MEASURING AND IMPROVING DATABASE QUERY PERFORMANCE FOR LARGE XML STRUCTURES

Marko Niinimaki[1], Matti Heikkurinen[2], Jan Schmidt[3] and Kitichai Chanyalikit[4]

[1]Lecturer, Webster University Thailand, 1 Empire Tower, South Sathorn Road, Bangkok 10120, Thailand, niinimakim@webster.ac.th

[2]Senior Advisor, Ludwig-Maximilians-Universität München, Geschwister-Scholl-Platz 1, 80539 Munich, Germany, heikku@nm.ifi.lmu.de

[3]Researcher, Ludwig-Maximilians-Universität München, Geschwister-Scholl-Platz 1, 80539 Munich, Germany, schmidtja@nm.ifi.lmu.de

[4]Lecturer, Webster University Thailand, 1 Empire Tower, South Sathorn Road, Bangkok 10120, Thailand, chanyalik@webster.ac.th

ABSTRACT

We measured XPath query performance with two XML database packages using a large sample of XML documents and queries utilizing both the document structure and keywords. One of the packages is a “native” XML database, and the other one an XML-enabled relational database system. With large number of documents, both packages have performance problems. Using MongoDB improves the performance of some of the queries significantly, with some limitations on the expressive power of the query language. This is most likely an acceptable trade-off in most of the application scenarios.

KEYWORDS: XML, database, performance, XPath, query

1. Introduction

As defined by a W3C recommendation [1], the Extensible Markup Language, abbreviated XML, describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. As presentation and data storage format, the benefits of XML are often seen as [2].

1) being easy to understand and read

2) having a large number of supporting platforms being manageable through an even larger set of tools for data reading, writing, and manipulation

3) being used across open standards

4) allowing developers to create their own data definitions and models of representation and

5) for complex data structures, being simpler to use than binary formats, due to a large number of tools available.

Though XML was originally seen as the standard for structuring and exchanging data over the Web, interest in using it also for storage and retrieval [3] soon followed. XML documents were initially stored in files and query languages like XPath [4] extracted information from a file or a set of files containing a collection of XML documents. However, using databases with XML later became popular. Existing database packages became “XML enabled” and solutions to store XML documents in a dedicated native XML database have emerged [5].

Due to the popularity of XML and XPath, we omit a detailed discussion of their structure and features. Instead, we list the main terminology based on [1, 4] as follows:

XML documents are made up of storage units called entities or nodes. A document begins in a "root" or document entity. Elements, delimited by start- and end-tags, nest properly within each other. An element has a name (corresponding to its start tag), 0..n attributes and (optionally) contents. The contents can be textual or consist of (sub)elements. Thus, the document forms a tree of elements or nodes. An attribute has a name and a value.

An XPath query expression is used to retrieve elements, their textual contents and attribute values that match the expression in an XML document. Unfortunately, XPath query evaluation is potentially a daunting computational task. Gottlob et al. [6] have shown that in most XPath query systems, the processing time grows exponentially with the size of the query. However, by limiting XPath to a “usable fragment”, they manage to develop a query evaluation algorithm that runs in linear time.

Our main goal is to evaluate the performance of data retrieval from a large set of XML documents stored in a database based on three simple, but realistic usage scenarios. Our sample data consists of hundreds of thousands of XML documents of varying sizes, downloaded from the U.S. National Institute of Health’s PubMed collection [7]. Each document contains detailed metadata of a medical article, often including the entire article text (omitting figures). Large samples of the documents are loaded into two well-known database products. Database product #1 is an “XML enabled” relational database management system (RDBMS), whereas database product #2 is a native XML database. We test the performance of both products using XPath queries that represent typical end-user information needs of varying complexity. We find that with large samples, the performance of both database products is unsatisfactory. To alleviate the problem, we convert our documents into a JSON (JavaScript Object Notation) form, load them to MongoDB, a so called “no-SQL” (document based) database system, convert our queries to support MongoDB’s query language and demonstrate the improved performance.

