R-SOX: Runtime Semantic Query Optimization over XML Streams

Song Wang, Hong Su, Ming Li, Mingzhu Wei, Shoushen Yang, Drew Ditto
Elke A. Rundensteiner and Murali Mani
Department of Computer Science, Worcester Polytechnic Institute
Worcester, MA 01609-2280, USA
(songwang|suhong|minglee|samanwei|shoushen|dditto|rundenst|mmani)@cs.wpi.edu

1 Motivation

Using schema knowledge to optimize query evaluation, known as semantic query optimization (SQO), has generated promising results in XML query processing [2, 3, 7]. In XML stream processing, we can use the schema constraints to expedite the traversal of the streams and to minimize memory consumption for holding the intermediate data during query evaluation. These are particularly critical for stream applications, which require real-time responses and both typically operate in limited main memory. However, as illustrated by the motivating scenarios below, these prior techniques assume that the XML schema is static and is available prior to the start of the query execution [2, 3, 7]. As the scenario below highlights this assumption is unrealistic and thus may render existing techniques non practical.

Case Study 1: Assume that in a news publishing (or dissemination) scenario, the news server retrieves news from a large number of multiple sources, such as different reporter devices, different broadcast agencies, and government sources and disseminates such heterogeneous messages as an XML stream to subscribers. Such sources may disagree with each other on some aspects of the schema. To provide a uniform interface to the downstream receiver, the stream server may pre-define an output XML schema. Such schema must be "coarse" enough so that all XML messages in the stream do conform to it. This universal schema is likely to be rather coarse, if the diversity of sources is large, as it must be the lowest common denominator of the features shared across all sources. The schema will contain huge optional elements and alternative subtypes, thus becoming less amendable for schema based optimization.

Case Study 2: In an online auction stream, auction items can be rather different over time. Maybe only a few core attributes, like price and expiration date, stay the same. Beyond these attributes, different sellers of items can introduce properties to describe their items at will. The stream server should be able to capture such schema refinements and provide a runtime schema to the stream receiver interleaved with the data stream. For instance, when one person or company who sells PCs happens to submit 200 laptops, the stream server can provide a refined schema to the stream receiver, which is valid only for the next 200 XML messages from that seller.

From the above two case studies, we observe that we need the ability to specify dynamic schema changes at runtime and utilize these refinements to perform not just static but run time SQO.

Our R-SOX Solution. Our proposed system *R-SOX* (**R**untime **S**emantic query **O**ptimization over **X**ML streams) is the first such system designed to tackle the above identified challenges. R-SOX efficiently evaluates XQuery expressions over highly dynamic XML streams. The schema can switch from optional to mandatory types, from potentially deep structures to shallow ones, or from recursive to non-recursive types.

In R-SOX, the dynamic schema changes are embedded within the XML stream via punctuation. The stream receiver then will exploit semantic optimization opportunities and provide the output stream in real-time using an optimized processing time and memory footprint, by short-cutting computation when possible and releasing buffer data at the earliest.

State-of-the-Art. YFilter [9] includes a type inference technique using schema knowledge to decide whether results of a pattern are recursion-free. However, it cannot be used at run-time. XHints [1] extends SIX by supporting predicates and online index generation using only partially buffered streams. R-SOX instead focuses on using embed-

This project is partially supported by NSF, under grants IIS 0414567, and CNS 0551584

VLDB '06, September 12-15, 2006, Seoul, Korea.

ded schema knowledge to speed up logical level pattern retrieval. In practice, these techniques are complimentary and could be combined to achieve better performance. Our R-SOX system, built with Raindrop [6, 4, 7, 5] as its query engine kernel, now can specify runtime schema refinements and perform a variety of runtime SQO strategies for query optimization.

Contributions of R-SOX include:

1. We design techniques that adaptively invoke multi-mode operators for efficient processing of recursive pattern queries on potentially recursive data guided by run-time schema.
2. We apply the early filtering techniques dynamically to avoid unnecessary computations on pattern retrieval to now be driven by runtime schema knowledge.
3. We put forward a novel technique, called unblocking data output, which avoids unnecessary data buffering thus maintaining a minimized memory footprint.
4. For changes of the plan at run time, we design techniques for safe migration by adjusting the transitions in automaton and associated plan execution controls.

2 R-SOX System

The architecture of the R-SOX system is described in Figure 1. The input XML streams are annotated by the stream sender with *RSI*s (Runtime Schema Information). The *Stream Loader* extracts these RSIs from the input stream, and the *Schema Knowledge Manager* maintains the runtime schema knowledge over time according to the RSIs.

