14

Web-Based Distributed XML Query Processing

Marko Smiljanić, Ling Feng, and Willem Jonker

14.1 Introduction

Web-based distributed XML query processing has gained in importance in recent years due to the widespread popularity of XML on the Web. Unlike centralized and tightly coupled distributed systems, Web-based distributed database systems are highly unpredictable and uncontrollable, with a rather unstable behavior in data availability, processing capability and data transfer speed. As a consequence, there exists a number of conspicuous problems that need to be addressed for Web-based distributed query processing in the novel context of XML. Some major ones are listed below.

- *High autonomy of participanting sites.* Data sources scattered on the Web may be equipped with either powerful query engines, say those which can support XPath, XQuery or some other XML query languages, or simple ones which offer limited processing capabilities, just like a plain Web server returning the whole XML files. In Web-based distributed database systems, both the data sources and their associated processing capabilities need to be modeled and used in query execution planning.
- *XML streaming data* is proliferating and flowing on the Web. As the size of the streaming data is usually enormous, it is not efficient to first wait for all data to arrive, store it locally and then query it. Instead, new techniques must be deployed for querying streaming XML data.
- *Unreliable response time* on the Web exists due to dozens of Internet routers that separate nodes participating in distributed processing. High delay and a total data jam must be taken into account. When congestions are detected by certain mechanisms like timers, querying should activate alternative execution plans to keep the system busy with performing some other relevant tasks.
- *Different expectations of query results.* The classical way of querying suggests the delivery of a complete and exact query result. In such systems, users prefer to have the "time to last" result as short as possible. However,

H. Blanken et al. (Eds.): Intelligent Search on XML Data, LNCS 2818, pp. 207–216, 2003.

in a dynamic environment like the Internet, more querying facilities must be introduced. For example, users may opt for getting the first element of the result quickly with others coming afterwords. The complete result is of no interest at the first instance. Such systems can thus be optimized to have "time to first" result shorter. Another approach is to have results available "on demand" at any stage of query processing. To this end, we can let the systems present the current state of the answer and then resume the processing.

The aim of this chapter is to survey recent research on Web-based distributed XML query processing, with the emphasis on technical innovations that have been devised in query planning, query execution and query optimization.

The reminder of the chapter is organized as follows. Section 2 describes several query processing schemas, including centralized vs. distributed schema, static vs. dynamic schema, and streaming data query processing schema. Section 3 presents several XML query optimization techniques. We review a few Web-based systems in Section 4. Section 5 gives the conclusion.

14.2 Web-Based Distributed XML Query Processing Schemas

Figure 14.1 shows a generic Web-based distributed XML query processing architecture, containing three nodes A, B and C. Each participant node accommodates some XML data and is equipped with a query processor. Users at the client site are unaware of such a distributed architecture. They pose queries via a single entry point, assigned to node A. Nodes in this system can be heterogeneous in terms of both data sources and query processors. Some systems also provide users with an integrated view, so that users can ask questions over the integrated view.

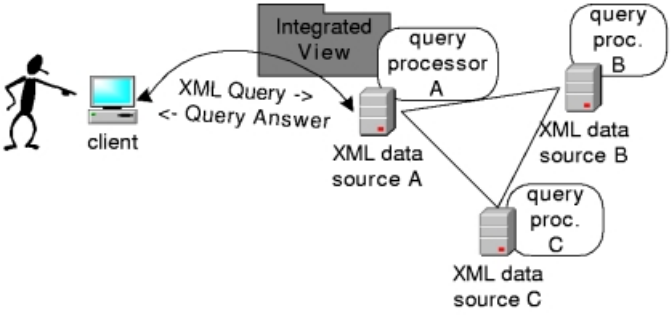

Fig. 14.1. A generic Web-based distributed query processing architecture

We now describe in detail various distributed query processing schemas. The taxonomies for classifying these schemas are based mainly on different approaches distributed query processors employ in query planning and execution in response to a user's query. Note that the following taxonomies are not orthogonal.

14.2.1 Centralized vs. Distributed Query Processing Schema

According to the location where queries are planned and executed, we categorize query processing into the following three groups.

Centralized Planning and Centralized Execution.

This is the simplest, and currently the most frequently used architecture in distributed Web systems. In such a system, one node carries all the responsibilities: it collects and warehouses XML data, plans and executes queries. Other nodes are accessed in the off-line mode and are asked only to provide the raw data. *Google*, a keyword-based search engine, falls in this group (though Google itself does not solely use XML data). Niagara system [227], a combination of a search engine and a query processing engine, allows users to raise arbitrary structured queries over the Internet. Niagara can perform "on-demand" retrievals on the distributed documents if they are not available in the local document repository, but are referenced by the full text search engine. Xyleme system [75] provides users with integrated views of XML data stored in its local warehouse.

