WebContent: Efficient P2P Warehousing of Web Data

S. Abiteboul[1] T. Allard[2] P. Chatalic[1] G. Gardarin[2] A. Ghitescu[1]
F. Goasdoué[1] I. Manolescu[1] B. Nguyen[2] M. Ouazara[1] A. Somani[1,4]
N. Travers[3] G. Vasile[1]
S. Zoupanos[1]
[1]INRIA Saclay–Île-de-France & LRI - Univ. Paris-Sud, France [2]Univ. Versailles-Saint-Quentin, France [3]CNAM, France [4]IIT Bombay, India

ABSTRACT

We present the WebContent platform for managing distributed repositories of XML and semantic Web data. The platform allows integrating various data processing building blocks (crawling, translation, semantic annotation, full-text search, structured XML querying, and semantic querying), presented as Web services, into a large-scale efficient platform. Calls to various services are combined inside ActiveXML [9] documents, which are XML documents including service calls. An ActiveXML optimizer is used to: (i) efficiently distribute computations among sites; (ii) perform XQuery-specific optimizations by leveraging an algebraic XQuery optimizer; and (iii) given an XML query, chose among several distributed indices the most appropriate in order to answer the query.

1. CONTEXT

The Web has become the platform of choice for the delivery of business applications. In particular, the popularity of Web service technologies (WSDL [18], BPEL4WS [10] etc.) and their closeness to HTML and XML, the predominant content delivery languages on the Web, has opened the way to the development of complex business applications by integrating Web services provided by different parties. This model has several advantages. From a development viewpoint, it relies on widely accepted standards, and benefits from the plethora of available application building blocks. From a business viewpoint, it allows organizing the activity in cleanly defined modules, each of which is implemented by some Web services. This enables several entities to provide implementations of a given module, and facilitates replacing one entity with another.

We are currently involved in a large R&D project called WebContent [23], whose purpose is to build and exploit large-scale repositories of rich, semantically annotated Web data. The overall setting of the project, outlined in Figure 1, exemplifies the kinds of applications discussed above. A *focused crawler (1)* service returns Web documents related to specific domains, in our case, aircraft sales by Airbus and Boeing (for a continuous, online market survey), respectively, food risk information, for a consortium of food companies seeking to organize and structure information related to different food problems (contaminations, allergens etc.).

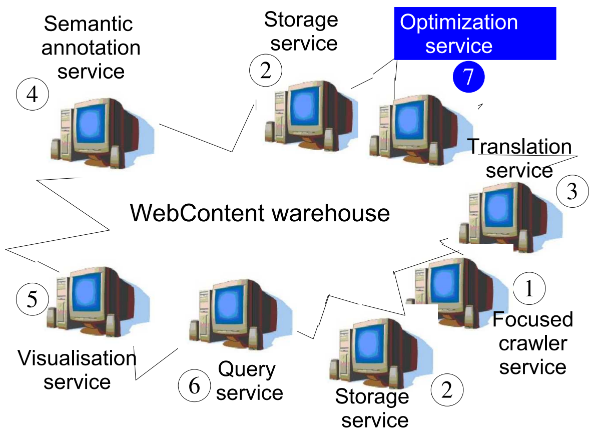


Figure 1: WebContent architecture outline.

The crawler service returns XML documents with information-rich headers (crawling date, origin site etc.). A *storage (2)* service can be invoked to make the crawled document persistent in the WebContent warehouse. Observe that multiple *Translation (3)* services are used to translate to and from English, French, Chinese etc. *Semantic annotation (4)* services are invoked to analyze the text of the crawled pages and extract, e.g., specific aircraft brands, names of edible plants or bacteria that taint food etc. The annotations are added as a semantic header to the XML documents, under the form of XML-ized RDF snippets, and the modified documents are put back in the store. *Visualization* (5) and *query* (6) services can be used at this point to exploit the corpus, either via advanced user interfaces (e.g. "fish-eye lens" view on documents) or by querying it, using a subset of XQuery (with full-text search) or SPARQL [15].

The WebContent warehouse is deployed in two settings. First, in a "closed" scenario, in a company Intranet, all services are provided by in-house components and communication takes place via an ESB [11]. Second, in a distributed, decentralized setting, computers are connected via the Internet and communication takes place via Web services exchanged over SOAP [19]. In both cases, there can be several instances of each service, in particular, storage services are provided by multiple machines, to cope with large data volumes; and, services can be called from inside or outside the federation of sites implementing the warehouse.

