Chapter 9
XML Stream Processing:
Stack-Based Algorithms

Junichi Tatemura
NEC Laboratories America, USA

ABSTRACT

This chapter reviews recent advances on stream XML query evaluation algorithms with stack-based encoding of intermediary data. Originally proposed for disk-resident XML, the stack-based architecture has been extended for streaming algorithms for both single and multiple query processing, ranging from XPath filtering to more complex XQuery. The key benefit of the stack-based architecture is its succinct encoding of partial query results, which can cause exponential enumeration if encoded naively. In addition, the chapter discusses opportunities to integrate benefits demonstrated in the reviewed work. For single-query processing, a sketch is given for an integrated algorithm, StreamTwig^{2}Stack, that achieves all the benefits of existing algorithms in terms of functionality, time complexity, and buffer memory optimality.

INTRODUCTION

XML has become an essential format in data exchange application domains, where an application processes data in XML documents sent from other systems. In order to ensure interoperability, XML data is usually sent as text data, which needs parsing before being consumed by the application. Various XML stream processing technologies have been developed to enable query processing over streaming XML documents for such scenarios.

XML stream processing is to evaluate queries only by a single scan over an XML document. It requires different technologies from querying over disk-resident XML data where an XML can be converted into proprietary data structure with indexing. Moreover, in this streaming environment, we should avoid materializing the entire document as a tree in the main memory. A system often needs to evaluate a query in a low memory-profile environment. In

DOI: 10.4018/978-1-61520-727-5.ch009

some cases, XML processing is embedded in an application as a utility component. In other cases, a system must achieve high throughput to process a large amount of documents and/or queries.

XML stream processing has been extensively studied in recent years. Motivated by a variety of applications, these studies address various issues under different assumptions and problem statements.

Earlier work typically targets information filtering applications, where a query is a pattern matching expression that returns a Boolean value. Received an XML document, the system should identify a small fraction of matching queries from a large set of registered queries. Many filtering algorithms naturally employ an automaton-based approach, where the query results are modeled as acceptance states of automaton.

As XML became pervasive in data exchange, various application needs emerged beyond filtering. Many applications use queries to express more complex patterns that extract structured data from an XML document. In such a scenario, structural pattern matching in XML queries can generate a large amount of intermediary data, which has been recognized as a challenging research issue.

In this chapter, we give a review mainly on recent advance on stream XML query evaluation algorithms that efficiently manage intermediary data in a special data structure based on multiple stacks. Originally, this succinct encoding of intermediary data was proposed for processing disk-resident XML data (Bruno, Koudas, & Srivastava, 2002). However, the proposed algorithms, PathStack and TwigStack, have inspired various streaming algorithms, which have been demonstrated to be efficient for various applications.

We categorize algorithms into two types and review them separately: single-query processing and multi-query processing. At the end of each review, we discuss approaches to integrate benefits of the reviewed algorithms. For single-query processing, we also give a sketch of an integrated algorithm, StreamTwig^{2}Stack, that achieves all the benefits of existing algorithms (functionality, time complexity, and buffer memory optimality).

BACKGROUND

Queries

XPath and XQuery

In the context of stream processing, most research focuses on *Univariate XPath*, a subset of standard XPath 1.0, with the following major limitations: (1) it only supports forward axes (child, descendant, and attribute axes); (2) it does not allow a predicate that compares values of multiple nodes (e.g., A[B > C] is not allowed). Some work also supports backward axes (parent and ancestor axes) in addition. Also, it is known that an XPath query with a backward axis can be rewritten into one only with forward axes (Olteanu, Meuss, Furche, & Bry, 2002).

In order to specify various fragments of XPath within Univariate XPath, the literature often uses a notation P^S, where S is a subset of features {/,//,*,[]} that are supported by a query. For instance, $P^{\{/,//\}}$ is XPath that supports child and descendant axes, and $P^{\{/,//,*,[]\}}$ supports predicates and a wildcard in addition.

XQuery is a further complex query language. Most research work on streaming XQuery processing focuses on how to support FOR, LET, WHERE, and RETURN clauses.

Twig Query

With a research focus on structural pattern matching, many studies do not directly discuss XPath/XQuery but model a query as a tree pattern, called a *twig query*. A node (query node) of the twig corresponds to a node test, and an edge between two nodes represents their structural relationship (i.e., axes). The query result is an ordered or unordered list of tuples, each of which