An XML Index Advisor for DB2

Iman Elghandour[*]
University of Waterloo
Waterloo, ON, Canada
ielghand@cs.uwaterloo.ca

Ashraf Aboulnaga
University of Waterloo
Waterloo, ON, Canada
ashraf@cs.uwaterloo.ca

Daniel C. Zilio
IBM Toronto Lab
Markham, ON, Canada
zilio@ca.ibm.com

Fei Chiang[†]
University of Toronto
Toronto, ON, Canada
fchiang@cs.toronto.edu

Andrey Balmin
IBM Almaden Research Center
San Jose, CA, USA
abalmin@us.ibm.com

Kevin Beyer
IBM Almaden Research Center
San Jose, CA, USA
kbeyer@us.ibm.com

Calisto Zuzarte
IBM Toronto Lab
Markham, ON, Canada
calisto@ca.ibm.com

ABSTRACT

XML database systems are expected to handle increasingly complex queries over increasingly large and highly structured XML databases. An important problem that needs to be solved for these systems is how to choose the best set of indexes for a given workload. We have developed an XML Index Advisor that solves this XML index recommendation problem and is tightly coupled with the query optimizer of the database system. We have implemented our XML Index Advisor for DB2. In this demonstration we showcase the new query optimizer modes that we added to DB2, the index recommendation process, and the effectiveness of the recommended indexes.

Categories and Subject Descriptors

H.2 [**Database Management**]: Physical Design

General Terms

Algorithms, Design, Experimentation, Performance

Keywords

XML Databases, Automatic Physical Database Design, Index Advisor

1. INTRODUCTION

XML support has been added to most commercial relational database systems, and these systems all employ different types of structural and value XML indexes to improve performance, potentially by orders of magnitude. Users of these XML database systems need to decide which set of indexes to create for a given XML database and query workload, which is not a simple task. This task is particularly difficult for XML database systems that allow for *partial indexing* of XML documents, such as DB2 [2] and Oracle [6]. A partial index is an index on parts of an XML document that satisfy *index patterns* specified by the user. These patterns can be represented, for example, by XPath path expressions, in which case only the XML elements that are reachable by these path expressions are included in the index. However, users now face the problem of choosing the set of XML patterns to index. In this demonstration, we present an XML Index Advisor for DB2 that automatically recommends the best set of XML indexes and index patterns for a given database and query workload, while taking into account the cost of updating the index on data modification. Details about our XML Index Advisor can be found in [4].

One of the key features of our Index Advisor is that it is tightly coupled with the query optimizer of the XML database system, in our case DB2. We rely on the query optimizer to enumerate the candidate index patterns for a query, and to evaluate the benefit to a query of having a particular index configuration. This tight coupling with the query optimizer helps us leverage its index selection and cost estimation capabilities, and provides a solid and easy way for ensuring that the indexes that we recommend are actually *used* by the optimizer in its generated query execution plans. Moreover, we can easily support the different query languages supported by the optimizer (XQuery and SQL/XML in the case of DB2).

In the next section, we present a high-level overview of the extensions that we made to the query optimizer to support our XML Index Advisor. We also present an overview of the main steps of the XML index recommendation process. More details are available in [4]. In Section 3, we describe our demonstration, which showcases the extensions that we made to the DB2 query optimizer, the details of the index recommendation process and how it is controlled by the user, and the tools that we provide to analyze the index recommendations and measure their effectiveness.

[*]Supported by an IBM CAS Fellowship. Also affiliated with Alexandria University, Alexandria, Egypt.

[†]This work was done while the author was at the IBM Toronto Lab.