Two problems have to be solved in both settings: identifying services that implement a given interface, and efficiently executing the Web service calls. Efficiency is a particularly important concern in the distributed setting, since data transfers from one site to another may become the bottleneck. However, distribution is a great asset for large-scale warehouses such as the ones envisioned in our target applications, with large (and growing!) data volumes, therefore we focus on the distributed setting. Another source of inefficiency concerns repeated (redundant) execution of identical service calls, which may occur in large data processing tasks.

To combine Web services provided by different partners in the WebContent project, our solution is based on a composition language, namely ActiveXML (or AXML in short) [9], which in our setting can be seen as equivalent to a subset of BPEL. An ActiveXML document is an XML document specifying which services to call, how to build their input messages, and how the calls should be ordered. AXML raises several interesting technical problems addressed in previous works [1, 2, 3, 7]. More recently, a conceptual AXML optimization framework [5] and an ActiveXML optimizer, named OptimAX [6], have been developed. Given an AXML document, OptimAX applies *equivalence-preserving rewritings* that transform it into different documents, producing the same results, but possibly very different in shape and in the set of services it invokes. Thus, the execution of the rewritten document is likely to both shorten response time, and consume less CPU resources, than that of the original document.

Following the service-oriented architecture illustrated in Figure 1, we have implemented OptimAX as a Web service which, when invoked with an AXML document, returns the rewritten document. This step allows to benefit from the kind of performance-enhancing techniques typically applied in distributed databases [17], but in a new setting: losely coupled (vs. tightly controlled servers), generic (vs. tailored to specific indices and execution techniques), extensible to any service (vs. limited to the "inside" of the database server box). Another important difference is that AXML (and OptimAX) support continuous (streaming) services, such as the crawler service in Figure 1, or more generally any RSS feed. XML streams are at the core of many modern Web applications, e.g. for keeping a portal's content up-to-date, or for implementing continuous business interactions in a workflow-style setting.

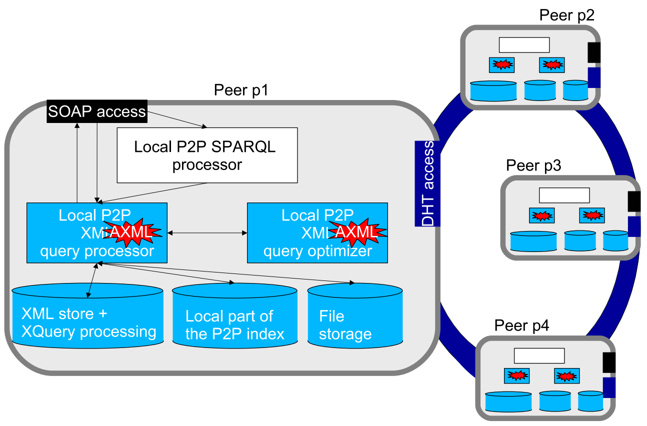
OptimAX functioning is extremely *generic*. It explores a search space based on a given set of rewriting rules. Some rules correspond to optimization techniques previously developed in isolation, such as query pushing and lazy query evaluation [2]. The same genericity allows us to apply several types of rewritings particularly useful in the WebContent setting:

algebraic XQuery manipulations : user queries expressed in XML over the distributed warehouse are analyzed with the help of an algebraic XQuery analyzer called TGV [16] which may e.g. decompose them into smaller queries that can be efficiently handled

query compiling : we have defined a few abstract services specific to the WebContent setting, for instance, a query answering service over the whole distributed warehouse. OptimAX will always replace a call to this service by a set of calls to concrete, implemented services, which compute the desired result.

DHT index selection : WebContent sites are connected in a DHT network, in which several types of XML content indices can be materialized. (Currently there are two such indices, provided by the KadoP [4] and PathFinder [13] systems, but this can clearly be extended.) OptimAX may pick one index or another, thus realizing distributed access path selection.

Demo highlights The demonstration will focus on two technical aspects. (i) The architecture of a WebContent peer; and (ii) the DHT XML index selection feature. While ingredients of the platform have been addressed in separate publications, the WebContent platform which integrates them, and the DHT index selection, are new and have not been presented yet.



Peer p1
SOAP access
Local P2P SPARQL processor
Local P2P XML AXML query processor
Local P2P XML AXML query optimizer
XML store + XQuery processing
Local part of the P2P index
File storage
DHT access
Peer p2
Peer p3
Peer p4

Figure 2: Internal architecture of a WebContent peer.

