On Indexing in Native XML Database Systems*

Pavel Loupal[1], Aleš Kantor[1], Ondřej Macek[2], and Pavel Strnad[2]

[1] Department of Software Engineering
Faculty of Information Technology, Czech Technical University in Prague
Czech Republic
loupalp@fit.cvut.cz, kantoale@fit.cvut.cz
[2] Department of Computer Science and Engineering
Faculty of Electrical Engineering, Czech Technical University in Prague
Czech Republic
macekond@fel.cvut.cz, pavel.strnad@fel.cvut.cz

Abstract. Database indices are fundamental data structures that improve the speed of data retrieval operations. In this paper, we focus on native XML database systems and provide an elementary survey of existing approaches for indexing semistructured data employed in selected academic open-source systems. Considering the requirements set for a particular system, ExDB, and the results of the accomplished research, we provide a design proposal of the indexing facility and discuss the properties of the solution we plan to subsequently realize.

1 Introduction

Native XML database management systems (NXDs) are nowadays a promising sort of document-based systems oriented on semistructured data. With the growing amount of XML data available it is essential to provide systems that can still process increased workloads efficiently. It is a fairly obvious challenge that is addressed by many research teams working on various aspects of data management. As a consequence of this situation there is a huge variety of existing algorithms and their prospective implementations in production-quality systems. To clarify the purpose and contribution of this submission let us first identify our position in this space and depict the issues we aim to address.

Our effort is driven by the endeavour to design and develop an indexing module in the ExDB system [7] that is being developed within our research group. Thus, this paper reflects the approach how to achieve this goal as a software engineering task. First, we depict here the theoretical background related to indexing (naturally only in a conceptual overview) to get acquainted with existing methods. The next step is to identify some of existing systems that might offer a useful real-world experience. The selection of presented systems we have made

* This work was partially supported by the Czech Technical University in Prague, grant no. SGS10/226/OHK3/2T/18 and by the grant project of the Czech Grant Agency (GAČR) No. GA201/09/0990.

J. Pokorný, V. Snášel, K. Richta (Eds.): Dateso 2012, pp. 127–134, ISBN 978-80-7378-171-2.

is not random; we have decided to include those claiming to offer distinct indexing facilities and which are regarded as stable products. We have already had a positive experience with some of them from our past experiments. An additional condition was also the source code availability for potential detailed exploration.

Upon the comparison of existing open-source products we can then provide a design proposal how to construct the indexing module in ExDB according to the requirements we have set. Subsequently, we discuss potential influence of this newly built module to operation of the database system. The final evaluation of the proposal will be naturally available after the implementation and adequate benchmarking.

Related Work. There are loads of papers and books focused on database systems and on related particular problems. Here we highlight only the most important resources for us. The general theoretical foundations required for the work are sufficiently covered by well-known "database Bibles", by Date [3], and Ramakrishnan [10]. Some internals of the systems we discuss later in this paper can be found on respective project homepages (i.e., for BaseX [4], eXist [9], Sedna [1], and CellStore [11]) – either by reading the documentation provided or by accessing their source codes.

2 Native XML Database Systems

Apparently, the most natural way of storing XML documents is to employ a native XML database system (NXD). The term itself is nevertheless understood differently by various groups. For our purposes we consider the XML:DB initiative definition [13]: a NXD database utilizes an (arbitrary) logical model for an XML document, as opposed to the data in that document, and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. The system then considers such XML document as its fundamental unit of (logical) storage (but, obviously, may employ an arbitrary physical storage model). To distinguish from so called XML-enabled databases, we require an NXD to be freshly grown-up upon the XML technology and not to benefit from facilities available in an existing (e.g., either relational or object) database system.

2.1 Selected Current NXDs & Feature Survey

To gain some experience with existing products we have selected few products that seem to offer appropriate and helpful view into the world of production-quality open-source systems and are initially originated in the academic environment. We try to study their internals and assess the particular findings to learn the best from it. There are two key decision points to be made in order to obtain valuable and beneficial information – *which systems* to examine and *what criteria* to consider – from such comparison.

Into the list of investigated systems we have selected those that we consider as potential competitors to our systems (CellStore, ExDB), i.e. open-source products grown-up in academic environment that are in active development and have a certain track of public releases. Hence, we have picked *BaseX*, *eXist* and *Sedna*.

