Measuring transaction performance based on storage approaches of Native XML database

Mohsen Marjani, Fariza Nasaruddin, Abdullah Gani, Shahaboddin Shamshirband

1. Introduction

Many organizations today use XML data interchange format for their business-to-business applications, and this has made XML to be one of the main storage formats used for storing critical business information and permanent data [1-4]. XML documents are usually managed via a common XML repository. In the raw documents, data are irregular, deeply hierarchical, and recursive, hence, we cannot rely on the relational database management systems (RDBMS) to provide the required efficiency and effectiveness to manage the data [5]. A purely relational model can be utilized but it has limitations such as in handling multi-way joins, and complex queries. XML and Native XML database (NXD) systems which store XML data natively using three main storage techniques – text-based, model-based, and schema-based techniques; and Hybrid Database systems which are comprised of both XML-Enabled and Native XML database systems. NXDs are faster than other database technologies because there is no need to convert the format of the data prior to storage. No performance evaluation has been carried out to compare all three storage strategies, hence, this paper reports on the first attempt to evaluate all three storage strategies by using open source products to measure the response time taken for each of the database basic tasks such as database creation, dataset insertion, and data manipulation. The results of the evaluation show that the schema-based storage strategy: performs 3.5 times faster than the other two storage techniques in data insertion; shows very good performance in query processing on small and large datasets; performs 10.33 times faster than text-based, and 7.5 times faster than model-based storage techniques in query processing of large datasets.

Many organizations today store their critical business information permanently in XML format. XML data can be managed using: XML-Enabled Database (XED) systems which convert and store XML files in traditional database systems; Native XML Database (NXD) systems which store XML data natively using three main storage technologies – text-based, model-based, and schema-based techniques; and Hybrid Database systems which are comprised of both XML-Enabled and Native XML database systems. NXDs are faster than other database technologies because there is no need to convert the format of the data prior to storage. No performance evaluation has been carried out to compare all three storage strategies, hence, this paper reports on the first attempt to evaluate all these storage strategies by using open source products to measure the response time taken for each of the database basic tasks such as database creation, dataset insertion, and data manipulation. The results of the evaluation show that the schema-based storage strategy: performs 3.5 times faster than the other two storage techniques in data insertion; shows very good performance in query processing on small and large datasets; performs 10.33 times faster than text-based, and 7.5 times faster than model-based storage techniques in query processing of large datasets.

1. Introduction

Many organizations today use XML data interchange format for their business-to-business applications, and this has made XML to be one of the main storage formats used for storing critical business information and permanent data [1-4]. XML documents are usually managed via a common XML repository. In the raw documents, data are irregular, deeply hierarchical, and recursive, hence, we cannot rely on the Relational Database Management Systems (RDBMS) to provide the required efficiency and effectiveness to manage the data [5]. A purely relational model can be utilized but it has limitations such as in handling multi-way joins, and complex queries. XML and Native XML database (NXD) offer a more “practical” solution for data in many of the real world documents [6].

Currently, many XML databases are available and they can be categorized into three types - XML-Enabled, Native XML, and Hybrid XML database systems. Among the three types, the Native XML database (NXD) requires no conversion of the data from its original format, as it stores XML documents natively. This saves time and resources because no computation is involved. Some of the NXD systems are briefly introduced in following.

Berkeley DB XML is an open source NXD system developed by Sleepycat Software using C++ + programming language. Berkeley DB XML is a key-value database which stores XML documents in their native format based on a text-based storage approach. Berkeley DB XML uses containers as logical storage units of XML documents in which multiple XML documents can be stored in every unit [7].

Xindice is a proprietary and open source NXD system developed by volunteers under Apache Software Foundation license using Java programming language. This database system stores and arranges XML documents based on a model-based storage approach into logical storage units called collections [8].

eXist is also a proprietary and open source NXD system developed by Wolfgang Meier using java programming language and under license GNU LGPL. It stores XML documents in collections as logical storage units based on a model-based storage approach [9].

dbXML is a proprietary NXD developed by dbXML group using C++ + programming language for earlier version of the system and using Java programming language for later versions. This NXD stores XML documents in collections as logical storage units based on a model-
based storage approach [10,11].

**Sedna** is an open source NXD system developed by MODIS team using C and C++ programming languages. Sedna uses data blocks/sets as logical storage units and stores XML documents based on a schema-based clustering storage approach [12,13].

**OrientX** is developed by Renmin University of China. This NXD system arranges XML documents according to their schema and stores them in data sets as logical storage units and based on a clustering storage approach which uses that schema. OrientX uses its own storage subsystem to store XML in pages on disk and the granularity of storage can be changed according to schema. Hence, they call it a schema-dependent database [14].