The main contribution of the paper is to complement studies by Runapongsa et al [8], Florescu and Kossman [9] and Boicea, Radulescu and Agapin [10] using similar measurement methods, but with actual (not simulated) large XML structures from a medical database.

The rest of the paper is organized as follows: in Section 2, we describe our motivation for this research, and related work. The sample XML documents, hardware and software environments and methods of measurement are introduced in Section 3. The performance results are presented in Section 4. In Section 5, we demonstrate an improved method for data storage and querying, and present its performance. Finally, Section 6 contains a summary and items for future research.

2. Motivation and Related Work

XML, the Extensible Markup language was originally an initiative to encode semantics into documents, a capability that is lacking in HTML (Hypertext Markup Language) [11]. In the 2000’s, XML started to be seen as the “language of the world wide web”: data was stored in an XML format and formatted as HTML for presentation in a web browser. Moreover, XML was used as a format for distributed services over the internet as with XML-RPC (XML Remote Procedure Call) and SOAP (Simple Object Access Protocol) [12]. Due to the popularity of XML, managing and querying data in large XML documents (or a large collection of XML documents) became essential. A stand-alone program like xpath (based on the libxml2 library developed by the Gnome project) reads XML documents and evaluates an XPath expression such that parts matching the expression are retrieved (and printed) [13]. Xpath, however, processes only one document at a time, needs to read the entire document in the memory and does not use permanent indexing of XML structures for speeding up queries. Storing XML documents in a database would overcome these limitations and provide better data management capabilities. Both XML-enabled relational databases systems and "native" XML databases have entered the market [14]. Despite their benefits over file-based processing, query performance can be slow when the document collection gets large. In this research, our aim is specifically to identify potential query performance problems with large document collections, and to propose solutions for improving the performance.

Researchers often make a distinction between data centric and document-centric XML. Bertino and Catania [15] describe data-centric documents as regular in structure and homogeneous in content, like invoice information. In document-centric XML, the structure is more irregular, and data are heterogeneous, as in books and e-mail messages (in the XML form). Obviously, data-centric XML is faster to process since we can rely on its regular format. Our sample documents have both data-centric and document-centric features. Since the documents contain both meta data of an article and (often) the actual article contents, there are regular elements (each article has publication year) and irregular elements (some articles have tables nested inside other tables).

In related work Runapongsa et al. [8] have constructed a general benchmark for XML queries. Balamurugan and Ayysamy [5] have compared performance between an XML-enabled database system and a native database. However, the sample size of XML documents used is quite small. Schmidt et al. [16] discuss benchmarking XML databases in general, and Florescu and Kossman [9] evaluate different methods of storing XML structures into relational databases. In order to speed up XPath queries, Khatchadourian et al. [17] propose parallelizing the queries using Hadoop. Boicea, Radulescu and Agapin [10] compare MongoDB with the Oracle database management system, and Dede et al. [18] discuss the performance of a MongoDB based solution with a 300 GB data sample (but its format was not XML).

3. The Environment and Data

We have built and executed our query performance benchmarking in a relatively typical higher end Linux environment. The hardware is a 24-core Xeon server (E5-2620 v2 @ 2.10 GHz) with 32 GB memory running the Debian 8 distribution of the Linux operating system. Our native XML database is a Java-based open source BaseX 9.2.2, and it runs on Java version 1.8.0_66. The XML enabled relational database management system is MariaDB 10.0.32 and it was installed using the Linux distribution's yum install program.

Our data consists of hundreds of thousands of XML documents downloaded from the U.S. National Institute of Health's PubMed collection of medical articles [7]. The articles (without images) are available in compressed files at ftp://ftp.ncbi.nlm.nih.gov/pub/pmc. The size of the compressed files is currently about 50 GB, and the uncompressed size about 140 GB. At the time of the writing, the files contained 2.1 million articles and thus the average size of an XML file was 67 kilobytes. The earliest article in the collection is from 1610, but almost 90% of the articles are from 2000 or later. The 1610 article is "The Whole Aphorismes of Great Hippocrates". Some documents contain years earlier than 1610 in the publication information, but they seem to be mistakes and years according to the Islamic calendar.