Centralized Planning and Distributed Execution.

When a node receives a user's query over a virtual view of data sources, it produces a complete set of instructions that will evaluate the query in the distributed environment. These instructions will be sent to corresponding nodes which will optimize the execution of their local subqueries. In this case, we have a centralized distributed query processor, responsible for generating a complete query execution plan. However, the query execution that follows is delegated to respective local query processors.

Distributed Planning and Distributed Execution.

In contrast to centralized query planning carried out by one node, distributed query processors at multiple nodes can coordinate with each other to derive a global query plan. For example, node A, upon the receipt of a query, can plan initial stages of the query execution using its local data, and then send the "rest" of the query to some other nodes, say B and C, where involved data sources are located. Node B and C then plan further execution of the rest of the query. The query processors of participanting nodes can be either identical or different.

14.2.2 Static vs. Dynamic Query Processing Schema

Once a query plan is generated, it can either be altered or remain unchanged during the query execution phase.

The *static* query processing schema will execute the query plan until completion. The problem with the static query processing schema is that it is vulnerable to unexpected events, like long initial delay in data arrival or general traffic congestions. The need for dynamic query processing capability that can act upon the emergence of such events especially in the Web context is thus highly desirable.

In *dynamic* query processing schema, the query execution planner gets a feedback from the query execution engine on the status of the execution. If some delays are detected, the query planner produces alternative plans for finishing the query processing or just reorders the actions that were planned for later execution. [11] describes a dynamic query plan modification strategy for wide-area remote access, where a scrambling technique is used to generate the alternative execution plan when delays in arrival of data are experienced. As soon as the original data becomes available, the execution is continued at the point before the scrambled plan was used. Scrambling is based on the rescheduling of the operators in the execution plan. However, it must be taken into account that the scrambled processing increases the cost (CPU, I/O and memory usage) of the query execution.

14.2.3 Streaming Data Query Processing Schema

Significant amount of streaming data may be generated and transported from one site to another during the execution of a distributed query on the Web. As such streaming data can be of enormous size, it is not efficient to first wait for all data to arrive, store it locally and then to do the processing. Instead, techniques for processing of the streaming XML data are exploited. Such processing techniques are applicable to all the schemas discussed above.

In [172], it is shown that a group of regular XPath expressions can be simultaneously executed over streaming XML data. The proposed X-scan processor uses a state machine to detect the satisfaction of XPath expressions over incoming XML data. States of the state machine correspond to steps of the XPath queries and the input to the state machine is the data stream arriving from the input. The key challenges in this approach include dealing with cyclic data, preserving the order of elements and removing duplicate bindings that are generated when multiple paths lead to the same data elements. To cope with those problems, the X-scan engine parses the data, creates the structural index to enable fast IDREF to ID traversal and also maintains a "not-yet-seen" ID list.

Joining can be also done between two or more simultaneously incoming streams of XML data. Techniques for joining the streaming data are based on non-blocking pipelined hash join method. In this approach, data is arriving

from two data sources from the network. Each data entity is first placed in the hash structure for that data source, and then the other hash is probed. As soon as a match is found, the result tuple is produced. In order to cope with the memory overflow due to the hash size, other techniques involving data storage on secondary storage medium are proposed. Some specifics of XML element joining are discussed in [299], demonstrating that Lattice-Join of XML documents can be implemented as merge operation.

If data streams are continuous like stock indexes, or simply too long, it is desirable for a system to be able to show the current status of results being generated. A query processor has been designed to support partial result generation for non-monotonic functions such as sort, average, and sum, etc. [278].

14.3 Web-Based Distributed XML Query Optimization

When comparing XML query processing with relational database query processing, one impression we have is that those two fields overlap on all the major problems. Indeed, techniques like selection pushdown or querying of materialized views are applicable to query optimization regardless of the data models used. What makes an essential distinction is the environment in which the two models are used and thus different requirements they shall satisfy in both performance and functional domains. In the following, we describe several techniques known from the "classical" query optimization approaches, and discuss how they fit the XML world.

14.3.1 Selectivity Estimation

Selectivity estimation is used by query optimizers to plan for the execution order of subexpressions. Expressions yielding smaller result sets are said to have higher selectivity and are scheduled for early execution. The more complex the operation used in an expression, the more difficult it is to have good selectivity estimation for it.

For XPath expressions containing no predicates on XML documents, selectivity estimation equals the number of elements selected by the XPath. This information can be provided by counting the markup particles in the documents. In the presence of predicates or joins in query expressions, such selectivity information must be extended with statistical data on the XML document content.

The selectivity information has to be stored and accessed with respect to performance requirements of the query planner. This inevitably puts limits on the selectivity estimation in both data size and complexity of selectivity calculation algorithms.