2. XML INDEX ADVISOR OVERVIEW

The architecture of the XML Index Advisor is presented in Figure 1. The high-level framework of the index recommendation process is as follows: First, for every query in the workload, we rely on the query optimizer to enumerate a set of candidate indexes that would be useful for this particular query. Next, we expand the

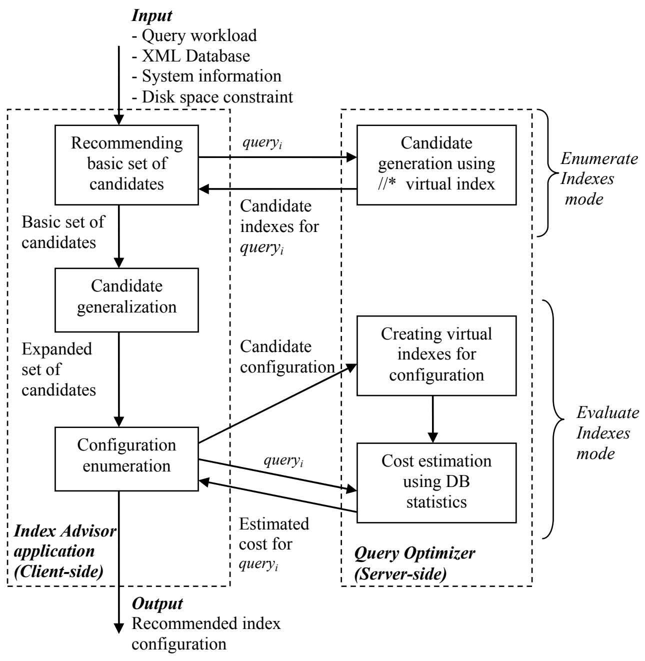

Input
- Query workload
- XML Database
- System information
- Disk space constraint
Recommending basic set of candidates
query_i
Candidate generation using //* virtual index
Enumerate Indexes mode
Candidate indexes for query_i
Basic set of candidates
Candidate generalization
Expanded set of candidates
Candidate configuration
Creating virtual indexes for configuration
Configuration enumeration
query_i
Cost estimation using DB statistics
Evaluate Indexes mode
Estimated cost for query_i
Index Advisor application (Client-side)
Query Optimizer (Server-side)
Output
Recommended index configuration

Figure 1: XML Index Advisor architecture.

enumerated set of candidate indexes to include more general indexes, each of which can potentially benefit multiple queries from the current workload or from future, yet-unseen workloads. Finally, we search the space of possible index configurations to find the optimal configuration, which is the one that maximizes the performance benefit to the workload while satisfying the disk space constraint provided by the user.

Much of the functionality of the advisor is implemented in a client-side application. However, to be able to use the query optimizer for index recommendation, we need to extend it with two new *query optimizer modes*, implemented as EXPLAIN modes in DB2. In the first mode, which we call the *Enumerate Indexes* mode, the optimizer takes a query and enumerates the indexes that can help this query. In the second mode, which we call the *Evaluate Indexes* mode, the optimizer simulates an index configuration and estimates the cost of a query under this configuration.

In the new modes, the optimizer needs to work with hypothetical indexes that do not exist, but are still needed to accomplish its task. To enable this, we modify the query optimizer to allow it to create *virtual indexes* that can then be used during query optimization. These virtual indexes are added to the database catalog and to all the internal data structures of the optimizer, but they are not physically created on disk and no data is inserted into them. Virtual indexes are used in relational index advisors to enable the optimizer to estimate the cost of candidate index configurations [8]. In our XML Index Advisor, we use virtual indexes for cost estimation, but a novel feature of our work is that we also use them for enumerating candidate indexes for workload queries.

2.1 Basic Set of Candidates

The first step of the index recommendation process is to enumerate a set of candidate indexes for each query. The DB2 query optimizer supports XQuery and SQL/XML, which are fairly complex languages. In these languages, XML patterns can appear in different parts of the query, but indexes cannot be used for some of them because of certain language features [1]. In addition, *index matching*, which is the process that decides which indexes are useful for which parts of the query, is dependent on the query optimizer implementation. Thus, it is best to tightly couple candidate enumeration with the query optimizer, which we do in our advisor. To obtain the basic set of candidate indexes that are useful for a given query, our Enumerate Indexes optimizer mode creates a virtual index with index pattern `//*` . This `//*` *virtual index* hypothetically indexes all elements in an XML document and hence can be matched with any XPath pattern in the query that can be answered using an index. The process of index matching in the optimizer determines the XML patterns in the query that match this `//*` virtual index, and we use these patterns as the basic set of candidate indexes for the current query. Essentially, we have enabled the query optimizer to answer the question: "If all possible indexes were available, which query patterns would benefit from them?" The next step in the index recommendation process is to expand this basic set of candidate indexes.

