Compressing and Filtering XML Streams

Giovanni Guardalben
HiT Software, Inc.
gianni@hitsw.com

Abstract

Information technology is widely adopting the use of XML for information exchange. As messaging standards migrate to XML, there is growing concern for the magnitude of messages compared to binary formatted messages. XML compression can help mitigate the risk of exceeding the capacity of current communication resources. However, it is critical that compression technologies do not hinder XML query processing efficiency. Schema-aware compression and stream-based query processing seem to provide the best combination because decoded values can be accessed by the filtering processor without producing the original XML document. When query specifications are predefined, further optimizations can be made by embedding path-related information for each tag and attribute in the compressed message. A similar approach could be adopted by other XML processes (such as XSL, XForms and others) by augmenting compressed information with specific 'hints' to improve performance .

1 Compressing and filtering XML

XML stream query processing systems provide fast and efficient matching of XML data to multiple XML query specifications simultaneously (typically, XPath [XPATH] and XQuery [XQUERY] expressions) [Diao et al. 2003][Green et al.2003]. It is now evident that event-based parsing can provide the basis for very scalable XML filtering systems. It is also well accepted that event-based parsing is the foundation of many efficient Infoset-level (schema-aware) methods for XML compression [XMill: Liefke & Suciu 2000][XCQ: Lam et al. 2003][Cheney 2001]. This convergence of XML query processing and XML compression induces optimism towards achieving efficient algorithms for XML filtering of XML compressed data without reconstructing the original XML document. Querying streamed XML data efficiently is critical because of potentially very high rate of stream data generation by remote data sources.

Typically, compressing and query filtering only interact when decoding compressed XML data. In fact, when encoding XML data, a SAX parser is invoked on the input XML document(s) and for each event generated by the parser only the encoder acts on the tokens. Although there are many event-based compression algorithms, in general XML tokens are assigned to specific containers. A different container is usually selected for XML structures compared to the ones used for XML data. When parsing is complete, each container is further compressed using redundancy-based methods such as winzip, gzip, or others [GZIP][WINZIP]. When decoding, the container for the XML structure container is unzipped first. Then, the decoder acts as a SAX parser, parsing encoded sequences rather than XML text.

Figure 1 shows the architecture of a Compression and Filtering engine. The XML Decoding Engine acts as a SAX parser for the XML Query engine. The XML filtering process is completely unaware that compression took place prior to sending the compressed XML stream to the data transport layer. In general, before handling SAX events, it parses the XML query specifications (XPath or XQuery expressions) and it generates intermediate data structures, such as XML query algebra trees (for query optimization purposes) and Nondeterministic Finite Automaton (NFA), or Deterministic Finite Automata (DFA). When handling SAX events, the automata create accepting reverse paths for query structure matching and predicate evaluations.