In [5], two methods for storing "basic" XPath selectivity information are used. First, the Path Trees come in the form of a schema tree for an XML

document. Each node of such trees represents a path, leading from the root to that node. Each such node is assigned an accumulated number of element instances reached using the path described by the node. This number represents the selectivity for that path. It can be calculated in a single pass over the XML document. The second technique is named "Markov tables". A table is constructed with a distinct row for each path in the data up to the length of m. The calculation of the estimation for longer paths based on the data in the table is shown in [5]. Several summarizing techniques can be exploited to reduce the size and the complexity of the data structures used in both approaches.

14.3.2 Selection Pushdown

Selection pushdown is an optimization technique in which selection operators are scheduled early, e.g., before join operators. When executed, selection reduces the number of values that are to be fed to the expensive join operator, resulting in a low execution cost. This technique has a straightforward application in the XML field [172].

In [69], selection operator placement is analyzed in the context of query systems, which serve a large number of continuous queries (i.e. persistent queries, yielding results once available). Such simultaneous queries are analyzed for common subexpressions which are then grouped for joint execution. This provides resource and time savings. It is shown that in this setup, PushDown (select first) technique is outperformed by the PullUp (join first) technique. The rational behind those results is that major improvement comes from the operator grouping (i.e. reuse) and not from the operator ordering. Executing the join first enables the wide reuse of the results of this expensive operation.

14.3.3 Integrated Views

In the presence of integrated views, query processors can start with query rewriting in such a way that a query over a global view is translated into a set of queries over local data sources. Such rewriting and creation of a distributed execution plan involves the techniques known as data-shipping and query-shipping. [210] describes an architecture for integrating heterogeneous data sources under an XML global schema, following the local-as-view approach, where local sources' schemas such as relational and tree-structured schemas are described as views over the global XML schema. Users express their queries against the XML global schema in XQuery, which is then translated into into one or several SQL queries over the local data sources. The advantage of using a relational query model lies in the benefit from the relational query capabilities that the relational or XML sources may have. The tuples resulting from the SQL query execution are then structured into the desired XML result.

In addition to facilitating query rewriting with views, using materialized views to answer a query can also greatly speed up query performance, especially when one is confronted with a large volume of data sources on the Web [67, 2, 212].

14.4 Web-Based Query Processing Systems

In this section, we overview a few Web-based database systems. The systems will be analyzed through the spectrum of architectural features covered in the previous sections.

14.4.1 Distributed Query Evaluation on Semistructured Data – Suciu

XML bears a close similarity to semi-structured data models [44, 54, 24]. One pioneering work on distributed querying over semistructured data was done by Dan Suciu [288], who proposed the efficiency definition for a distributed query from the following two aspects.

1) The total number of communication steps between the data sources is constant, i.e. independent on the data or on the query. A communication step can be a broadcast, or a gather, and can involve arbitrary large messages.

2) The total amount of data transferred during query evaluation should depend only on (a) the total number of links between data sources, and (b) the size of the total result.

Suciu investigates distributed queries in a context where data sources are distributed over a fixed number of nodes, and the edges linking the nodes are classified into local (with both ends in the same node) and cross edges (with ends in two distinct nodes). Efficient evaluation of regular path expression queries is reduced to efficient computation of transitive closure of a distributed graph. For more complex queries, where regular path expressions are intermixed freely with selections, joins, grouping, and data restructuring, a collection of recursive functions can be defined accordingly. Those iterate on the graph's structure. The queries in this formalism form an algebra $\mathcal{C}$, which is a fragment of UnQL [53, 55]. By following an algebraic rather than an operational approach, a query Q can be rewritten into Q', called a decomposed query, such that on a distributed database, Q can be evaluated by evaluating Q' independently at each node, computing the accessible part of all results fragments, then shipping and assembling the separate result fragments at the user site.

The proposed query evaluation algorithms provide minimal communication between data sources. Even if several logical 'jumps' (joins in queries) between data sources exist, execution is planned in such a way that those data sources exchange data between each other just once. This does not come

without a price. The centralized query planner has to know all the metadata on the participating data sources to plan the query.

The algorithm and the systems described by Suciu fall in the category of centralized planning and distributed evaluation architectures.

Since the autonomy and the dynamics of the data sources are quite high on the Web, a high cost of maintaining a central metadata repository will be incurred. Some alternative approaches of query evaluation are thus raised in the sequel.