Three types of XML storage strategies are used in current NXDs - text-based, model-based, and schema-based strategies. It is important to evaluate the performance of available XML databases for the benefits of developers, organizations, and researchers. Developers and organizations are keen to know which database can fulfill their needs, and in the most cost-effective manner [15]. Database researchers wish to know the factors that can affect performance in the real environment so that they can focus on making improvements on the design and implementation of new algorithms. Also, XML database researchers and developers need to know which storage strategies are suitable for developing NXDs and what are the impacts of current storage techniques on the performance of the NXDs.

The main objective of this paper is to compare the performance of the current storage strategies used in different NXDs. For this purpose, three NXDs will be selected, each using a different storage technique. A number of articles have been published on the performance evaluation of NXDs [16–22]. But only a few of these reports pertain to evaluation of the performance based on their storage formats [23,24]. Moreover, these studies were usually conducted separately, or piecemeal.

The first section of this paper discusses the different XML database technologies and the storage strategies for NXDs. This is followed by a review of the performance evaluation of the text-based, model-based, and schema-based storage strategies, which include: Berkeley DB XML; Xindice, eXist, and dbXML; and Sedna and OrientX approaches, respectively [8–10,12,14,25,26]. The last section of the paper discusses the performance evaluation of the three storage strategies.

### 2. Motivation

Database systems such as NXD manage the storing, retrieving, updating, deleting, and searching of many types of data [8,10,12,14,25,27]. Studies are still ongoing regarding the performance of existing NXD systems vis-à-vis their storage strategies. There are different NXDs with different storage approaches. The number of XML documents is increasing; hence, the need for better ways of storing, manipulating, and retrieving XML data is increasing too [1]. It is also crucial to look into various aspects of XML storage performance such as: evaluating which database system can fulfill the organizational needs in the most cost-efficient manner; and evaluating which storage approach has the best impact on the performance of the database systems.

Currently, there are a number of XML database system benchmarks - X007, XMark, XMach-1, and XBench [26]. Most of these benchmarks focus on the query capability of the XML databases. XMach-1 is the only system which has both query and update capabilities.

Some studies have been conducted on finding ways of improving and optimizing the NXDs’ performance [16–22]. A few studies have evaluated the performance of NXDs based on their speed for storing, retrieving, and updating data, as well as other processing functions based on their storage approaches. Mahanza conducted a performance evaluation on NXDs which use text-based and model-based storage approaches [23]. In a review of the literature, there has been no report on comparing all three storage strategies used in current NXDs.

### 3. Native XML storage strategies

Native XML databases (NXDs) store XML documents natively which means that there is no prior conversion of the XML data before storing them. There are three types of XML storage strategies in current NXDS - text-based, model-based, and schema-based. Table 1 summarizes the features of several NXDs using these three storage strategies, based on information extracted from researches from 2002 to 2013.

#### 3.1. Text-based storage strategy

In this type of storage strategy, XML data is stored as text in a file inside the database, or in a file system outside the database, or in a proprietary text format. This storage technique greatly facilitates the retrieval of whole XML documents or some fragments of the documents by using indexes (a single index-lookup). It is, however, not suitable for manipulating data inside an XML document which has an inverted tree (hierarchy) structure. Berkeley DB XML is an example of NXD that uses this type of storage approach.

The Berkeley DB XML is an embedded database built on top of the Berkeley DB library. It is a key-value database and stores XML documents in their native format using a text-based storage technique. One logical storage unit of an XML document contains one or more XML documents. XML data is stored in files known as containers. In comparison with relational databases that store data in relational tables, rows and columns, Berkeley DB XML stores XML data in some arbitrary binary trees and performs queries using the XPath query language in the containers. It stores metadata attributes in each container as a top-level element. It does not require any Document Type Definition (DTD), and indexing is done manually. Therefore, programmers maintain the indexes for all documents in a container, and this greatly facilitates the query process. Indexes can be added to the container accurately after the container has been initialized and before any document is added into the container. Containers hold the indexes and information of the XML documents.

Similar to the relational databases, Berkeley DB XML has the capability for specifying and highlighting a part of the data to be indexed in order to achieve faster retrieval. It provides a single mechanism for maintaining indexes for XML data based on four characteristics - path type, node type, key type, and syntax type - to facilitate look-up. Indexes can be replaced, deleted, and added. Updated XML documents contained in database supports XUpdate, which is a programmable API with similar functionality as the XUpdate [23].

#### 3.2. Model-based storage strategy

In this storage strategy, XML documents are stored by using an internal object model inside either a relational database or an object-oriented database. This strategy is good for producing a document in
Document Object Model (DOM) format by using a DOM model, but it is not good at returning data in a format different from the format the data is stored in the XML document [28]. Xindice, eXist, and dbXML are three examples of NXDs that use this storage strategy.