To select the criteria most relevant to indexing is a more difficult problem with respect to the complexity of database management systems (in general). For our purposes we focus mainly on the following areas: supported types of indices along with their configuration options, utilized numbering schemas, involvement of available indices in query processing and space consumption (either by database or index). If any additional and beneficial properties have been identified then they are naturally included in this section, too.

BaseX [4] claims to be a light-weight, high-performance and scalable NXD. It is written completely in Java and shall be thus available on all supported platforms. The system supports XPath and XQuery query languages with almost complete coverage of the XQuery Test Suite (99.9 %). For client applications, provides the most of the APIs utilized nowadays – REST, WebDAV, XML:DB and XQJ.

The product package contains both server part and GUI client. There are two ways how to utilize the suite – either in client/server architecture (the most common deployment scenario) or (locally) as an embedded database. Undoubtedly, the supplied GUI client is the best one from all systems mentioned in this paper. It is user-friendly and offers many ways how to look on data stored at the server. Moreover, it provides also valuable statistical reports exposing interesting internals such as index configuration parameters, index size or detailed query execution plans.

Internally, the system supports a (1) structural index (Path Summary Index) and (2) value indices (text and attribute indices) and a (3) full-text index. All of these can be independently turned on/off and (3) can be moreover configured in detail. Particular employment of these indices can be tracked in execution plans for queries executed within the GUI client.

eXist [9] is another Java-based NXD that, in contrast to other products, depends heavily on several external components, e.g. from the Apache Foundation, such as Xerces and Xalan. The system supports almost all relevant query languages – XPath 2.0, XQuery 1.0 and XSLT (1.0 + 2.0). The XQuery compliance is slightly lower than for BaseX (99.4 %).

Although the documentation of the system's internals is very sparse we can observe that the vast majority of work has been done (at least in the recent time) in the field of numbering schemas and indexing concepts. According to [8] there are two node ID identification schemes implemented – Level-Order Numbering (LON) and preferred Dynamic Level Numbering (DLN). The LON uses a simple arithmetic computation to determine the relationship between two given nodes, therefore the algorithm works well for all XPath axes (on the contrary, such algorithm is not update friendly and there exists a document size limit due to

existing number of available IDs). The DLN is based on decimal classification and removes thus the disadvantages of the former one.

Using these schemas there are various (built-in or optional) indices available. The modularized design of the indexing subsystem easily allows to plug in a new index and attach it to the indexing pipeline. Supported built-in indices are basically a B+-Tree based *Structural Index* that is created by default for each element or attribute in a document and a *Range Index* (able to directly select nodes based on their typed values and applied when comparing nodes by way of standard XPath operators and functions, e.g. $=, >, <$). Pre-packaged optional indices are the *Spatial Index*, *N-Gram Index* and a *Full-text Index* (realized by the Apache Lucene engine).

Configuration of indices in eXist is accomplished by collection-specific configuration files (stored in special path with `.xconf` extension). The system does not index any element or attribute values by default therefore the configuration is needed. Subsequently all indices are automatically maintained and updated as necessary (according to all modification operations performed).

Sedna [1] is an NXD written in C++ aiming to provide "schema-based clustering storage efficient for querying and updating". This motto has been only partially confirmed by our benchmark (will be published in detail at the conference) as the storage size grew too steeply and the execution times did not overcome its competitors considerably. Such results might even re-swirl a discussion on C++ vs. Java environment efficiency.

The system is available on all major operating systems (Windows, Linux, FreeBSD, MacOS) and comprises of several command-line programs. This approach differs from all the other systems and makes the use of provided tools a bit more difficult (at least at the first glance). On the other side, there are API drivers available for a really wide variety of languages (Java, C, PHP, Python, Ruby, Perl, Delphi, C#) and XQJ and XML:DB drivers for Java. According to documentation provided the only query language supported is XQuery 1.0 (with the coverage confirmed by the XQuery Test Suite to 98.8 %).

Sedna provides two kinds of indices – value (to index XML element content and attribute values) and full-text index. In the current version, however, the query executor *does not* use these indices automatically, but it is necessary to explicitly use the respective XQuery functions `index-scan`, `index-scan-between`, and `ftindex-scan`. Value indices can be stored is either B+-tree or Block String Trie (BST). The latter option is an experimental feature that should provide more space-efficient alternative to B+-tree with the same search speed.