14.4.2 WEBDIS

WEBDIS system processes queries over Web documents by query shipping, where all the data sources are equipped with WEBDIS query servers running as daemon processes that can access the HTML, XML or other types of data [149]. An example query that WEBDIS can process is like "*Starting from COMPUTER SCIENCE HOMEPAGE find RESEARCH PROJECTS PAGES linked to it over maximum 5 links and return the available PhD POSITIONS.*"

WEBDIS starts query execution at the user's site based on the initial execution plan. Parts of the query are processed locally, while the rest of the query is dispatched to other nodes. Those nodes then proceed in a similar fashion using their own plan generators. Hence we classify WEBDIS system in the group of systems where both query planing and query evaluation are performed in a distributed fashion. Some particular issues to be addressed by this distributed query processing system include:

1) *Query completion.* The query should migrate from one node to another without a control of the query issuer. To enable query completion monitoring, additional protocols are used.

2) *Query termination.* The cancellation request of an ongoing query should be dispatched to all active data sources. One simple solution is to close the communication sockets at the user site, which will eventually bring all participating data sources to cease the query processing.

3) *Avoiding query recomputation.* Some data sources may be faced with the same subquery during one query execution. Mechanisms to detect duplicated requests are introduced to prevent repeated computation of such queries.

4) *Result returning.* Results of the query can be directly sent to the user site from every participant node or collected by backtracking the path through which query was distributed.

14.4.3 Xyleme

Xyleme is designed to enable querying of large amounts of XML documents stored in its warehouse [75]. Documents are collected and refreshed by being (re)loaded from the Web. This activity involves crawling or subscription arrangements.

User can query the documents in the repository through a predefined integrated view. The integrated views for specific thematic domains are defined by domain experts but the mapping between each integrated view and the documents in the warehouse is established using the support of sophisticated mapping algorithms. Apart from that, the main strengths of Xyleme lie in the layered and clustered internal architecture. The architecture provides good scalability in terms of both the number of users and the number of XML documents stored in the system.

As illustrated, Xyleme falls in the group of Web-based databases with centralized planning and centralized evaluation approaches.

14.4.4 Niagara and Tukwila

Niagara [227] and Tukwila [171] are both data integration systems implementing XML query processing techniques .

Niagara system is built as a two-component system with the first component being a search engine and the second one being an XML query processing engine. Niagara allows users to ask arbitrary structured queries over the Web. Its search engine uses the full text index to select a set of the XML documents that match the structured content specified in the query, while its XML query engine is used to perform more complex actions on the selected documents and to present the requested results. Niagara can perform on-demand retrievals of XML documents if they are not available in the local document repository. Still, all the XML query processing is performed centrally.

In comparison, Tukwila provides a mediated schema over a set of heterogeneous distributed databases. The system can intelligently process the query over such mediate schema, reading data across the network and responding to data source sizes, network conditions, and other factors.

Both Tukwila and Niagara possess a dynamic feature. Their query processing is adaptable to changes in the unstable Web environment. Adaptability is thus defined as a special ability of the query processor, using which the execution plan of the query is changed during the course of its execution in response to unexpected environmental events. Both systems achieve adaptable query processing by implementing flexible operators within their query engines. In Niagara, operators are built in such a way that they provide non-blocking functioning. This means that they can process any data available at their input at any time. Faced with data delays, the operators can switch to process other arriving data, and resume the original task when data becomes available.

In Tukwila, a re-optimization is done on the level of query execution fragments - which are units of query execution. After each fragment is materialized, Tukwila compares the estimated and the achieved execution performance. If sufficiently divergent, the rest of the execution plan is re-optimized using the previous performance sub-results [227]. In addition, a collector operator is proposed for managing data sources with identical schemas. The

collector operator can dynamically switch between alternative different data sources when getting the necessary data.

Table 14.1. A comparison of Web-based query processing systems

Feature	*System*				
	Suciu	WEBDIS	Xyleme	Niagara	Tukwila
Data Source	semistructured rooted labeled graphs	hyperlinked XML, HTML documents	XML data	XML data	XML data
Query Planning	static centralized	static distributed	static centralized	dynamic centralized	dynamic centralized
Query Execution	static distributed	static distributed	static centralized	dynamic centralized	static centralized
Querying of Streaming Data	-	-	-	yes	yes
Integrated View	-	-	yes	-	yes
Query Granularity	graph node	XML document	XML component	XML component	XML component
Query Language	UnQL	DISQL	XQL	XML-QL	XQuery

Table 14.1 summarizes different query processing schemas that the above systems utilize, together with their query facilities offered to users.

14.5 Conclusions

In this chapter, we address some major challenges facing Web-based distributed XML query processing. Recent work on distributed XML query processing and optimization were presented. We review and compare several existing systems according to their query planning and query execution strategies, as well as their query facilities offered to users.

With XML becoming the dominant standard for describing and interchanging data between various systems and databases on the Web, Web-based distributed XML query processing opens up a new research area, attracting lots of attention from both academic and practical activities nowadays. It is not possible and also not our intention to cover all the activities in this short chapter. Our purpose is to stimulate the interests among the data management community as we feel that there are still quite a number of issues to be addressed by both academic researchers and practitioners.