Demo scenarios We plan to use scenariso inspired from the WebContent project. A first scenario concerns market surveillance for the European Advanced Defence System (EADS) company, interested in economical news and market analysis concerning the sale of its Airbus aircraft.

This document is organized as follows. The next section details the WebContent peer architecture. Then, we focus on the precise issue of the DHT XML index selection in this context.

2. ARCHITECTURE OF A WEBCONTENT PEER

The overall architecture of the P2P platform is outlined in Figure 2. All peers are connected to a Distributed Hash Table (in short, a DHT) [12]. The DHT keeps the peers logically connected in distributed structure that assigns them *logical IDs*. Given a peer ID, a DHT structure is typically able to route a message to that peer, from any peer in the network, in at most $log_2(N)$ hops across the network, where N is the number of peers in the DHT.

The **storage** functionality is implemented jointly by peers in the network, each of which may become responsible of storing part of the resources, as follows:

- WebContent resources: XML documents are stored in an XML repository local to each peer, whereas other types of files (such as Word documents, PDF files etc.) are stored directly in the local file system. Modifications (such as enrichment etc.) to any resource which is an XML document are made via the update functionality of the local XML data management store on each peer.

- Parts of the index: an important aspect of the WebContent P2P platform is the capability to efficiently search the distributed warehouse. To that effect, data access structures are built and stored collaboratively by the peers.

- Other helper data structures, which (like the index) may increase performance. This category mainly concerns materialized views or caches (this feature is currently under development).

In principle, any XML database can be used to store XML resources on a given peer. We have integrated so far eXist [21] and MonetDB/XQuery [22], and we are currently working on interfacing with the Microsoft SQL Server.

Several modules in each peer cooperate to implement the platform's **search** and **update** functionality. We outline their roles next.

Local SPARQL semantic query processor Each peer is capable of processing semantic queries over RDF data, expressed in a conjunctive subset of the SPARQL [15] language. The semantic query processor exploits RDFS (RDF Schema) information in order to rewrite a query asked at a given peer, based on the *mappings* that explain how resources are potentially stored at several peers [8]. The output of the rewriting process is a union of XML queries which correspond precisely to the XML resources that are part of the answer to the query.

Local component of a distributed execution engine Peers collaborate in order to ship data from one to another and to compute query results in a collaborative fashion. To that effect, each peer is endowed with several functionalities which reside in its local execution module. Thus, a peer is capable of *sending and receiving* data (including XML subtrees, index entries etc.), as well as information on how a given query should be evaluated.

Local optimizer The volume and variety of resources considered in WebContent raise a significant challenge from the performance viewpoint. To address these challenges, the indexing framework and the execution engine provide several tools, such as data access support structures (or indices, described in the next section), efficient join algorithms, alternative evaluation strategies etc. Furthermore, peers in the WebContent network may have different capabilities (e.g. different computing power) which make them more or less interesting choices to process a given query. All these choices are available to the OptimAX optimizer on the query peer; OptimAX explores them using its local cost and catalog information.

Based on Figure 2, the *query processing chain* in WebContent can be described as follows.

Assume that peer p receives a query q_x expressed in a dialect of XQuery (for the time being, restricted to downward navigation and excluding type-dependent features such as *cast*, *typeswitch* etc.). The semantics of the query is: return results for query q_x from all the documents published in the WebContent warehouse. Clearly, a naïve semantics-driven implementation would exhibit very poor performance.

This query is modeled in our setting as an ActiveXML document including a call to the *WebContentQuery@p* service. Here, *WebContentQuery* is an abstract service, i.e. it is not provided by any concrete endpoint [18]. For a human reader-friendly version of AXML, we use the notation *f@p* to refer to a service f provided by peer p. OptimAX is aware that *WebContentQuery* is an abstract service, and includes a rewriting rule that replaces the call to *WebContentQuery(*q_x*)* with an expression containing nested service calls, of the form:

($\diamond$) *GenericQueryService@p(recomposeQ,*
tpq@any(t1), tpq@any(t2), . . ., tpq@any(tn))

where *t1, t2* etc. are conjunctive tree pattern queries, *tpq* is a tree pattern query processing service (to be detailed next), and *recomposeQ* is an XQuery query such that, for any set of XML documents indexed in the WebContent warehouse:

$$q \equiv recomposeQ(t1,t2,\ldots,tn)$$

The *GenericQueryService* is a query service available on all peers which, given an XQuery and some locally available XML inputs, evaluates the query and returns the XML answer, as in [2, 6] etc. In other words, the query q_x has been *decomposed* in a set of tree pattern queries, and a recomposition query which typically performs value joins, and element construction (in case q_x requires new elements in its result). The interest of the decomposition is that the tpq service is concretely available on all WebContent peers, and efficiently implemented based on distributed XML indices materialized on the DHT. The notation *tpq@any* specifies that any concrete endpoint for the *tpq* service can be used; they are all guaranteed to give the same answers, and using *any* informs the optimizer that the processing of these tree pattern queries can be pushed to any particular site, as is judged best for performance purposes.