2.2 Generalizing the Candidates

The optimizer helps us identify index patterns specific to each query. However, it is unable to identify index patterns that can benefit multiple queries in the current workload and also future queries with similar patterns. To address this shortcoming of relying on the optimizer for candidate enumeration, we expand the set of candidate indexes generated by the optimizer by applying a set of *generalization rules*. These rules allow us to generate more general candidate index patterns that can be useful for multiple queries from the specific index patterns enumerated by the optimizer for individual queries. For example, assume that these two patterns appear in the input workload: `/regions/namerica/item/quantity` and `/regions/africa/item/quantity`. These two queries ask about item quantities in the North America and Africa regions. An index that includes the item quantities in all regions helps these two queries as well as other similar queries that are inquiring about item quantities in different regions. Hence, our generalization rules generate the pattern `/regions/*/item/quantity`. If the workload also contains: `/regions/samerica/item/price`, then we perform another step of generalization to generate a new pattern that indexes all items' specifications that are available in all regions: `/regions/*/item/*`. We then expand the list of candidate indexes by adding the two generated general indexes.

During generalization, we construct a *Directed Acyclic Graph* (DAG) of the candidate indexes. Each node in the DAG represents an XML pattern, and has as its parents the possible generalizations of this pattern, based on our candidate generalization algorithm. At the end of the generalization phase, we have a DAG rooted at the most general indexes that can be obtained from the workload. Figure 4 shows an example of this DAG.

2.3 Searching for the Optimal Configuration

After the candidate enumeration and generalization steps, we have in hand an expanded set of candidate indexes. We need to search the space of possible index configurations consisting of indexes from this candidate set to find the index configuration with the maximum benefit to the workload, subject to a constraint specified by the user on the disk space available for the configuration. This combinatorial search problem can be modeled as a 0/1 knapsack problem [8]. A greedy approximation of the 0/1 knapsack problem has been used in [8]. The greedy search starts with an empty configuration and adds candidate indexes to the recommended configuration until the space budget is exhausted. This greedy approximation of the 0/1 knapsack problem, as well as other search approaches (e.g. [3]) were not suitable for our XML Index Advisor. Hence, we propose two novel search strategies [4].

Our XML Index Advisor allows the user to choose between two search algorithms. The first is a greedy search augmented with heuristics to detect redundant indexes (indexes whose index patterns are already covered by other indexes) as soon as possible and reclaim the space that they use so that we can include more useful, non-redundant indexes in this reclaimed space. The second algorithm is a top down (root-to-leaf) search through the DAG constructed in the candidate generalization step. The goal of this algorithm is to recommend the most general set of indexes that fits within the available disk space budget.

Greedy Search with Heuristics. Greedy search relies only on the benefit and size of candidate indexes when selecting the recommended configuration so it can select general indexes that can be used for path expressions that are already covered by other indexes in the configuration. This can result in some indexes chosen by the advisor never being used by the optimizer. To address this index redundancy problem, we add one more objective to our search problem: maximizing the number of workload XPath path expressions that use indexes in the recommended configuration. This objective guarantees that every index recommended by the XML Index Advisor will be used by at least one query in the workload. The greedy search algorithm with heuristics maintains a bitmap of XPath patterns in the workload queries that have indexes on them. Then, before adding any general index to our configuration, we use this bitmap to make sure that this index will not be a replication of others already chosen.