Xindice, eXist, and dbXML are proprietary NXDs which store XML data in logical units, known as collections. dbXML needs Data Type Definition (DTD) or a schema to store XML documents, but DTD is optional for eXist, and is not needed for Xindice. However, Xindice and dbXML allow manual indexing to improve query processing, but in eXist, indexing is auto-generated, before the XML documents are stored.

Xindice stores XML documents in the database based on a model-based storage strategy. Like the traditional file system, XML documents are arranged in a hierarchical structure. This NXD eliminates the need to map XML documents to a query language such as SQL and XPath. However, it uses XML:DB XUpdate to update the database or a single document or all collections of the documents. This NXD provides a command line tool with the same functionality as the XML:DB API. It allows manual indexing through its command line interface.

eXist can be developed using different methods such as a stand-alone server process, inside servlet-engine (servlet engine is a Java-based platform technology to enhance the capabilities of the server by using servlets, which are small Java-based programs that can run on servers like web servers), or directly embedded into an application. It is readily accepted by applications that use XML documents (web-based applications or CDROM-based applications). It manages a hierarchy of collections (comparable to file storage in a file system) and stores one or more XML documents in each collection inside a schema-less storage. It stores XML documents as a DOM tree. It uses an improved query processing technique - a numerical indexing schema - which helps to identify the structural relationships between nodes quickly, for instance, to determine which node in the tree is the parent node of which nodes, and vice versa. Nodes for each document are elements, attributes, texts, and comments. eXist uses indexes for all nodes in the XML document and performs full-text indexing of the whole text and the values of all attributes as a default. Query processing is index-based and queries are executed at the collection level. eXist supports the XPath query language with extended syntax for querying specific sections of the hierarchical collections or all the XML documents stored in the databases. eXist uses XUpdate for updating the XML documents.

dbXML manages collections in a hierarchical manner, comparable to an operating system directory structure. Each collection contains one or more XML documents based on a schema or a DTD. This NXD stores either schema-less XML documents or XML documents that conform to any XML schema. It offers value indexing, full-text indexing, and name indexing based on some patterns using elements or attributes as well as full-text to increase its query performance. XPath is used as the query language to query a specific collection or a document inside a collection. Full-text searches are supported. dbXML uses the XUpdate technique for updating the database.

In a study, Manbanza compared model-based and text-based storage strategies [29]. As shown above in Table 1, we compared all three storage strategies used in NXDs. Table 2 summarizes six NXD systems which use the three main storage strategies, respectively, and Table 3 presents some general features of the six NXDs.

<table>
<thead>
<tr>
<th>Storage strategies</th>
<th>NXD</th>
<th>Logic unit</th>
<th>DTD</th>
<th>Indexing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text-based</td>
<td>BerkeleyDB</td>
<td>Container</td>
<td>No</td>
<td>Manual</td>
</tr>
<tr>
<td></td>
<td>Xindice</td>
<td>Collection</td>
<td>No</td>
<td>Manual</td>
</tr>
<tr>
<td></td>
<td>eXist</td>
<td>Collection</td>
<td>Optional</td>
<td>Auto</td>
</tr>
<tr>
<td></td>
<td>dbXML</td>
<td>Collection</td>
<td>Yes</td>
<td>Manual</td>
</tr>
<tr>
<td>Model-based</td>
<td>Xindice</td>
<td>Collection</td>
<td>No</td>
<td>Manual</td>
</tr>
<tr>
<td></td>
<td>eXist</td>
<td>Collection</td>
<td>Optional</td>
<td>Auto</td>
</tr>
<tr>
<td></td>
<td>dbXML</td>
<td>Collection</td>
<td>Yes</td>
<td>Manual</td>
</tr>
<tr>
<td>Schema-based</td>
<td>Sedna</td>
<td>Data set</td>
<td>No</td>
<td>Auto</td>
</tr>
<tr>
<td></td>
<td>OrientX</td>
<td>Data set</td>
<td>No</td>
<td>Auto</td>
</tr>
</tbody>
</table>

### 3.3. Schema-based storage strategy

In this storage technique, XML documents are stored based on a schema built into the database. A standard schema (or DTD) is normally defined in the underlying XML files to facilitate data exchange. Chemical Markup Language, Mathematical Markup Language, and News Markup Language are examples of standard schema or DTDs. Sedna and OrientX are examples of NXDs that use the schema-based storage approach.

Sedna has a technique for managing the memory based on layered address space to produce a clear address using a combination of unique identifiers and qualifiers. It performs indexing based on the B-tree. It uses the W3C XQuery query language, which is validated by the W3C XQuery Test Suite. Full-text searching is possible through its integration with XQuery. It uses a declarative node-level update language for database updating. The NXD organizes data in two ways to improve query processing performance. It uses direct pointers to represent relationships among nodes to indicate which one is the parent node and which one is the child or sibling node, and follows a direct pointer to traverse an XML document, unlike the relational-based approach which uses joins to traverse an XML document. Also, it uses a descriptive schema-driven storage strategy which shows the positions of clustering nodes of an XML document.