As previously mentioned, two DHT-based XML indexing models are currently supported in WebContent. In our current platform, all peers provide thus two concrete implementations of the *tpq* service, each using a distinct index. The choice of which one to use is again made by OptimAX, based on the particular tree pattern query ti that is a parameter of each *tpq* call (this will be detailed in the next section).

Starting from the AXML expression ($\diamond$) previously mentioned, OptimAX enumerates some distributed execution strategies, seeking to replace the *any* locations of the *tqp* calls with specific peers, so as to reduce the amount of index data transfers necessary for evaluating the *tpq* calls. To do this, OptimAX uses cardinality statistics maintained in the network, locally cached copies of which reside on every peer. The search space is potentially large, therefore OptimAX can employ a set of heuristics such as greedily exploring the best cost-saving rewritings, exploring at most 30 rewritins etc., in the style of classical distributed query processing [17] applied to the specific OptimAX search space. The expression is then brought to a form:

($\square$) *GenericQueryService@p(recomposeQ,*
tpq@p1(t1), tpq@p2(t2), . . ., tpq@pm(tn))

and handled to the local AXML evaluation engine, which will trigger the calls to peers *p1, p2, . . ., pm*, receive the results, and integrate them in the document (as siblings of the service calls). The last call to be triggered is the one to *GenericQueryService*, which performs the last computations needed in order to produce the XML result of q_x at peer *p*.

An important extra layer to this query processing chain is needed when queries are not asked at the level of XML (syntax), but at the level of RDF classes (semantics). WebContent documents can be enriched with semantic annotations (recall Figure 1), and user may be interested in retrieving resources that are pertinent for a given semantic concepts. Consider such a semantic query q_s, expressed at peer *p* in a SPARQL-style language. Peer p may know of some *mappings* describing how a concept in s relates to other concepts present in the WebContent warehouse. Then, p initiates a semantic rewriting step, which traverses the available mappings and reformulates (enriches) s. For instance, if q_s asks for documents relevant to the concept "Airbus aircraft", if a mapping specifies that "A320 *isA* Airbus aircraft", then documents annotated with the "A320" concepts should also be part of the result.

RDF semantic annotations are serialized in XML in our platform, and indexed like any other XML resource. Therefore, once q_s has been fully rewritten based on mappings, it is automatically translated into an XML query q_x, and processed just as explained above.

3. DHT-BASED XML INDEXING

We now outline the two XML indexing models employed in WebContent and show how they can be used during query processing.

KadoP indexing The first WebContent XML indexing model is provided by the KadoP system [4]. In this model, *index keys* are extracted from: all XML element and attribute names; and, all words (after stemming and ignoring stop-words) that appear in text nodes and attribute values in XML documents. *Index values* are structural identifiers of the form *docID, start, end* characterizing *all the locations* where each individual key occurs in the system. For instance, for an element name *title*, the key is *n-title* (opposed to *w-title* which stands for a text word "title"), and the value is the list (sorted by *docID, start*) of the structural identifiers of "title" elements occuring in all WebContent documents. KadoP's tree pattern language consists of trees where each node is labeled with an XML node or a word, and edges stand for parent-child or ancestor-descendant relationships. KadoP processes such tree patterns by performing a Holistic Structural Twig Join on the lists of identifiers (also known as *posting lists*) associated to the query node labels. Observe that KadoP cannot process tree pattern queries with inequalities, e.g., *//article[year>2005]*. The KadoP index distribution on peers is left to the DHT's hash function.