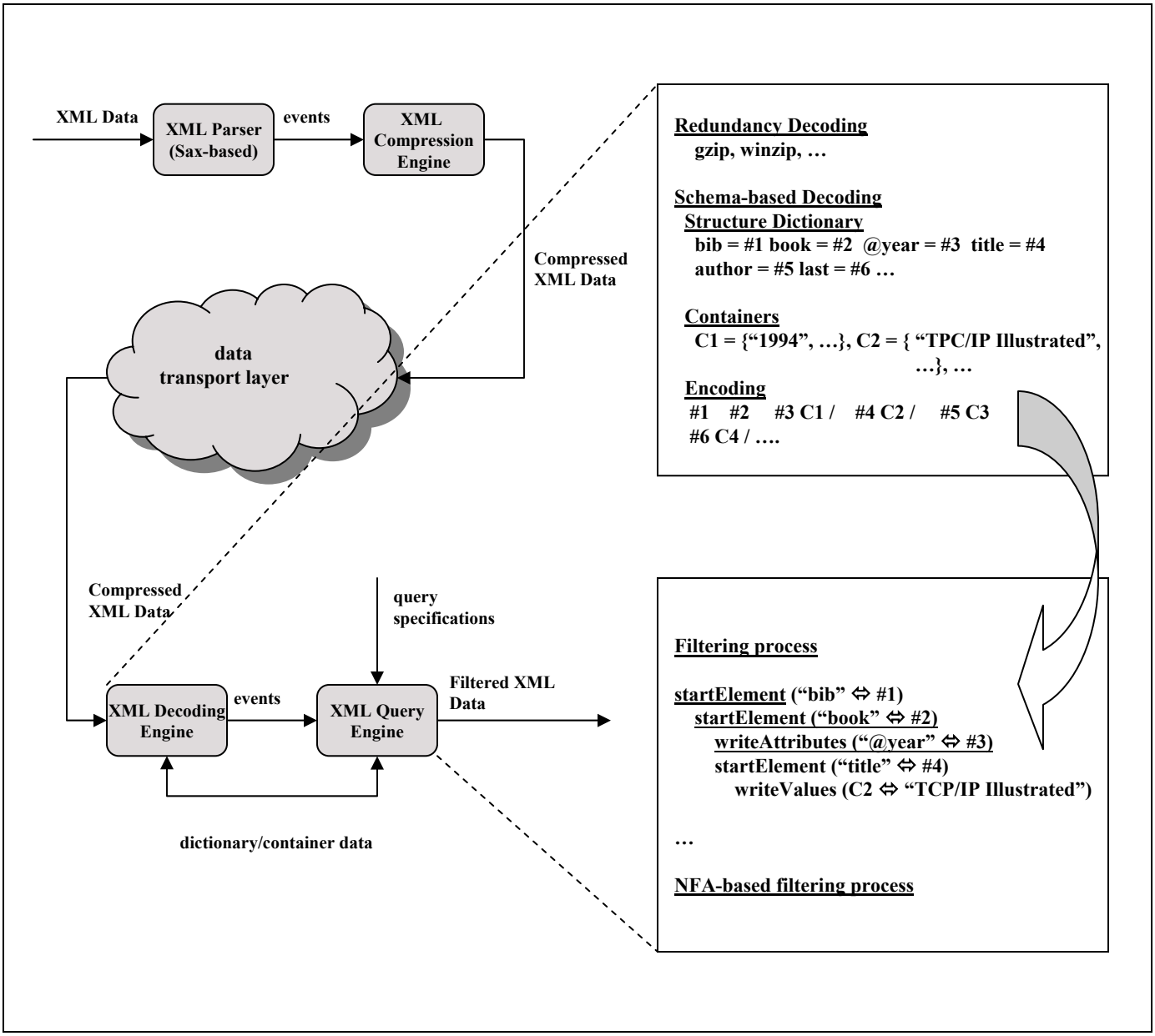

Figure 1: Architecture of a typical compression and filtering system

By far, the biggest advantages to using the automata-based filtering methodology are the scalability of the approach (because this technology does not rely on in-memory data structures – such as the DOM) and query path sharing (because a single automata can process simultaneously multiple query specifications).

The architecture we just described for a 'typical' compression and filtering system can be further improved if filtering specifications are known in advance of streaming (and compressing) XML data. In fact, there are conditions (such as when deriving statistics information from network monitoring systems that log network events using XML, or when obtaining billing information from telephone exchanges or B2B transactions) when users know well in advanced their query requirements and wish to execute queries as efficiently as possible. To do so, an option is to "augment" compressed data with a single NFA (or DFA) for all XML queries and to add NFA (or DFA) state transitions for each tag and attribute. When decompressing and filtering, we easily generate reverse paths from state transitions. These paths are subsequently used for structure matching and predicate evaluations.

Figure 2 shows the architecture of a compression and filtering system where some queries are predefined and (possibly) other queries (ad-hoc query specifications) are known only prior to decoding. Compared to Figure 1, the XML Compression Engine also acts as Pre-Filtering Engine by processing 'predefined' query specifications. The encoded information is extended with a serialized NFA (or DFA) obtained by pre-compiling the predefined queries. The structure container is also extended to include state transitions for each tag and attribute. States identifiers correspond to the ones that occur in the NFA (or DFA).

When decoding the compressed XML stream, if ad-hoc queries are present, they are compiled and the NFA (or DFA) from the compressed stream is extended with new states corresponding to the ad-hoc filtering steps. Processing is then similar to the 'typical' scenario, except for the fact that state transitions for predefined queries are known in advance. Reverse paths can be built on-the-fly without scanning the NFA (or the DFA) for token matches. The automata are scanned only for matches for the states introduced by the ad-hoc queries. We found that for large XML streams, identifying (and creating) state transitions may take a significant percentage of the total processing time. So, there is real advantage to pre-computing them when compressing XML data. We will provide detailed results in future technical reports.

2 Conclusions

When defining the requirements to develop standardized compression methods, provisions should be made for adding extended information to schema-aware compressed XML data. While bloating the size of compressed XML, the extra data may help improving significantly the overall execution performance. In the case of compressing and filtering XML according to predefined query specifications, we found that embedding NFA (or DFA) state transitions to compressed tags and attributes helped reduce the execution time when decoding XML.

3 References

Cheney, J., 2001. Compressing XML with Multiplexed Hierarchical PPM Models. In *Proceedings of the 2001 IEEE Data Compression Conference, pp. 163-17, 2001.*

Diao, Y., Altinel, M., Franklin, M.J., Zhang, H., Fischer, P. 2003. Path Sharing and Predicate Evaluation for High Performance XML Filtering. *Submitted for publication.*

GZIP. Gzip, version 1.2.4. htpp://www.gzip.org/.