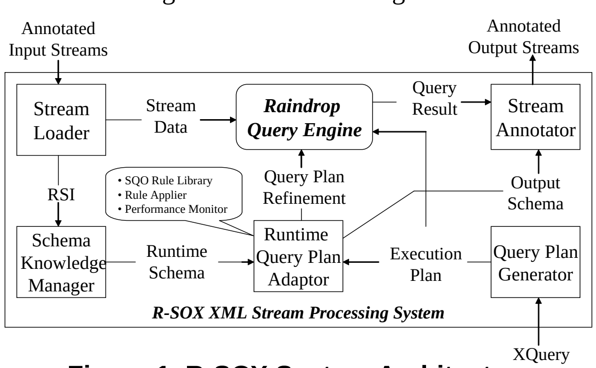

Figure 1. R-SOX System Architecture

The user XQuery is parsed and translated into a stream execution plan by the *Query Plan Generator*. The *Runtime Query Plan Adaptor* collects runtime schema knowledge, performs online semantic query optimization and incremental query plan migration. The output schema is inferred by the runtime query plan adaptor based on the updated schema. This output schema is propagated to the *Stream Annotator*, which will annotate the query result generated by the *Raindrop Query Engine* with output RSIs.

3 Runtime Schema Management

Designing the Runtime Schema Model. We now briefly describe the dynamic schema punctuation model we have designed to interleave schema change metadata into XML data streams, called runtime schema information (RSI). RSIs indicate schema changes applicable for all subsequent XML elements in the stream until when the schema change expires or is overwritten by a later RSI. RSIs are sent along with other XML messages in the stream as punctuations.

RSI	::=	$Scope,Target,Action$
$Scope$	::=	$ScopeType,ScopeLength,ScopeLenType$
$ScopeType$	::=	$xpath$
$ScopeLength$	::=	$integer\|inf$
$ScopeLenType$	::=	$TIME\|COUNT$
$Target$	::=	$TargetType,TargetPosType,TargetCard$
$TargetType$	::=	$xpath$
$TargetPosType$	::=	$xpath\|null$
$TargetCard$	::=	$*\|+\|?\|(min,max)\|null$
$Action$	::=	$+\|-\|R$

Figure 2. Grammar of the RSI

RSI contains information for schema knowledge construction and updating. The grammar of the RSI is sketched in Figure 2. The following example RSIs are defined over the schema of element type $news$ based on the grammar.

`S1:  <!ELEMENT news (source?, (paragraph|comment))>`

RSI1: ((/news, inf, TIME), (/news/comment, ,), -)

`S2: <!ELEMENT news (source?, paragraph)>`

*RSI2: ((/news, 200, COUNT), (/news/category, /news/paragraph, *), +)*

`S3: <!ELEMENT news (source?, paragraph, category*)>`

$RSI1$ on current schema $S1$ denotes the change that the stream will not have any $comment$ element for future $news$ nodes. The runtime schema after arrival of $RSI1$ will be $S2$. $RSI2$ says that $category*$ will appear after the node type $paragraph$ for the subsequent 200 $news$ nodes. The runtime schema is changed correspondingly to $S3$.

Building the Runtime Schema. Similar to other projects, we model the runtime schema as a directed ordered graph. R-SOX maintains the current schema graph incrementally by synchronizing it with newly arriving RSIs.

By the example above, RSI *S2* indicates that the change on $news$ will be expires after 200 $news$ nodes. At that time, we need to roll back the change. However, we cannot simply roll back to the previous version of the schema graph because other RSIs may already have been installed in the mean time on the schema graph after this RSI. If we were to blindly apply the delta change in reverse, like adding the $category$ node back to $news$, it is possible that the $news$ node may not exist any more. R-SOX offers the schema management based on schema version with reversable delta changes augmented by change dependencies.

4 Runtime Query Optimization Strategies

We now highlight some of the semantic query optimization(SQO) strategies used by our run time optimizer. We now apply query optimization strategies whenever the schema changes. Thus the system has to perform plan migration after the query optimization.

Run-time Plan Migration Strategy. When the schema changes, a new query plan will be generated by optimizer. In traditional stream systems, it is safe to drain out all existing tuples in the middle operators if operators are stateless. However, this is not the case for XML streams. The buffer in the middle operators in the plan may contain partial elements. So we could be corrupting the results if migration is not done carefully.

The algebra plan change can also negatively effect the automaton. Since the query plan is changed, the patterns to retrieve by automaton may have to be changed as well. For this, we identify safe moments for migration and then remove appropriate transitions from states and adjust the automaton stack if needed.