Grineva, Gribneva, and Lizorkin stated that in contrast to the prescriptive schema that is known in advance and is usually specified in DTD or XML schema, the descriptive schema is generated from data dynamically (and is maintained incrementally), and it represents concise and accurate structure summary for data [12]. The descriptive schema has the following advantages over the prescriptive schema: (1) it is more accurate; (2) the storage strategy is applicable to any XML document, even the document that has no prescriptive schema.

OrientX stores XML documents based on the clustering storage strategy based on that schema, hence it is a schema-dependent database. It uses the schema to choose a suitable storage strategy, automatically. It integrates several requirements such as a composite index manager, a cost-based XQuery Optimizer, several XML query evaluators, an Access Control module, and an extension to XQuery1.0 for XML update, besides a native storage system on XML data repository. OrientX handles both XQuery and XPath query languages efficiently because it stores XML documents in their native format.

OrientX’s schema information can be searched to guide storage, optimization and query processing to improve the performance of the system, noticeably.

OrientX uses schema information to validate checking in query or updating processing. The Optimizer inside OrientX integrates the schema with the histogram and collects the statistical information. A role-based access control is used in OrientX. The structure of the tree for the roles is the same as the structure of the schema information. It means that each node in the schema corresponds to a role in the access control. It maintains path indexing in accordance with the schema. OrientX summarizes all paths in the data.
Meng, Wang, Xie, Zhang, and Zhou stated that OrientX stores XML data using its own storage subsystem, Orient Store, in which XML data are stored in pages in the disk, and the granularity of storage can be changed according to the defined schema [14]. When stored data are loaded into memory, the tree structure of the related nodes is built dynamically. Therefore, OrientStore offers a DOM-like navigation interface to the upper modules. Besides OrientStore, there is a holistic data model scheme called SUPEX, in which all path indexes and some value indexes are put together. XPath queries can be handled efficiently through fast navigation and join operation with the help of SUPEX. There are two query evaluators for XQuery in OrientX - navigation-based and algebra-based evaluators.

4. Testing environment and methodology

In our study, we evaluated three NXDs, each one using one of the three different storage strategies, respectively. Rasha and William conducted a survey on the database system performance evaluation models and provided details on some of the current evaluation models [28]. However, Manegold and Manolescu contended that to conduct a proper evaluation, we need to analyze the data to understand the behavior and characteristics of the database, and also to find out where and why time is consumed in a process [30]. We should then highlight the important and relevant details and compare the results of the evaluation with other evaluations. To conduct any evaluation, we need to use the appropriate hardware and software, as well as the metrics to use and what we are going to measure. Manegold and Manolescu suggested three steps for evaluating the performance of databases: Step 1: measuring throughput like query processing response time, evaluation time for wall-clock, CPU, I/O, and server-side vs. client-side, measuring the amount of memory, storage usage, and requirements; Step 2: making comparison based on scale-up or speed-up approach; Step 3: analyzing the results, system events and interrupts, or even hardware events [30].

In our study, we set up the necessary testing environment to test all the three storage strategies using Berkeley DB XML to represent a text-based NXD, eXist to represent a model-based NXD, and Sedna to represent a schema-based NXD. The measurement or evaluation criteria used in this comparative study is time, which includes the time taken to execute and complete standard database operations such as insert, delete, modify, and query, of each NXD. The performances of the three selected NXDs are compared based on the total amount of time each NXD takes for all the operations. The following sections discuss the testing environment, the time measurement, and a comparison of the methods used, in more detail.

4.1. Testing environment

To eliminate and avoid any corruption of the test results that could be caused by other factors such as transformation and changing of the output, network overhead, and communication costs, we used a PC system with 2 GB RAM, 2.2 GHz of CPU speed, and 640 GB hard disk capacity. The selected open source NXD systems were installed using an appropriate OS. A program was written to load the ‘synthetic’ datasets obtained from the University of Washington repository [31]. We evaluated the performance of the NXD based on the response time for database creation, dataset insertion, and retrieval time for simple and complex queries. Three distinct XML documents of three different sizes - small, medium, and large - were used as datasets, and a list of simple and complex queries was created based on the query language of the selected NXDs and the content of the selected XML files. We selected three datasets of different sizes with the small dataset being less than 0.5 MB, the medium dataset being more than 23 MB, and the large dataset being about 127 MB, to study the performance of NXDs for storing, and applying different types of queries - from simple to complex queries.

startTime1 = System.currentTimeMillis();
CreateEnv Env1 = new CreateEnv();
CreateXML manager myContain1 = new CreateXMLmanager();
AddDocToContainer myDoc1 = new AddDocToContainer();
endTime1 = System.currentTimeMillis();
duration1 = endTime1 - startTime1;

Fig. 1. Sample of codes to obtain the response time for database creation.