About 5% of the articles are "stubs": instead of actual article text, they contain only "scanned-page" tags with a name of the TIFF file that contains the scanned page. However, even these articles contain complete article publication data. The depth of the XML tree of these "stub" articles is 7. Articles with full text contents have a depth of 9 or more.

We have used five document sample sets for our measurements: a set of 100 000, 200 000, 300 000, 400 000 and 500 000 documents. The data was loaded into the native XML database using the database management system's "add" command designed for XML files. The native XML database was able to read our sample sets correctly, but after 750,000 documents, it failed since the number of elements exceeded 2^{31}. The relational database table was created to have two columns: a "name" column corresponding to the name of file from which the data was inserted, and a "contents" column (type text). The data was inserted into the table by SQL insert statements where the contents was loaded from a file.

The queries that we tested with the sets reflect actual information needs, and were as follows:

- Q1: Print the publication year (contents of element "year") of each article. The corresponding XPath expression is: //pub-date[@pub-type="ppub"]/year/text()
- Q2: If the article contains the word "genitalia" anywhere, print the article ID (element). /article[//text()[contains(.,"genitalia")]]/front/article-meta/article-id[1]
- Q3: Print the title of the most cited article and how many times it has been cited. The list of references within each article can be located at "//element-citation" and the title of each cited article at "//element-citation/article-title".

For Q1 and Q2, no specific “localization” for the underlying database management system was needed. The native database uses simply “xpath <expression>” syntax and the XML-enabled RDBMS uses “ExtractValues(column_name, xpath-expression)”. However, in the case of Q3, the XML-enabled RDBMS had limitations that prevented us from using the full XPath expression, so we used an XPath expression with SQL additions. The full XPath expression and its XPath+SQL counterpart are shown in Figure 1.

```
for $title in distinct-
values(//element-
citation/article-
title/text()) let
$total :=
count(//element-
citation/article-
title[text() = $title])
order by $total
descending
return   $title || " : "
|| $total
```

```
select
ExtractValue(contents,'//ele
ment-citation/article-
title') ,
count(distinct(ExtractValue(
contents,'//element-
citation/article-title')))
from docs  group by
ExtractValue(contents,'//ele
ment-citation/article-
title') order by
count(ExtractValue(contents,
'//element-citation/article-
title'));
```

Figure 1 XPath and XPath+SQL expressions for finding article titles and the times the article was cited

For curious readers, the most frequent year of publication was 2016, the word “genitalia” appeared in about 1,700 articles (of 500,000), and the most cited article was “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs”.

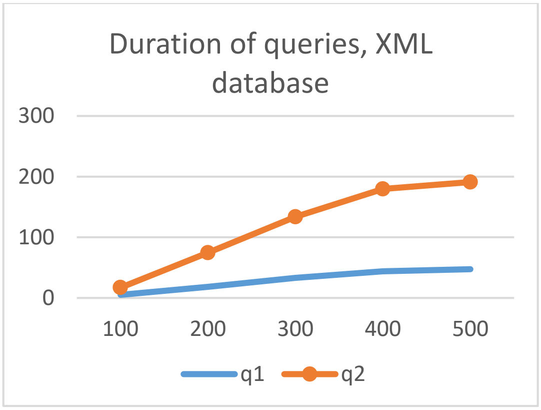
4. Measurements and Results

We timed the execution time of each query with the standard Linux “time” command. Each measurement was repeated several times. Other than the usual operating system tools, there were no programs executing in the computer during the measurements. On the native XML database, the execution time of Q3 was impractically long at 200000 or more documents (we interrupted the execution after it had been running for 3 weeks without producing a result).

The results are presented in Table 1 and illustrated in Figures 2 and 3. We can see that the XML database is faster with the "simple" queries Q1 and Q2, whereas the XML enabled RDBMS can handle the more complex query Q3.