Processing Recursive Types. Recursive types will make the descendent pattern retrieval ("//") in XPath more complex and thus resource intensive. For a input stream having recursive schema, RSIs can be used by the stream server to indicate the existence of recursion for data fragments or indicate the depth of the recursion level. For instance, if RSIs indicate the data fragment is recursive, we will apply recursive mode algebra operators in the query plan that maintain and associate ID information with each element. We must perform ID based comparison in the downstream join operator to obtain correct results. If RSIs indicate the data fragment is not recursive, non-recursive mode algebra operators which do not need to perform ID comparison in the downstream join operatorss are invoked at runtime. Thus both the memory and computation cost can be saved when we use RSIs to indicate the recursion information about these elements [8].

Early Filtering. Dynamic SQO in R-SOX utilizes early detection of failed predicates. If within a binding of $\$v$, results of p_α must all occur before any of p_β, we say a result of p_β is an *ending mark* of p_α. We can test the predicates of p_α earlier as soon as we see an occurrence of p_β, without waiting for the end tag of $\$v$. This failure test will invoke skipping the evaluation of all the other XPaths within this binding. R-SOX supports dynamic SQO rules utilizing ordering, occurrence, or exclusive constraints in the XML schema. While in our earlier work [7] we supported static optimization, we now have enhanced these techniques to be triggered by RSIs at runtime. Let's consider the example shown in Query 1, which asks for the $paragraph$ and $comment$ information under the $news$ element with state name "MA".

```
Query 1: for $p in /news_stream/news
         where $p/state = ''MA''
         return $p/paragraph, $p/comment
```

Suppose that the runtime schema for the current $\$p$ binding has been refined by RSIs from Schema S_4 to S_5:

```
S4: <!ELEMENT news (source, nation?, state?,
     (paragraph | comment | category)*)>
S5: <!ELEMENT news (source, (nation | state),
      paragraph*, comment*, category*)>
```

Both the exclusive and ordering constraints can be used to achieve the early filtering optimization in Query 1. The ordering constraint indicates that $\$p/paragraph$, $\$p/comment$ and $\$p/category$ are candidate ending marks for the XPath $\$p/state$. The exclusive constraint between the $nation$ and $state$ provides another possible ending mark for the $\$p/state$.

Unblocking Data Output. In the query execution, operators need to wait for the completeness of the whole bound pattern before passing data up to the output because some predicates may not yet be satisfied or the extracted patterns need to be output according to a specified sequence. The plan rewriting algorithm of R-SOX can avoid such data holding by early detection of successful predicates and switching the output mode of related operators. This optimization is called *unblocking data output*.We perform this optimization by: (a) checking predicates earlier and (b) ensuring the elements satisfy the output sequence.

Consider the example shown in Query 1 that outputs the $paragraph$ and $comment$ lists for each $news$ element while the predicate on $state$ has been satisfied. Assume we check the predicate early and the runtime schema for the current binding $news$ is refined by RSIs from S_4 to S_5. Under S_4, we need to hold at least the $comment$ list because the output sequence requires the $paragraph$ list to be returned before the $comment$ list. The refined schema S_5 provides order constraint between $paragraph$ and $comment$. Now we need not hold any $paragraph$ or $comment$ once the predicate is satisfied.

Sometimes data holding cannot be avoided because the available constraint information is not sufficient. For instance, consider the schema is refined from S_4 to S_6:

```
S6: <!ELEMENT news (source, (nation | state),
     (paragraph | comment)*, (category | comment)*)>
```

In this case, we do not have enough schema information to remove the data holding of $comments$. However $category$ could now serve as the ending mark of the $paragraph$. Therefore, when we see the first $category$, all the extracted $comment$s can be output.

5 Demonstration Focus

The prototype of R-SOX has been implemented using Java with the Raindrop as its core query engine [5]. We use an online auction monitoring as one of the example applications in our demonstration. Steps shown include:

Plan Visualization Tool. R-SOX parses the XQuery and generates automaton-algebra stream plans. Our visual tool allows viewers to explore the plans.

Runtime Schema Management. Figure 3 depicts an incrementally maintained schema graph. The left shows the RSIs received and the right represents the current schema knowledge for one particular auction data input. The highlighted node is to be deleted according to the new RSI.