4.2. Timing technique

We applied a timing technique similar to the methods used by Mabanza and Mabanza Chadwick, and Rao to measure the response time for each of the basic operations of the database [23,24]. Two check points were placed inside the codes to collect the timing data - one check point at the beginning of a request, and the second check point at the end of the request. The response time (in milliseconds) is the difference between the time recorded at the second check point and the time recorded at the first check point. Each request was repeated a number of times and the average time taken is calculated and recorded.

Fig. 1 shows a sample of the codes used to capture the response time for database creation. The first check point is represented by the identifier startTime1, and the identifier endTime1 is the identifier of the second check point. The identifier duration1, endTime1 - startTime1, is the response time of one transaction request. The response time or the duration of each transaction is printed out.

In our study, the time to connect to the database and to access a specific collection is not considered in the time measurement. The time measured is the instruction’s execution time for a specific and basic database operation.

4.3. Comparative methods

In our study, we compared: (1) the databases based on key features (as listed in Tables 1 and 2) for the three storage strategies, for example, based on their impacts of the performance reported from other research works; and (2) the results of the evaluation of the three selected NXDs as instances of three storage strategies.

5. Simulation system

This section describes in detail the simulation systems used in the evaluation which include their specifications and functionalities, and how each part of the system works and its function. The following sections discuss all requirements of the simulation engine and the hardware.

5.1. Simulation engine requirements

In order to evaluate the NXDs and to measure the response time of the databases transactions, a simulation engine was used to run tasks such as inserting datasets, and applying queries to the databases. The java-based engine was developed, and the database connection, required environment, and workspaces were also set up. Workspaces are used for inserting the prepared XML datasets and for applying queries. Three datasets of different sizes - small, medium, and large - were prepared, and three different queries for each database were designed. The simulation engine was used for connecting to the selected databases, creating appropriate containers, inserting XML datasets into the databases, applying different queries to the containers, and measuring the response time for inserting XML datasets, and for processing queries.

This engine has four different layers. The top layer is the user interface, which allows the client to select the sizes (small, medium, and
large) of the XML datasets, and also to select XML queries which range from simple to complex queries. The second layer consists of the requirements for preparing different XML documents, different queries, receiving the response time for inserting XML documents, and processing the different queries. The third layer consists of the simulation engine which is responsible for establishing the connection to the selected NXDs and sending XML documents to them, applying queries to the containers of the XML documents, and calculating the response time for XML document insertion and query. Fig. 2 shows the simulation engine architecture.

5.2. Hardware requirements

In order to develop the simulation engine, a fully functional personal desktop or laptop computer with minimum hardware requirements shown in Table 4, is needed.

6. System design

Fig. 3 shows the data flow diagram (DFD) to give a clear overview of our system. It also shows the design structure, data flow (the type of data going into and out from the system, the location and movement of data), and also the processes involved in the system.

6.1. Data flow diagram

The input data is in the XML documents or DTD files. Data is inserted into three selected NXDs. Fig. 3 shows Level 1 of the DFD. The Level 2 DFD is based on the process of each transaction that includes the initial process, database creation, insertion of XML documents, and queries. Based on the selected NXD and its functionalities for the initial processes, the database creation process, and inserting XML documents, the Level 2 DFD can be different. Therefore, there are different Level 2 DFDs. For all transactions, the processes are executed inside the NXDs while the simulation engine merely calls the proper classes and functions, sets the required variables and values, and subsequently, the NXD performs the required processes, and returns the results to the engine together with the appropriate messages.

7. Performance evaluation

To carry out performance evaluation, we used three XML datasets of different sizes - small, medium, and large - which were obtained from the University of Washington [31]. The use of different sizes of XML datasets was intended to have better test data to achieve better and more accurate performance evaluation of the selected NXDs in different situations, as in the real environment. Table 5 presents brief information on three prepared datasets – DBLP, NASA, and Sigmod Record datasets. The DBLP dataset includes bibliographic information on major computer science journals and conference proceedings, and is categorized a large-sized dataset. The NASA dataset is a legacy flat-file that had been converted into XML and is considered a medium-sized dataset. The Sigmod Record dataset consists of index of articles from SIGMOD Record and is considered a small-sized dataset [31].

Two types of evaluation - storing time, and retrieving time - were performed using the datasets. The following sections discuss the evaluation results.