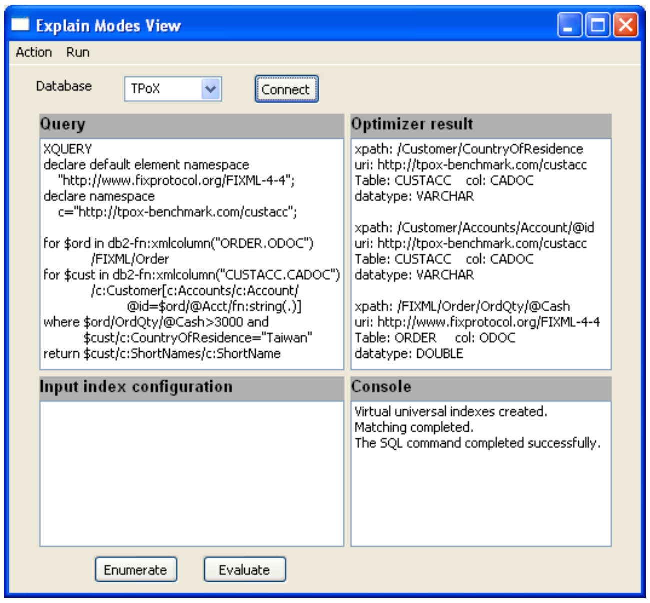
Top Down Search. The recommendations of the XML Index Advisor are highly dependent on the input workload. A possible scenario is that the DBA has assembled a representative training workload, but the actual workload may be a variation on this training workload. Thus, the workload presented to the XML Index Advisor is a representative of a larger class of possible workloads. In this case, we posit that the goal of the advisor should be to choose a set of indexes that are as general as possible while still benefit the workload queries. We start with the roots of the DAG constructed in the generalization step as our current configuration. Since general indexes are typically large in size, this starting configuration is likely to exceed the available disk space budget, but it potentially has the maximum benefit that can be achieved. We progressively replace a general index from the current configuration with its specific (and smaller) children in the DAG until the configuration that we have fits within the disk space budget.

During our greedy or top down search, we need to estimate the benefit to the workload of candidate index configurations. For this purpose, we use the query optimizer in the Evaluate Indexes mode. In this mode, the indexes in the configuration are created as virtual indexes, and the queries in the workload are optimized with these indexes in place to measure the improvement in estimated cost. When estimating a configuration benefit, we take into account that the benefit of an index can change depending on which other indexes are available (index interaction).

3. DEMONSTRATION

Our demonstration illustrates: (1) the two new EXPLAIN modes that we added to the DB2 query optimizer, (2) the steps of the index recommendation process and how it can be controlled by the user, and (3) the effectiveness of the index configurations recommended by our XML Index Advisor. The demonstration uses XQuery and SQL/XML queries on XML data from standard benchmarks such as XMark [7] and TPoX [5]. The workloads used consist of the standard benchmark queries augmented with synthetic queries. Users can also specify additional queries.

The first part of the demonstration illustrates the two new EXPLAIN modes that we added to the DB2 query optimizer. We use a visual client to demonstrate the following two scenarios:



```
XQUERY
declare default element namespace
   "http://www.fixprotocol.org/FIXML-4-4";
declare namespace
   c="http://tpox-benchmark.com/custacc";

for $ord in db2-fn:xmlcolumn("ORDER.ODOC")
         /FIXML/Order
for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")
         /c:Customer[c:Accounts/c:Account/
                @id=$ord/@Acct/fn:string(.)]
where $ord/OrdQty/@Cash>3000 and
        $cust/c:CountryOfResidence="Taiwan"
return $cust/c:ShortNames/c:ShortName
```

```
xpath: /Customer/CountryOfResidence
uri: http://tpox-benchmark.com/custacc
Table: CUSTACC    col: CADOC
datatype: VARCHAR

xpath: /Customer/Accounts/Account/@id
uri: http://tpox-benchmark.com/custacc
Table: CUSTACC    col: CADOC
datatype: VARCHAR

xpath: /FIXML/Order/OrdQty/@Cash
uri: http://www.fixprotocol.org/FIXML-4-4
Table: ORDER     col: ODOC
datatype: DOUBLE
```