Runtime Plan Migration. With updated schema knowledge, our query plan refinement will update the query plan incrementally using R-SOX's runtime SQO techniques. Figure 4 depicts as one representative example the adapted execution plan showing runtime computation shortcuts by

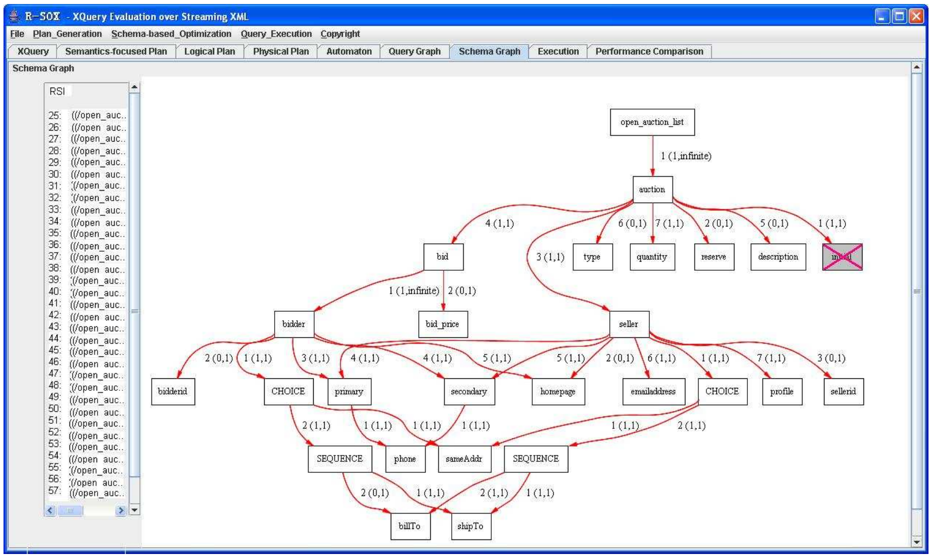

Figure 3. Runtime Schema Graph in R-SOX

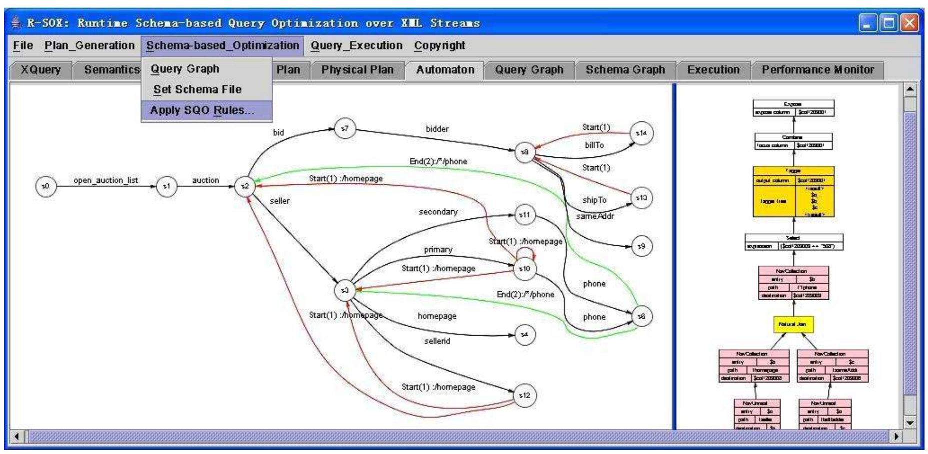

Figure 4. Runtime Query Plan in R-SOX

applying the early filtering technique. We will also show how and when to migrate the plan safely.

Performance Monitoring. We will demonstrate the performance benefits of different SQO strategies using metrics, such as execution time and buffer requirements.

References

[1] A. Gupta and S. Chawathe. Skipping Streams with XHints. Technical report, Univ. of Maryland, College Park, 2004.

[2] C. Koch, S. Scherzinger, N. Scheweikardt et al. FluxQuery: An Optimizing XQuery Processor for Streaming XML Data. In *VLDB*, pages 228–239, 2004.

[3] D. Florescu, C.Hillery et al. The BEA/XQRL Streaming XQuery Processor. In *VLDB*, pages 997–1008, 2003.

[4] H. Su, E. A. Rundensteiner and M. Mani. Semantic Query Optimization in an Automata-Algebra Combined XQuery Engine over XML Streams. In *VLDB Demo*, 2004.

[5] H. Su, E. A. Rundensteiner, M. Mani. Automaton Meets Algebra: A Hybrid Paradigm for XML Stream Processings. *DKE Journal*, 2006.

[6] H. Su, J. Jian, and E. A. Rundensteiner. Raindrop: A Uniform and Layered Algebraic Framework for XQueries on XML Streams. In *CIKM*, pages 279–286, 2003.

[7] H. Su, E. A. Rundensteiner, and M. Mani. Semantic Query Optimization for XQuery over XML Streams. In *VLDB*, pages 1293–1296, 2005.

[8] M. Wei, M. Li, E. A. Rundensteiner, and M. Mani. Processing recursive xquery over xml streams: The raindrop approach. In *ICDE Workshops*, page 85, 2006.

[9] Y. Diao, M. Altinel and M. J. Franklin. Path sharing and predicate evaluation for high-performance xml fi ltering. In *TODS*, pages 467–516, 2003.