Green, T.J., Miklau, G., Onizuka, M., Suciu, D. 2003. Processing XML Streams with Deterministic Automata. In *Proceedings of ICDT, 2003.*

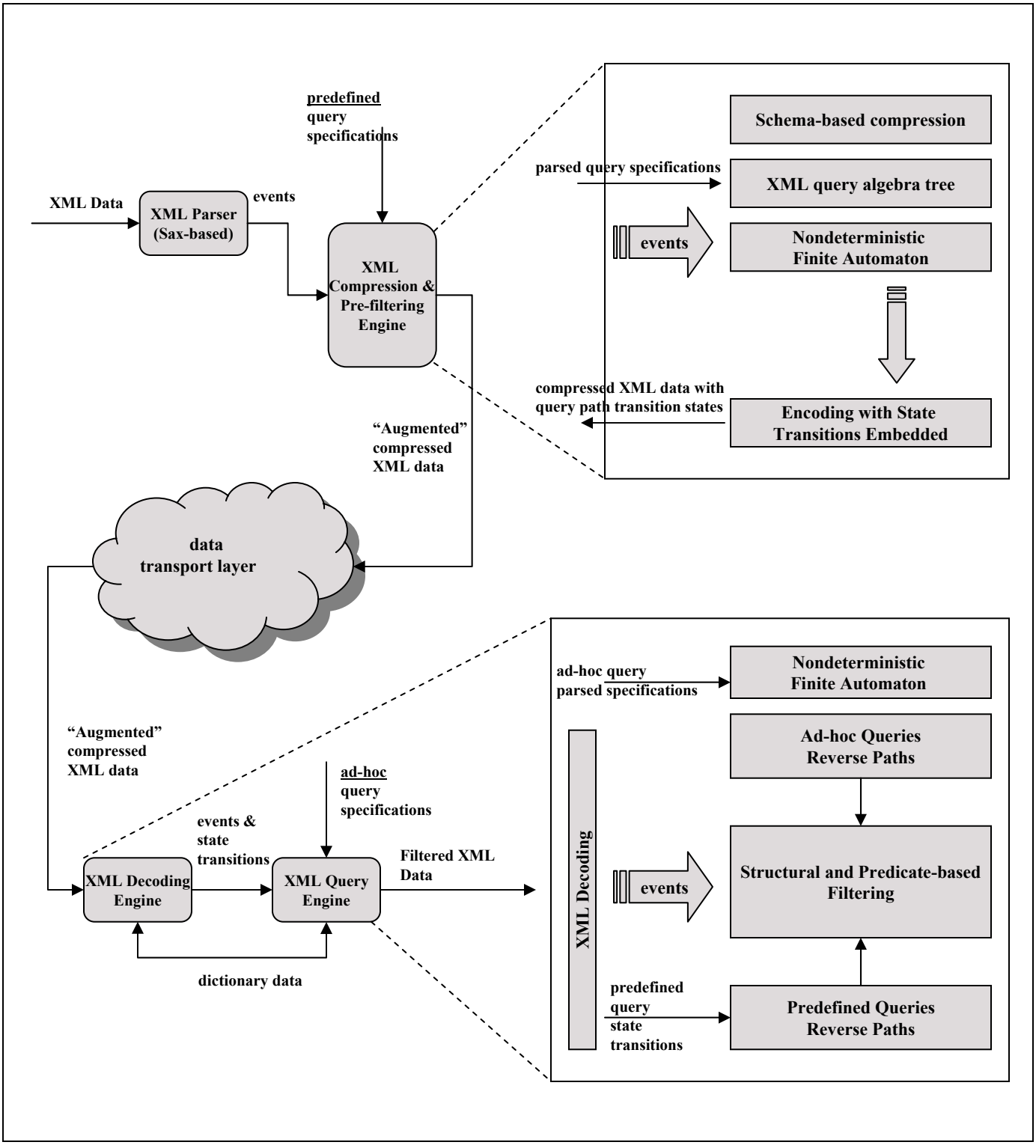

Figure 2: Architecture of a compression and filtering system with predefined queries

Lam, W.Y., Ng W., Wood, P.T., Levene, M. 2003. XCQ: XML Compression and Querying System. In *WWW2003 Poster 59, 2003.*

Liefke, H., Suciu, D. 2000, XMill: an Efficient Compressor for XML Data. In *Proceedings of SIGMOD, pp. 153-164, 2000.*

XPATH. 1999. XML Path Language (XPath) – Version 1.0. http://www.w3.org/TTR/xpath.

XQUERY: 2003. XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/.

WINZIP. http://www.winzip.com/