java, used to execute XQuery expression via XQJ API;

after being executed, the XQSeuence xqs stores the query results, XQJ API supplies functions directly writing result to HTTP response as xqs.writeSequence does.

2. XPath Evaluation on original Data Graph:

```
XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
XPathExpression expr = xpath.compile(expression);
// calculate time consuming
long execTime = 0;
long startTime = System.nanoTime();
Object result = expr.evaluate(doc, XPathConstants.NODESET);
long endTime = System.nanoTime();
execTime = endTime - startTime;
writer.append("<time>"+execTime/1000000+"ms"+(execTime%1000000)/1000+"</time>");

NodeList nodes = (NodeList) result;
String queryResult="";
for (int i = 0; i < nodes.getLength(); i++) {
    queryResult=queryResult+nodes.item(i).getNodeValue()+"\r\n";
}
writer.append("<result>"+queryResult+"</result>");
```

This snips is in XPathQuery.java, used Java XPath API to execute XPath expressions on specified XML document; it is independent to Sedna.

I used two long number to record the start time before execution of XPath and endtime after execution, then calculate the approximate execution of XPath on original Data Graph.

3. XPath Evaluation on Indexed XML document:

```
VTDGen vg = new VTDGen();
if (vg.parseFile(file.getAbsolutePath(),true)){
    // exam whether the index file is existed, and has the same hashcode
    vg.writeIndex(vxlPath); // create INDEX file "*.vxl"
```

This snips is in XPathQuery.java, used to create Indexed XML document based on the same file queried on original Data Graph as above; after executed vg.wirteIndex(original_file); the XimpleWare API will create a new Indexed file on the same directory named original_filename.vxl;

```
VTDNav vn = vg.loadIndex(vxlPath);
AutoPilot ap = new AutoPilot(vn);
long execTime = 0;
long startTime = System.nanoTime();
ap.bind(vn);
ap.selectXPath(expression);
ap.evalXPathToString();
long endTime = System.nanoTime();
execTime = endTime - startTime;
writer.append("<time>"+execTime/1000000+"ms"+(execTime%1000000)/1000+"</time>");
int i=-1;
String queryResult="";
while((i=ap.evalXPath())!=-1){
    queryResult=queryResult+vn.toString(i)+"\r\n";
}
writer.append("<result>"+queryResult+"</result>");
```

This snips is just following the above snips, it will load the index from the new created *.vxl file to Java running time, and then do the XPath query on Indexed XML;

I also calculate the execution time of XPath Evaluation on the Indexed XML after the *.vxl is created.

The comparison of performance according this two methods of XPath Evaluation will be stated in the next section.

Part III. pretty XML output

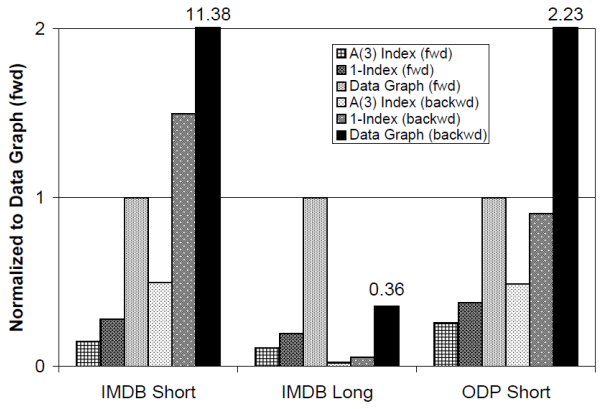
The algorithm is simple in javascript, I'd like to mention an important thing is the < and > should be converted to **<** and **>** to prevent them being recognized as HTML tags.

```javascript
function formatXml(xml) {
    var formatted = '';
    var reg = /(>)(<)(\/*)/g;
    xml = xml.replace(reg, '$1\r\n$2$3');
    var pad = 0;
    jQuery.each(xml.split('\r\n'), function(index, node) {
        var indent = 0;
        if (node.match( /.+<\/\w[^>]*>$/ )) {
            indent = 0;
        } else if (node.match( /^<\/\w/ )) {
            if (pad != 0) {
            pad -= 1;
        }
        } else if (node.match( /^<\w[^>]*[^\/]>.*$/ )) {
            indent = 1;
        } else {
            indent = 0;
        }

        var padding = '';
        for (var i = 0; i < pad; i++) {
            padding += ' ';
        }

        formatted += padding + node + '\r\n';
        pad += indent;
    });

    return formatted;
}
```

Beside this simple javascript in layout, there are also many open source CSS available could be used to preety the colorful output of XML formats.

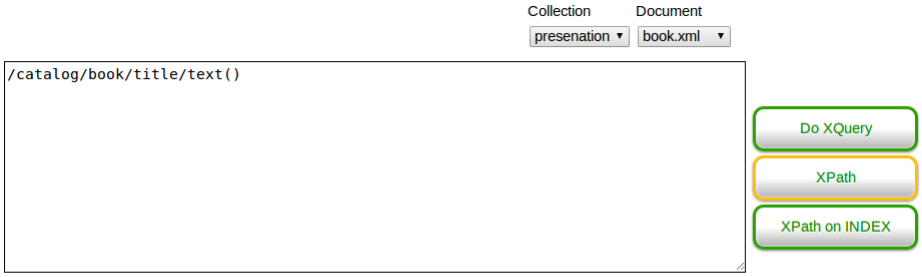
4. Performance

As I learned from the Paper “Exploiting Local Similarity for Indexing Paths in Graph-Structured Data” published in 2002, there are three common algorithms to do Path Evaluation in Graph-Structured Data. And XML can be converted to in Graph-Structured Data model.