Table 1 Duration of queries (in seconds)

	Native XML			XML-enabled RDBMS		
	Q1	Q2	Q3	Q1	Q2	Q3
100k	5.1	17.2	215621	10.5	145.8	45.5
200k	18.3	74.6	-	53.0	889.0	262.0
300k	33.1	133.7	-	91.9	1731.6	523.1
400k	44.0	179.6	-	122.0	2415.8	722.3
500k	47.5	191.0	-	136.6	2684.0	834.2



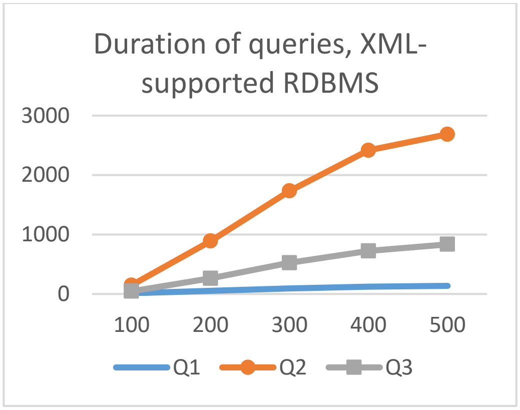


Figure 2 Duration of queries, XML database (left) and XML supported RDBMS (right)

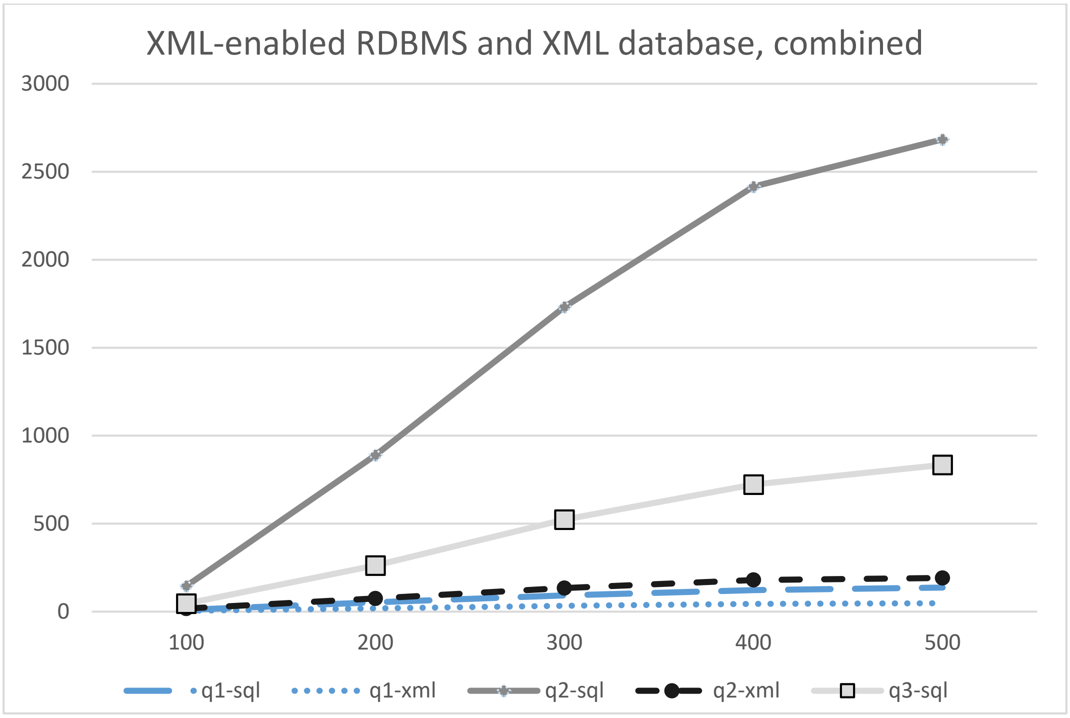


Figure 3 Duration of queries, combined

5. Measurements and Results: MongoDB

Our improved storage and query method is based on MongoDB 2.4.10, an open source "no-SQL" database [19]. MongoDB does not organize data in tables with columns and rows. Instead, data is stored in "documents", each of which is an associative array of scalar values, lists, or nested associative arrays. MongoDB documents are serialized as JSON objects, and are stored internally using a binary encoding of JSON called BSON [18]. MongoDB is able to import documents in JSON format. For converting our XML documents into JSON we used the Python based "xml2json" tool by H. Kranen. Xml2json was able to convert approximately 13 documents per second in our system. MongoDB's import speed was about 25 documents per second.

