ABSTRACT

Efficient XML filtering has been the fundamental technique in recent Web service and XML publish/subscribe applications. In this article, we consider the problem of filtering a streaming XML data efficiently against a large number of branch XPath queries. To improve the performance of XML filtering, branch queries are grouped into similar queries, and the common paths between queries in the same group are identified. After performing structural matching of queries, queries are organized in a way that multiple queries can be evaluated simultaneously in the post-processing phase. In the post-processing phase, join operations are executed in a pipeline fashion, and intermediate join results are shared amongst the queries in the same group. As a result, the total number of join operations performed in the post-processing phase is significantly reduced. In addition, we also present how to efficiently return all matching elements for each matching branch query. Experiments show that our proposal is efficient and scalable compared to previous work. [Article copies are available for purchase from InfoSci-on-Demand.com]

Keywords: Algorithms; Publish/Subscribe; Query Processing; XML; XML Filtering; XML Stream Processing; XPath

INTRODUCTION

With the development of the World Wide Web (WWW), there has been an increasing interest in Web-based applications. Recent works include online finance (Nah, Siau, & Tian, 2005), education (Siau, Sheng, & Nah, 2006), government (Siau & Long, 2006), healthcare (Siau & Shen, 2006) and firewall (Benedikt, Jeffrey, & Ley-Wild, 2008) applications. One important problem in building Web service applications is how to have interoperability of heterogeneous information. To address this problem, XML was developed, and has become the standard data exchange format in the WWW. There have been many research works on developing XML-based Web service applications, and XML stream processing has lately received much attention from the community.
We consider a Web service application that receives XML messages from various data sources from the WWW, and forwards these messages to users or other applications that have subscribed to such messages. An application that implements such functionality is called an XML publish/subscribe (pub/sub) system. In an XML pub/sub system, messages and user subscriptions are expressed in XML documents and XPath (Clark & DeRose, 1999) queries respectively. One key challenge of building an XML pub/sub system is how to efficiently evaluate a large number of complex user subscriptions against a continuous stream of XML data in real time. In this article, we consider an XML pub/sub system with the following two properties: (1) the system evaluates many but similar XML documents that share a common document structure against a set of branch XPath queries, and (2) the system has information about the XML document structure (i.e., XML Schema or DTD). In this system, subscribers formulate branch XPath queries specific to the XML documents that the system processes by using the document structural information obtained from the system. In practice, this is most likely the case for Web service applications. As a result, the probability of this set of queries being matched is normally higher than general and random queries that are formulated without such information.

There are many previous works that present techniques for filtering streaming XML documents. Some works are not suitable for our needs, as: (1) XFilter (Altinel & Franklin, 2000), LazyDFA (Green, Gupta, Miklau, Onizuka, & Suciu, 2004), Bloom Filter (Gong, Yan, Qian, & Zhou, 2005) and AFilter (Candan, Hsiung, Chen, Tatemura, & Agrawal, 2006) cannot evaluate branch XPath queries, and (2) TwigStack (Bruno, Koudas, & Srivastava, 2002), FluX (Koch, Scherzinger, Schweikardt, & Stegmaier, 2004), XFPro (Huo et al., 2006), XSQ (Peng & Chawathe, 2003, 2005), SPEX (Bry et al., 2005, Olteanu, 2007) and Gou and Chirkova (2007) cannot process multiple XPath queries. Supporting branch XPath queries is important in Web service applications, as subscribers usually subscribe to the documents by setting a few constraints that documents should satisfy. To process a branch query, we split the query into a set of multiple linear queries, evaluate the linear queries against a streaming document, and report the branch query as matched if all of its linear queries are matched. This is similar to the approach used by YFilter (Diao, Fischer, Franklin, & To, 2002; Diao, Altinel, Franklin, Zhang, & Fischer, 2003) and Onizuka (2003). The experiments in YFilter demonstrate that the approach is efficient for processing a large number of queries. One characteristic of this approach is that most of the processing time is spent in the post-processing phase where decomposed linear queries are combined and checked for validity. To reduce the time required for the post-processing phase, it is necessary to: (1) identify and avoid any unsuccessful and duplicate processing of queries; and (2) branch queries must be evaluated as a group rather than individually by inspecting any commonalities between branch queries. While many previous works (Chan, Felber, Garofalakis, & Rastogi, 2002b, 2002a; Diao et al., 2002, 2003; Gupta & Suciu, 2003; Onizuka, 2003; Rao & Moon, 2004; Kwon, Rao, Moon, & Lee, 2005) present some ways of combining states/stacks/nodes while parsing XML documents, they do not provide any methods that evaluate groups of queries without checking the status of each individual branch query. In addition, they only report matching query IDs per streaming document, and do not provide methods for returning all matching elements for each matching query. In this article, we present a technique that addresses the above problems.

First, to avoid unnecessary join operations, we only consider the queries whose query structures match the document structure against which the queries are evaluated. To identify such queries, we build a query index for a set of documents based on the information about document structures, and pre-process the queries against the query index prior to evaluating them against streaming XML documents. By doing so, we identify the queries whose structures
Quality Analysis Specifications: A Comparison of FOOM and OPM Methodologies
www.igi-global.com/chapter/quality-analysis-specifications/4379?camid=4v1a

On Estimators for Aggregate Relational Algebra Queries
www.igi-global.com/article/estimators-aggregate-relational-algebra-queries/51174?camid=4v1a