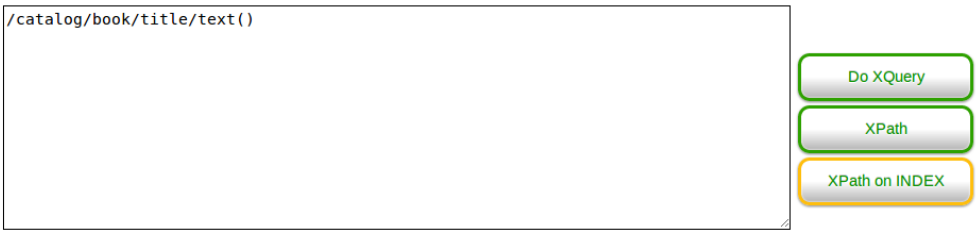
This picture shows the performance of Path Evaluation on Data Graph may be 2 to 9 times slower than that on 1-Index. According to the following two screen shots from my project, this prediction maybe correct in Java running time as well.

I did the same XPath query expression "/catalog/book/title/text()" on presentation collection book.xml document. The 1st executing time shows 3ms995, while the 2nd shows 0ms377; which is 10 times difference of the XPath Evaluation on Indexed XML.



Return Results

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
 <time>3ms955</time>
 <result>XML Developer's Guide
Midnight Rain
Maeve Ascendant
Oberon's Legacy
The Sundered Grail
Lover Birds
Splish Splash
Creepy Crawlies
Paradox Lost
Microsoft .NET: The Programming Bible
MSXML3: A Comprehensive Guide
Visual Studio 7: A Comprehensive Guide
</result>
</response>
```



Return Results

```
<?xml version="1.0" encoding="UTF-8"?>
<response>
 <time>0ms377</time>
 <result>XML Developer's Guide
Midnight Rain
Maeve Ascendant
Oberon's Legacy
The Sundered Grail
Lover Birds
Splish Splash
Creepy Crawlies
Paradox Lost
Microsoft .NET: The Programming Bible
MSXML3: A Comprehensive Guide
Visual Studio 7: A Comprehensive Guide
</result>
</response>
```

I'll generally talk about the Indexed method used by XimpleWare -- VTD-XML.

It is claimed to be the industry's most advanced and powerful XML processing model for SOA and Cloud Computing! It is simultaneously. And through looking into the VTD-XML, it is a little more efficient than 1-Index, but not as efficient as A(k)-Index performs introduced in the paper "Exploiting Local Similarity for Indexing Paths in Graph-Structured Data".

So, after I do a pile of tests on different XPath Evaluations, I found out the average performance of XimpleWare is 5 to 6 times than that of original Java XPath API.

5. Future Development in My Opinion

When I was searching for Indexing processing projects for XPath Evaluation, I was firstly aimed at A(k)-Index. However, I could not find it. I guess maybe the implementations on Persistence Storage solutions with plain XML Database are not popular as traditional Relational Database like SQL based ones.

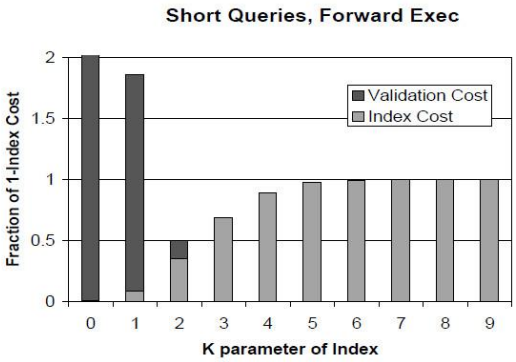


Figure 9. Short Queries on ODP Data

But XML, having features of readable make it quite unique. The users of the database can be non-technology people, like public library resources.

The other advantage of XML, distributable, make is benefit to distributed systems like Distributed Hash Table, which is also an academic project.

6. Related Work

A widely used XML database is BaseX . It supports various environments, also support XSLT, which is a useful technology in programming.

References:

[1] Sedna http://www.sedna.org/

[2] The XQuery Evaluation is implemented in server side by XQJ (XQuery API for Java) library http://xqj.net/sedna/ ;

[3] The XPath Evaluation on original Data Graph is implemented in server side by The Java XPath API http://www.ibm.com/developerworks/library/x-javaxpathapi/index.html ;

[4] The XPath Evaluation on Indexed XML documents is implemented in server side by XimpleWare http://www.ximpleware.com/ ;

[5] XML Database, http://en.wikipedia.org/wiki/XML_database ;

[6] BaseX , http://basex.org/;

[7] VTD-XML, http://vtd-xml.sourceforge.net/ ;

[8] Exploiting Local Similarity for Indexing Paths in Graph-Structured Data, authors: Raghav Kaushik, Pradeep Shenoy, Philip Bohannon, Ehud Gudes, published in 2002.