The queries modified to support MongoDB's query language are as follows:

- Q1 db.myCollection.find({}, { "article.front.article-meta.pub-date.year":1, "_id":0 })
- Q2 db.myCollection.runCommand("text", { search: "genitalia" })
- Q3 db.myCollection.aggregate([{$group : {_id : "$article.back.ref-list.ref.element-citation.article-title", "article-title" : {$sum : 1}}}, { $sort : { "article-title" : -1} }, { $limit: 500 }])

Other adjustments for MongoDB include setting the DBQuery.shellBatchSize variable to one million to support printing large result sets, enabling text indexing and creating a text index ("ensureIndex" command).

Results in Table 2 indicate that MongoDB is very fast with the more complex queries (Q2 and Q3) but slower than the XML and relational databases with the simple query Q1. Figure 4 illustrates the execution times of the queries in all three database packages.

Table 2 Duration of queries using MongoDB (in seconds).

	MongoDB		
	Q1	Q2	Q3
100k	24.5	1.2	5.2
200k	55.1	4.2	5.2
300k	84.9	4.3	10.5
400k	123.7	4.6	18.2
500k	144.3	5.3	18.4

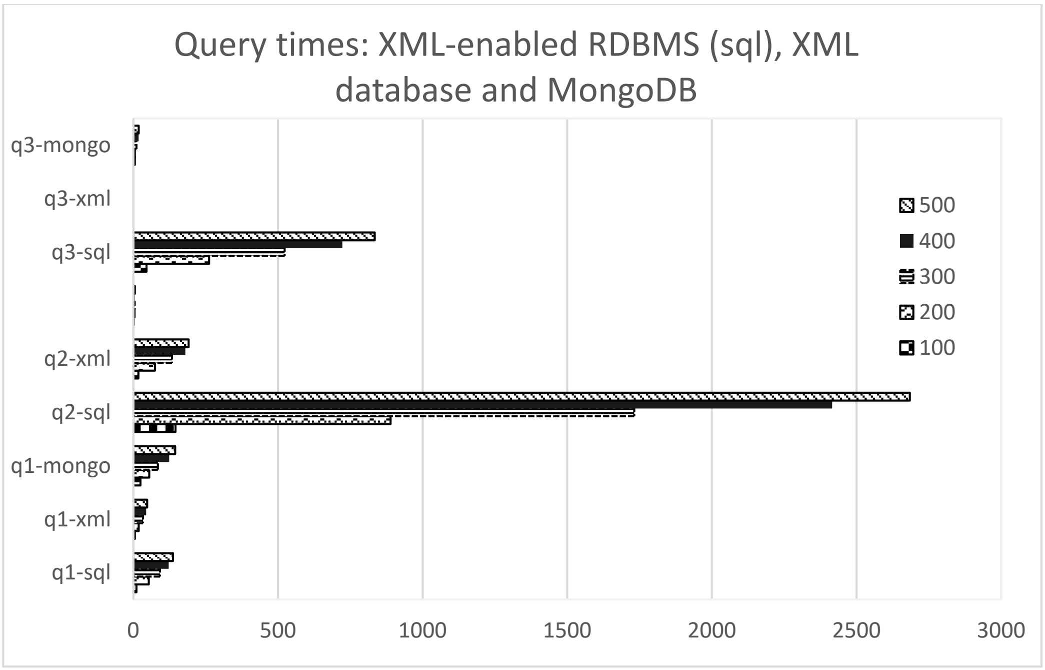


Figure 4 Performance of queries: RDBMS (sql), XML database (xml) and MongoDB

The results demonstrate an impressive gain of performance when using MongoDB for text search and aggregation. Simple queries seem to run faster with the RDBMS and XML databases.