7.1. Storing time evaluation

In this evaluation, the time taken to store every XML dataset inside each NXD is measured. Fig. 4 illustrates the storing process. This operation was repeated four times for each transaction and the average time was then calculated. The same operation was carried out for each NXD, each representing a different storage strategy, and on each dataset of three different sizes (small, medium, and large). The results were compared and the NXD with the fastest storing time for each dataset size was recorded.

7.2. Retrieving time evaluation

To evaluate the retrieval time, some queries were applied to the database to retrieve the desired sections of the XML datasets stored in the NXDs. For this purpose, three different query statements (based on the NXDs’ query languages) of different levels of complexity were used.

In this study, we applied three query statements on each dataset. Since there are three datasets (small, medium, and large), therefore, nine different queries were applied on each NXD. Each query transaction was repeated four times. The average of the retrieval times for each NXD was calculated and recorded. In order to ensure that the query was applied under similar condition for all the NXDs, and the retrieved data using the query from each database were similar, the data stored inside the NXDs must be the same. The NXD with the fastest average response time for query processing is deemed to perform better. Fig. 5 illustrates the query process for measuring the response time.

The process illustrated in Fig. 5 begins after inserting the related XML documents into the NXDs. At this stage, we used the appropriate query language and the prepared query statements in the appropriate syntax for all the NXDs. The result-set for each NXD and XML dataset for the same query statement is obtained. Three different query statements (converted to the NXD’s query language’s statements) with different levels of complexity were used for each XML dataset. We then used the appropriate Java classes and codes to apply the queries and to retrieve the result, and also to calculate the retrieving response time.
8. Evaluation results

Mabanza and Chadwick stated that the storage operation is fastest with Berkeley XML DB and slowest with eXist [24]. Of the six NDXs introduced earlier in this paper, we have chosen three NDXs—Berkeley XML DB, eXist, and Sedna. Berkeley XML DB is a text-based database, eXist is a model-based NXD, while Sedna is a schema-based NXD. Thus, in our study, we have used a complete set of database, each representing a different storage strategy used in current NDXs. The performance of each of these three storage techniques was evaluated based on the response time for data insertion and query processing. Table 6 provides salient information on the three NDXs used in the evaluation, complementing the information provided in Tables 2 and 3, above. Table 6 shows that the Berkeley DB XML is not a database server, but a file-based database [7]. Therefore, there is no host address and port for this NXD. The following sections present the results of the evaluation of the selected NDXs.

In the evaluation, data insertion was repeated four times for each NXD, with the response time recorded for each insertion. The average response time for data insertion for each NXD was calculated. Table 7 lists the response times for inserting large dataset (DBLP) into each NXD.

Table 8 shows the response times for inserting small dataset (SigmodRecord) into each NXD, and Table 9 shows the response times for inserting medium dataset (NASA) into each NXD.

To evaluate the response time for query processing, three queries with different levels of complexities were applied to each dataset of each NXD. With three queries for each dataset, therefore, nine queries were applied to each NXD. Table 10 lists the query statements for a large dataset (DBLP); Table 11 lists the query statements for a small dataset (SigmodRecord); and Table 12 lists the query statements for a medium dataset (NASA).

All the prepared query statements were applied to each NXD, and the response time was recorded for each query processing on each dataset of each NXD. The average response time was calculated for each transaction. Table 13 shows the response times for processing nine queries on Berkeley DB XML; Table 14 shows the response times for processing nine queries on eXist; and Table 15 shows the response times for processing nine queries on Sedna database.

Analysis of the above results and discussion on the implications are presented in the section below.

8.1. Data insertion response time

In Fig. 6, four columns to the right of each NXD show the response times for inserting the DBLP dataset. The first column shows the average of the response time for uploading a large XML dataset (DBLP dataset) into the selected NDXs. The result shows that the Sedna has the fastest response time. The response time for inserting DBLP dataset into
eXist database is the slowest, thus, it has the lowest performance. The difference between the fastest response time (highest performance) and the slowest response time (lowest performance) is significant. In fact, the performance of Sedna is three times better than the performance of eXist and BDB XML for inserting large dataset (DBLP dataset). In other words, Sedna is 350% better than eXist, and 298% better than BDB XML in uploading large files like the DBLP dataset. On the other hand, BDB XML is about 120% \((\text{eXist.Avg} / \text{BDB.Avg}) \times 100\) faster than eXist. In addition, Sedna can perform database creation and data insertion, simultaneously. This can increase the speed of data insertion, and hence, further enhancing the performance of Sedna in data insertion of large