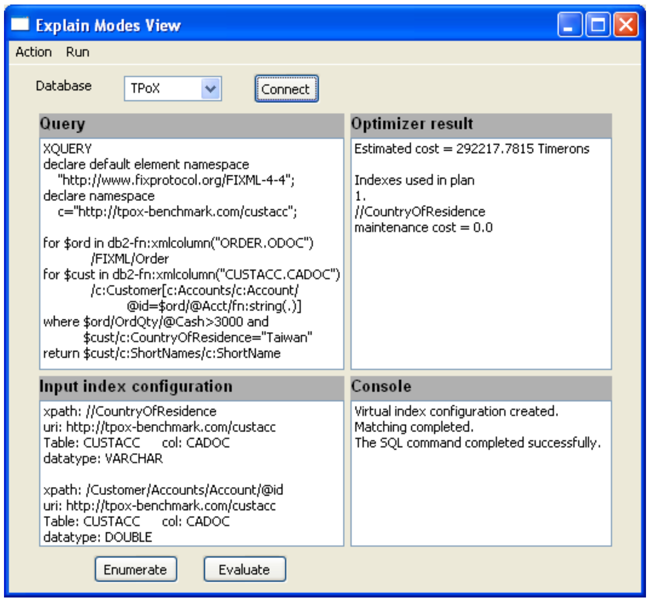
```
Virtual universal indexes created.
Matching completed.
The SQL command completed successfully.
```

Figure 2: Basic candidate recommendation.

- Given an XML query (XQuery or SQL/XML), we invoke the optimizer in the Enumerate Indexes mode to generate the basic set of candidate indexes for this query (Figure 2).
- Given an XML query and an index configuration consisting of a set of XML index patterns, we invoke the optimizer in the Evaluate Indexes mode to estimate the cost of the query for the given index configuration (Figure 3).

The second part of the demonstration focuses on the functionality of our XML Index Advisor and the effectiveness of its recommendations. We demonstrate the following aspects:

- For an input workload, we show the basic set of candidate indexes and the DAG that represents the generalized candidate indexes. This allows us to see the relationship between the basic and generalized index patterns (Figure 4).
- We show how the two proposed search algorithms traverse the generalization DAG to find the optimal configuration that fits within the disk space budget (Figure 4).
- We provide the user with the ability to analyze the recommendations of the XML Index Advisor. The recommendation analysis feature allows the user to graphically compare three estimated costs for each query in the workload: (1) the original cost with no indexes, (2) the cost with the index configuration recommended by the advisor, and (3) the cost with an index configuration consisting of all the basic candidate indexes enumerated by the advisor for the input workload. This last configuration is "overtrained" for the input workload and may be larger than the available disk space budget, but represents the maximum benefit that we can achieve for the given workload. The tool also allows the user to add more queries beyond the input workload and evaluate the benefit of the recommended configuration to these queries. This will illustrate the benefit of recommending generalized index configurations. The tool also allows the user to modify the recommended configuration by adding and removing



Database: TPoX

Query

```
XQUERY
declare default element namespace
    "http://www.fixprotocol.org/FIXML-4-4";
declare namespace
    c="http://tpox-benchmark.com/custacc";

for $ord in db2-fn:xmlcolumn("ORDER.ODOC")
            /FIXML/Order
for $cust in db2-fn:xmlcolumn("CUSTACC.CADOC")
            /c:Customer[c:Accounts/c:Account/
                    @id=$ord/@Acct/fn:string(.)]
where $ord/OrdQty/@Cash>3000 and
          $cust/c:CountryOfResidence="Taiwan"
return $cust/c:ShortNames/c:ShortName
```

Optimizer result

```
Estimated cost = 292217.7815 Timerons

Indexes used in plan
1.
//CountryOfResidence
maintenance cost = 0.0
```