CellStore The main goal of the CellStore project [11] is to develop an NXD for both educational and research purposes. It is meant rather as an experimental platform than an in-box and ready-to-use database engine. We planned such an engine because the students can easily look inside it, understand and create new components for this engine as, e.g., a built-in XSLT engine, a query opti-

mizer, an index engine or an event-condition-action (ECA) processing. CellStore is developed in Smalltalk/X.

System's *architecture* is depicted in Figure 1. It can be approached through several interfaces at different levels of services. The lowest layer, low level storage, consists of several cooperating modules. Modules depicted in solid boxes are already implemented, whereas modules in dotted boxes are not ready yet.

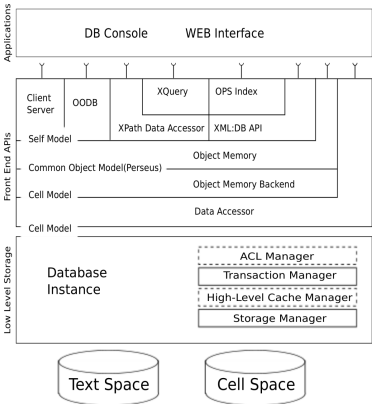

Fig. 1. *CellStore* Architecture

The system currently uses only the Path Summary Index (particular design and benchmarking depicted in [2]), which is a structural index very similar to the C–Tree index structure. As the CellStore has a specific internal structure based on the Self Model [12] the common indexing structures are not directly applicable. It is an issue to be addressed by future developments.

3 ExDB

ExDB [7] is an NXD being developed as a student research project at our university. Its primary goal is to prototype a working database environment in Java based upon the XML-λ Framework, a functional framework for XML, and thus confirm its suitability for such use case. The framework and related research activities are described in detail in [6].

Currently, it allows to persist data in either filesystem-based or native storage. XML data can be queried by XPath, XQuery and XML-λ languages. The weak point of the present solution is the non-existence of any indexing facility. Although we have investigated some potential options and proposed a solution ([5]) yet there is no indexing support available. Such shortcoming naturally prevents the system from performing efficiently for any query-based workload.

3.1 Indexing-related Requirements

The overall goal of our effort is to design and develop a configurable and extensible indexing module for the ExDB system. This module should be in charge of all documents indices and should provide access to them.

According to the previous survey we have identified fundamental requirements listed in short as follows. The design proposal should be

- *configurable* in various directions
  - enabling general setup of the indexing subsystem
  - supporting database, collection or document level configuration
  - allowing alternative physical storage structure approach – either clustered by index type or by collection hierarchy
- *extensible* for future improvements and modifications
- supporting *multiple index types*, generally both value, structure and full-text
- offering *automatic index update*
- open to *multiple query languages* – actually for XQuery and XML-λ
- helping with *query cost estimation*

Moreover, with respect to the nature of the ExDB project, we need to be able to select some of these requirements and build an prototype within a few months (till the end of current term). The remaining part is to be done subsequently.

3.2 Analysis

The requirements stated in the previous section cover a very wide range of particular sub-items that shall be analyzed in detail. In order to cover the most important ones and due to limited space available we focus here only on the major issues. Basically, with all these requirements in mind we propose a design addressing the key areas and furthermore attempt to take into account also those that remain for the future work. There are three important parts that need to be examined at first – module *configuration*, its *storage strategy* and *query interface*.

The configuration of the indexing module will obviously control its behavior and the scope of features available. Technically, there already is a configuration facility within the system and thus no additional extension is necessary. So far, we have identified about 20 parameters that could be used for setting up the module and its activities. For its length we do not publish it here in depth.

For persisting indices we need to extend current Storage module. Principally, retrieving and storing does not differ much from working with XML data and is thus not too complicated. The only doubt is the physical structure of the data – there are a few different approaches that might affect the efficiency of read/write operations – particular indices (of distinct types) can be stored in separate operating system files according to database collection hierarchy or in one "big" file all-together. One might consider also a hybrid approach when, for example, the full-text index for all documents in database is stored in one file

and all the remaining indices are stored separately and organized in collection-like directory hierarchy. Each approach has its pros and cons (chiefly clashing memory consumption with disk look-up time overhead) and we are not aware of any study with general and clear results. Thus, presuming the first approach – an individual per-document, per-index file – to be sufficiently suitable (i.e. still efficient and easily implementable) for our prototype.