### Table 6
Selected NXDs used in the evaluation.

<table>
<thead>
<tr>
<th>NXD</th>
<th>Version</th>
<th>Storage strategy</th>
<th>Logical unit</th>
<th>Query language</th>
<th>Host address</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDB XML</td>
<td>2.3.16</td>
<td>Text-based</td>
<td>Container</td>
<td>XPath</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>eXist</td>
<td>1.4.0</td>
<td>Model-based</td>
<td>Container</td>
<td>XPath</td>
<td>Localhost</td>
<td>5050</td>
</tr>
<tr>
<td>Sedna</td>
<td>3.4.66</td>
<td>Schema-based</td>
<td>Data Block</td>
<td>XQuery</td>
<td>Localhost</td>
<td>8080</td>
</tr>
</tbody>
</table>

### Table 7
Response time for data insertion of the DBLP dataset.

<table>
<thead>
<tr>
<th>NXD</th>
<th>Data insertion time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1th</td>
</tr>
<tr>
<td>BDB</td>
<td>128248</td>
</tr>
<tr>
<td>eXist</td>
<td>47877</td>
</tr>
<tr>
<td>Sedna</td>
<td>167850</td>
</tr>
</tbody>
</table>

### Table 8
Response time for data insertion of the SigmodRecord dataset.

<table>
<thead>
<tr>
<th>NXD</th>
<th>Data insertion time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1th</td>
</tr>
<tr>
<td>BDB</td>
<td>652</td>
</tr>
<tr>
<td>eXist</td>
<td>562</td>
</tr>
<tr>
<td>Sedna</td>
<td>209</td>
</tr>
</tbody>
</table>

### Table 9
Response time for data insertion of the NASA dataset.

<table>
<thead>
<tr>
<th>NXD</th>
<th>Data insertion time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1th</td>
</tr>
<tr>
<td>BDB</td>
<td>19713</td>
</tr>
<tr>
<td>eXist</td>
<td>20736</td>
</tr>
<tr>
<td>Sedna</td>
<td>6950</td>
</tr>
</tbody>
</table>
datasets, compared to the other two NDXs. Fig. 7 shows the response time of the NDXs for inserting a small dataset - SigmodRecord. Again, Sedna has the fastest response time, while BDB XML has the slowest response time.

Based on the results of the evaluation, Sedna also out-performs eXist and BDB XML in inserting a medium dataset and in creating a database. Fig. 8 shows that the time taken to insert the medium dataset (NASA dataset) is faster than the time taken to insert this dataset into eXist and BDB XML databases. eXist shows the worst performance in inserting a medium dataset. The performance of Berkeley DB XML is only slightly better than the performance of eXist.

Table 16 shows a comparison of the performance of the NDXs for inserting datasets of three different sizes. Sedna, which uses schema-based storage strategy, shows the best performance when compared to BDB XML (text-based) and eXist (model-based) NDXs.

### 8.2. Data retrieval and query response time

Three different queries ranging from simple to moderate to complex queries, were applied to each XML dataset. Thus, for each NDX, nine queries were applied. These queries were prepared in XQuery and XPath query languages. XPath queries were applied to Sedna and eXist, while XQuery queries were applied to BDB XML. In order to have an accurate evaluation of the query processing response times, each query was applied four times to the relevant databases and the average of the response times was used in the evaluation. Fig. 9 shows the processes involved in query processing evaluation. The simulation engine consists of three major parts - database creation and server connection, datasets insertion, and query applier responsible for applying the prepared queries to the databases. Fig. 10 shows a comparison of the performance of the three selected NDXs based on their processing response time to query1 (simple level query), query2 (moderate level query), and query3 (complex level query).

In query processing, Sedna performs very well on small and medium datasets, but eXist shows the highest performance with large datasets. Table 17 shows a comparison of the performance of NDXs in the data retrieval process on a large dataset (DBLP). The performance of BDB XML in data retrieval on the large DBLP dataset is the worst among the three NDXs. On the other hand, Sedna performs better than both eXist and BDB XML in data retrieval from a large dataset. The data retrieval processing response time is more than 10.33 times faster than BDB XML, and more 7.5 times faster than eXist. eXist retrieves data only 1.37 times faster than BDB XML.

Table 18 shows a comparison of the performances of the three NDXs in the data retrieval process from a small dataset - SigmodRecord. The performance of BDB XML in data retrieval on the SigmodRecord dataset is the worst among the three NDXs. On the other hand, Sedna is 10.33 times faster than BDB XML, and 1.56 times faster than eXist in data retrieval.
retrieval from a small dataset.

Table 19 shows a comparison of the performances of the NXDs in the data retrieval process from a medium dataset (NASA). BDB XML performs the worst in data retrieval in response to complex queries (Q2, Q3) on the medium NASA dataset when compared with the other two NXDs. But in responding to simple query (Q1), the performance of BDB XML in data retrieval is better than Sedna. eXist shows the best performance in responding to queries on the medium NASA dataset with an average data retrieval processing time that is 12.52 times faster than Sedna, and 44.14 times faster than BDB XML. On the other hand, the average data retrieval processing time of Sedna is 3.52 times faster than that of BDB XML.