Input index configuration

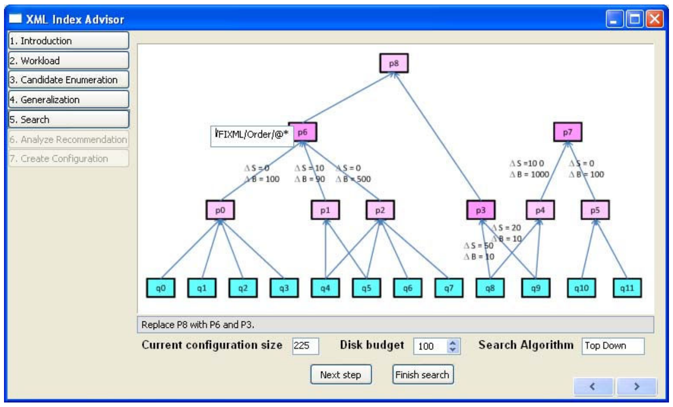
```
xpath: //CountryOfResidence
uri: http://tpox-benchmark.com/custacc
Table: CUSTACC     col: CADOC
datatype: VARCHAR

xpath: /Customer/Accounts/Account/@id
uri: http://tpox-benchmark.com/custacc
Table: CUSTACC     col: CADOC
datatype: DOUBLE
```

Console

```
Virtual index configuration created.
Matching completed.
The SQL command completed successfully.
```

Figure 3: Estimating the benefit of an index configuration.



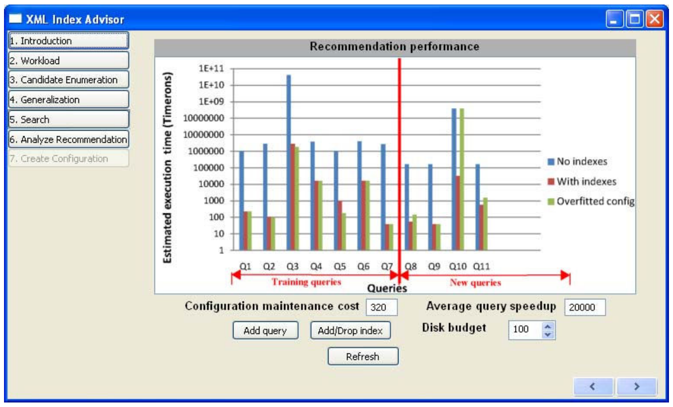
Replace P8 with P6 and P3.

Current configuration size: 225 Disk budget: 100 Search Algorithm: Top Down

Figure 4: Searching the space of candidate indexes.

indexes and to see the effect of these modifications on query performance (Figure 5).

- Finally, the tool allows the user to review the final recommended index configuration and to create it. The actual execution time taken by the queries can then be displayed.



Configuration maintenance cost: 320 Average query speedup: 20000 Disk budget: 100

Figure 5: Analyzing the XML Index Advisor recommendations.

4. REFERENCES

[1] A. Balmin, K. S. Beyer, F. Özcan, and M. Nicola. On the path to efficient XML queries. 2006.

[2] K. Beyer et al. DB2 goes hybrid: Integrating native XML and XQuery with relational data and SQL. *IBM Systems Journal*, 45(2), 2006.

[3] N. Bruno and S. Chaudhuri. Automatic physical database tuning: a relaxation-based approach. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2005.

[4] I. Elghandour, A. Aboulnaga, D. C. Zilio, F. Chiang, A. Balmin, K. Beyer, and C. Zuzarte. XML index recommendation with tight optimizer coupling. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, 2008.

[5] M. Nicola, I. Kogan, and B. Schiefer. An XML transaction processing benchmark. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2007. Benchmark Available at: `https://sourceforge.net/projects/tpox/`.

[6] Oracle Corp. *Oracle Database 11g Release 1 XML DB Developer's Guide*, 2007. Available at: `http://www.oracle.com/pls/db111/homepage`.

[7] A. R. Schmidt et al. The XML benchmark project. Technical Report INS-R0103, CWI, 2001.

[8] G. Valentin, M. Zuliani, D. C. Zilio, G. Lohman, and A. Skelley. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *Proc. IEEE Int. Conf. on Data Engineering (ICDE)*, 2000.