However, since MongoDB's aggregation function seems to be very efficient, we can modify Q1 as follows:

Q1b: db.myCollection.aggregate([{"$group": {_id: "$article.front.article-meta.pub-date.year", count: {$sum: 1}}}, {$sort: {_id: 1}}])

This query prints the summaries of publication years within the collection and the output format as follows:

```
{ "result" : [
..
                { "_id" : "1901",   "count" : 1728  },
..
                { "_id" : "2016", "count" : 12698 },
```

Using this method, we obtain the following performance:

100k	200k	300k	400k	500k
0.7	1.5	2.4	3.9	4.0

A similar aggregation with the RDBMS "select ExtractValue(contents,'//pub-date[1]/year'), count(ExtractValue(contents,'//pub-date[1]/year')) from docs group by ExtractValue(contents,'//pub-date[1]/year');" runs for 540 seconds with 500,000 articles.

Parker, Poe and Vrbsky [20] have compared data retrieval with MongoDB and find MongoDB generally faster as well (but relational database systems are faster at loading data into the database). They think that MongoDB's fast retrieval performance is due to the combination of the index used by MongoDB and its use of memory.

6. Summary and Future Work

In this paper, we have compared the query performance of an XML-enabled relational database package and an XML database package using a collection of up to 500,000 XML documents. The documents originate from the U.S. National Institute of Health's PubMed collection. The queries represent “typical” tasks in an information system containing a database, namely: Q1 List the publication year of all the documents in the database; Q2 List the document ID’s of all documents containing word “genitalia” anywhere in the document; Q3 Find the article that is most cited by other articles in the collection.

With both the XML-enabled relational database package and the XML database package, we have used the xpath query language as it is supported by both. However, small adjustments were made in the queries since the XML-enabled RDBMS does not support all the functionality of xpath. For instance, xpath’s “for” construct that would be needed to express Q3 is not supported, and we therefore rewrote the query with a corresponding SQL construct.

Instead of a database package, we could use a standalone xpath program with a collection of XML documents stored in files. A simple measurement demonstrates that this is not practical. Evaluating Q1 with the xpath program and 100,000 documents took 14 hours in our test system (24-core Xeon server with 32 GB memory). With a database system, the query times were, of course, significantly better.

Despite the improvements, large amounts of documents still mean long query processing times. Processing Q2 with 500,000 documents still took minutes with the XML database and half an hour with the XML-enabled DBMS. To speed up queries, we have studied the feasibility of a “no-SQL” database MongoDB.

Obviously, MongoDB’s approach is not as rigorous as with other database products. It is difficult to see how MongoDB could support references to other documents in the collection in the same way as foreign keys in the relational database world or idref’s in the XML world. Similarly, MongoDB’s query language may not have the expressive power of XPath. However, we were able to convert our data and store it into MongoDB seemingly without difficulties. Our XML documents did not have multiple namespaces. Those would have been difficult to express with MongoDB. The queries could be expressed with MongoDB’s query language, too.

MongoDB’s performance was very good, especially with aggregation queries. There, the execution time was only about 10% of the corresponding query execution with the XML database and XML-enabled RDBMS. As a summary, we see a MongoDB based solution feasible if the focus is performance rather than complicated queries.

One of topics of our earlier database research has been energy efficiency of database applications [21]. In the future, we plan to expand this research into XML databases.

Acknowledgement

The research activities described on this paper have received partial support from the PROCESS project (https://www.process-project.eu/), that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 777533.

References

[1] Bray T, Paoli J, Sperberg-McQueen CM, Maler E, Yergeau, F. Extensible Markup Language (XML) 1.0. The World Wide Web Consortium, 2008.

[2] Evjen B, Sharkey K, Thangarathinam T, Kay M, Vernet A, Ferguson S. Professional XML, New Jersey: Wrox Publishing; 2007.

[3] Elmasri R, Navathe S. Fundamentals of Database Systems, Boston: Addison-Wesley; 2011.