9. Analysis of the results

Based on the results of the evaluation, we can conclude that the use of manual indexing and text-based storage strategy in a NXD like BDB XML can influence its performance. In the database creation process, the schema-based NXD, Sedna, achieves the highest performance. eXist, which is a model-based NXD, shows the worst performance compared to Sedna and BDB XML, while BDB XML, which is a text-based technique, shows better performance than eXist. In query processing evaluation of large and small datasets, Sedna achieves the highest performance, while in the evaluation of medium datasets, eXist shows better performance than the other two NXDs. In a comparison of the total time taken for query processing, eXist shows the best performance, followed by Sedna. BDB XML performs well in processing a simple query on a medium dataset. However, there is a wide gap in performance between BDB XML and both Sedna and eXist.

Selecting an appropriate NXD to use depends on the specific needs or situations [24]. The performance of schema-based NXDs in database creation, data insertion, and query processing is reasonably acceptable compared to the other two NXDs, which use different storage strategies. The text-based storage technique is recommended for a small document and also for storing and retrieving XML documents without changing their native format. Moreover, the result shows that a NXD, which uses a model-based storage technique, achieves the highest performance when processing medium datasets, when compared to text-based and schema-based NXDs. However, in situations where it is necessary to store and manipulate a large dataset, a schema-based NXD like Sedna is recommended.

10. Conclusion

Based on the key features of the selected NXDs, the manual indexing
method in BDB XML and also its use of text-based storage strategy influence its performance. In data insertion, the average response time of the NXD which uses schema-based storage strategy is 3.56 times faster than that of the NXD which uses a model-based storage strategy, and 2.97 times faster than that of the NXD which uses text-based storage strategy. In query processing of large and small datasets, the performance of schema-based Sedna NXD is 10.33 times faster than text-based storage strategy, and 7.5 times faster than that of the NXD which uses model-based storage technique. But for medium datasets, the model-based NXD, eXist, performs 12.25 times faster than Sedna, and 44.14 times faster than BDB XML. Also, the average query processing time of Sedna is 3.52 times faster than BDB XML. In making an overall comparison, the time taken to process all queries, the performance of the model-based NXD, eXist, is better than both BDB XML and Sedna that use text-based and schema-based storage strategies, respectively. In the overall comparison, the schema-based NXD, Sedna, is 4.88 times faster than BDB XML which uses text-based storage. Although the performance of the text-based BDB XML in processing a simple query on a medium dataset has improved, there is still a wide gap in the performance between the text-based NXD as well as the schema-based NXD and the model-based NXD.

Table 17
Performance of NXDs in data retrieval of a large dataset (DBLP).

<table>
<thead>
<tr>
<th>Storage strategy</th>
<th>NXD</th>
<th>Large (DBLP) dataset response time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>Q2</td>
</tr>
<tr>
<td>Text-based</td>
<td>Berkeley DB</td>
<td>1365 Lowest</td>
</tr>
<tr>
<td>Model-based</td>
<td>eXist</td>
<td>170 Middle</td>
</tr>
<tr>
<td>Schema-based</td>
<td>Sedna</td>
<td>43.75 Highest</td>
</tr>
</tbody>
</table>

Table 18
Performance of NXDs in data retrieval from a small dataset (SigmodRecord).

<table>
<thead>
<tr>
<th>Storage strategy</th>
<th>NXD</th>
<th>Small (SigmodRecord) dataset response time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>Q2</td>
</tr>
<tr>
<td>Text-based</td>
<td>Berkeley DB XML</td>
<td>259 Lowest</td>
</tr>
<tr>
<td>Model-based</td>
<td>eXist</td>
<td>25 Highest</td>
</tr>
<tr>
<td>Schema-based</td>
<td>Sedna</td>
<td>36 Middle</td>
</tr>
</tbody>
</table>
Table 19

<table>
<thead>
<tr>
<th>Storage strategy</th>
<th>NXD</th>
<th>NASA dataset data retrieval processing time (milliseconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
<td>Q2</td>
</tr>
<tr>
<td>Text-based</td>
<td>Berkeley DB</td>
<td>1707 Middle</td>
</tr>
<tr>
<td>Model-based</td>
<td>eXist</td>
<td>62 Highest</td>
</tr>
<tr>
<td>Schema-based</td>
<td>Sedna</td>
<td>1811 Lowest</td>
</tr>
</tbody>
</table>

Acknowledgment

This work is supported by Malaysian Ministry of Higher Education under University Malaya Research Grant (UMRG) – Project/Program UM.0000168/HRU.RP.IT and RP029D-14AE1.

References