Regarding the index-lookup interface we are confronted with a problem of using two implementations of distinct query language (XPath/XQuery and XML-λ) using different internal data model. There is a planned work on transforming XQuery queries into their XML-λ equivalent but a working transformation library seems to be still too distant. There are two alternatives how to face this problem – we can either speed up the development of the transformation tool or create a more general interface with adapters to both implementations. From our programmers experience we prefer to invest the time in the later option and put some additional effort into developing an adapter layer between indexing and querying modules.

3.3 Design Proposal

From the requirements and analytical notes stated above we have derived an outline of module design as shown in Figure 2. The diagram depicts the separation of the functionality into two logical parts, the first is the manager aiming to administer particular document indices. Its task is the creation, modification, deletion, storage and loading of indices. This component is connected to the system core from where the commands arrive. Moreover, upon its configuration, the component may automatically reindex or drop obsolete indices when necessary.

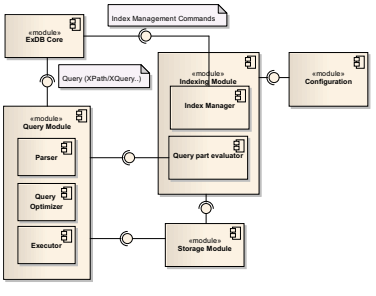

Fig. 2. Indexing Module Component Diagram

Parsing the query and consequent preparation of execution plans will be naturally the task for particular implementation of the query language. These plans are sent to the indexing module for estimating their costs. The query module then chooses the most convenient execution plan and starts to evaluate its steps it accessing either the storage or the indexing module.

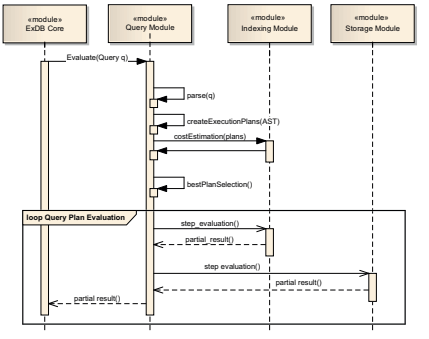

Fig. 3. Query Evaluation Process Model

4 Conclusion and Future Work

The aim of this paper was to provide an overview of existing approaches for indexing XML data available in open-source NXDs and to describe a particular design proposal for a configurable indexing module inside the ExDB system. Although we have not dipped into the problem in full detail we suppose that the text sufficiently covers the judgment of our proposal.

Our future work will include the implementation and benchmarking of the indexing module. These activities are scheduled for the following months along with implementation of a new transaction module.

References

1. K. Antipin. Sedna project homepage. `http://www.sedna.org`, 2012.
2. K. Beyr. Index implementation in CellStore project. Master's thesis, Dept. of Computer Science and Engineering, FEE CTU, Prague, 2008.
3. C. J. Date. *An Introduction to Database Systems.* Addison-Wesley, 1995.
4. C. Grün. BaseX project homepage. `http://www.basex.org`, 2012.
5. M. Janek. Indexing techniques for native XML database systems. Master's thesis, Dept. of Computer Science and Engineering, FEE CTU, Prague, 2011.
6. P. Loupal. *XML-λ : A functional framework for XML.* PhD thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, February 2010.
7. P. Loupal. ExDB project homepage. `http://exdb.fit.cvut.cz`, 2012.
8. W. Meier. Index-Driven XQuery Processing in the eXist XML Database. `http://www.xmlprague.cz/2006/slides06/meier.pdf`, 2006.
9. W. Meier. eXist project homepage. `http://exist.sourceforge.net`, 2012.
10. R. Ramakrishnan and J. Gehrke. *Database Management Systems.* McGraw-Hill Science/Engineering/Math, 3rd edition, 2002.
11. M. Valenta. CellStore project homepage. `http://swing.fit.cvut.cz/projects/cellstore`, 2012.
12. J. Vraný. CellStore - the vision of pure object database. In *DATESO*, 2006.
13. XML:DB. What is a XML database? `http://xmldb-org.sourceforge.net`, 2003.