[4] Robie J, Dyck M, Spiegel J. XML Path Language (XPath) 3.1. The World Wide Web Consortium, 2017.

[5] Balamurugan S, Ayyasamy A. Performance Evaluation of Native XML Database and XML Enabled Database. International Journal of Computer Science and Software Engineering 2017;7(5):182-91.

[6] Gottlob G, Koch C, Pichler R. Efficient algorithms for processing XPath queries. ACM Transactions on Database Systems (TODS) 2005;30(2):444-91.

[7] Steinbrook R. Public access to NIH-funded research. New England Journal of Medicine 2005;352:1739-41.

[8] Runapongsa K, et al. The Michigan benchmark: towards XML query performance diagnostics. Information Systems 2006;31(2):73-97.

[9] Florescu D, Kossmann D. A performance evaluation of alternative mapping schemes for storing XML data in a relational database, INRIA, 1999. Report 3680.

[10] Boicea A, Radulescu F, Agapin LI. MongoDB vs Oracle - database comparison. Proc. third international conference on emerging intelligent data and web technologies, September 2012 Bucharest, Romania, IEEE; 2012.

[11] Lie H, Saarela J. Multipurpose Web publishing using HTML, XML, and CSS. Communications of the ACM 1999;42(10):95-100.

[12] Cerami E. Web services essentials: distributed applications with XML-RPC, SOAP, UDDI & WSDL. Sebastopol: O'Reilly; 2002.

[13] Krulis M, Yaghob J. Efficient implementation of XPath processor on multi-core CPUs. Databases, Texts, Proceedings of the Dateso 2010 Workshop, Prague, Czech Republic. 2010.

[14] Vakali A, Catania B, Maddalena A. XML data stores: emerging practices. IEEE Internet Computing 2005;9(2);60-9.

[15] Bertino E, Catania B. Integrating XML and databases. IEEE Internet Computing 2001;5(4):84-8.

[16] Schmidt AR, Waas F, et al. The XML benchmark project. CWI, Amsterdam. 2001. Report INS-R0103.

[17] Khatchadourian S, Consens M, Siméon J. ChuQL: processing XML with XQuery using Hadoop. in Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research (CASCON '11), Toronto, Canada, IBM; 2011.

[18] Dede E, Govindaraju M, Gunter D, Canon RS, Ramakrishnan L. Performance evaluation of a Mongodb and Hadoop platform for scientific data analysis. Proc. 4th ACM Workshop on Scientific Cloud Computing, New York, ACM; 2013.

[19] Banker K. MongoDB in action. New York: Manning Publications; 2011.

[20] Parker Z, Poe S, Vrbsky SV. Comparing noSQL Mongodb to an SQL db. In Proceedings of the 51st ACM Southeast Conference, Savannah, Georgia, ACM; 2013.

[21] Niinimaki M, Abaunza F, Niemi T, Thanisch P, Kommeri J. Energy-efficient query processing in a combined database and web service environment. Green Computing Strategies for Competitive Advantage and Business Sustainability. Hershey: IGI Global; 2018.

Author's Profiles

Marko Niinimaki (niinimakim@webster.ac.th) is a lecturer of computer science at Webster University Thailand, Bangkok center. Mr. Niinimaki gained his Ph.D. at the University of Tampere, Finland. His research interests include databases, big data and computer security.

Matti Heikkurinen (heikku@nm.ifi.lmu.de) has been involved with large-scale e-Infrastructures at CERN and later as a consultant. He currently works with a European Union funded PROCESS project and lives in Geneva, Switzerland.

Jan Schmidt (schmidtja@nm.ifi.lmu.de) is a graduate student at Ludwig-Maximilians-Universität Munich, Germany. His research interests include computer security and Internet of Things.

Kitichai Chanyalikit (chanyalik@webster.ac.th) earned his MS ITC degree in Bangkok. He joined Webster University in 2017 and his specialty is computer networks.

Article History:

Received: January 11, 2019

Revised: August 1, 2019

Accepted: August 5, 2019