PathFinder indexing The second indexing model comes from the PathFinder platform [13], which seeks to place index entries on the DHT peers in such a way as to reduce the number of hops (peers contacted) for answering a query. PathFinder keys are *linear parent-child rooted paths* encountered in XML documents of the networks, and the values are the sorted *docID* lists of identifiers of documents exhibiting a given path. The last step in PathFinder paths may be either an element or attribute name, or a text node (word occuring in an XML string) or attribute value. Sample index keys are: *article*, *title/title*, *article/title/WebContent* etc. By appropriately tuning the hash function, PathFinder is able to ensure that index entries corresponding to keys that are *close* (in lexicographic order) are placed on peers that are close (in the sens of proximity in the DHT structure). Therefore, it is capable of answering queries such as *article[title/'XML']/author* efficiently. Observe that the PathFinder index requires less data transfers than the KadoP one, because of its larger granularity (document, as opposed to element). It can also answer inequality queries such as */article[year>2005]*, because it can find all index paths starting whose prefix is */article/year* and whose last step is a value larger than 2005. In exchange, it does not support queries involving the // axis.

Putting it all together Given an XQuery query x_q, the TGV algebraic analyzer extracts the recomposition query and a set of tree patterns, which are then analyzed. Depending on their syntax, OptimAX will dispatch them to the KadoP, resp. to the PathFinder index. For instance, consider the query *for $x in //report[//'A320'], $y in /airbuslib/report[year>2005] where $x/author=$y/author return <res>{$x/title, $y/title}</res>*. The pattern *//report[//'A320']* will be handled by KadoP, while */airbuslib/report[year>2005]* will be handled by PathFinder. The recomposition query is: *for $i in $res1, $j in $res2 where $i/author=$j/author return <res>{$i/title, $j/title}</res>*. Patterns featuring both // and inequalities are currently handled to KadoP ignoring the value predicate, which is applied as a post-processing step.

4. RELATED WORKS AND CONCLUSION

We have mentioned in the introduction how the current work relates to previous AXML results. More generally, WebContent can be seen as an attempt to bridge the numerous existing technologies for handling XML data, distributed queries, Semantic Web technologies and Web services into building new-generation data management platforms. From this perspective, our approach is comparable with similar project such as [20]. WebContent differs from these in the focus on distributed data management, which we believe is a must for large-scale, growing data warehouses such as those we envision; in this context, WebContent leverages DHT-based XML indexing technologies [4, 16] that follow the first proposal in this area [14]. Finally, a specific innovative aspect in WebContent is the presence of OptimAX, an "external" optimizer that attempts to coordinate distributed data management stages without requiring a tight connection between the pluggable modules (implemented as Web services), facilitating the construction of efficient distributed Web applications.

5. REFERENCES

[1] S. Abiteboul, Z. Abrams, S. Haar, and T. Milo. Diagnosis of asynchronous discrete event systems: Datalog to the rescue! In *PODS*, 2005.

[2] S. Abiteboul, O. Benjelloun, B. Cautis, I. Manolescu, T. Milo, and N. Preda. Lazy query evaluation for Active XML. In *SIGMOD*, 2004.

[3] S. Abiteboul, A. Bonifati, G. Cobéna, I. Manolescu, and T. Milo. Dynamic XML documents with distribution and replication. In *SIGMOD*, 2003.

[4] S. Abiteboul, I. Manolescu, N. Polyzotis, N. Preda, and C. Sun. XML processing in DHT networks. In *ICDE*, 2008.

[5] S. Abiteboul, I. Manolescu, and E. Taropa. A framework for distributed XML data management. In *EDBT*, 2006.

[6] S. Abiteboul, I. Manolescu, and S. Zoupanos. OptimAX: Efficient Support for Data-Intensive Mash-Ups (demo). In *ICDE*, 2008.

[7] S. Abiteboul, T. Milo, and O. Benjelloun. Regular rewriting of active XML and unambiguity. In *PODS*, 2005.

[8] P. Adjiman, F. Goasdoué, and M.-C. Rousset. SomeRDFS in the semantic web. *Journal on Data Semantics*, 8, 2007.

[9] ActiveXML home page. Available at http://www.activexml.net.

[10] Business Process Execution Language for Web Services. www.ibm.com/developerworks/library/ws-bpel.

[11] D. Chappell. *Enterprise Service Bus*. O'Reilly, 2004.

[12] F. Dabek, B. Zhao, P. Druschel, J. Kubiatowicz, and I. Stoica. Towards a common API for structured P2P overlays. In *Proc. of IPTPS*, 2003.

[13] F. Dragan, G. Gardarin, and L. Yeh. Pathfinder: Indexing and querying XML data in a P2P system. In *WTAS*, 2006.

[14] L. Galanis, Y. Wang, S. Jeffery, and D. DeWitt. Locating data sources in large distributed systems. In *VLDB*, 2003.

[15] SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/.

[16] N. Travers, T. Dang-Ngoc, and T. Liu. TGV: A tree graph view for modeling untyped XQuery. In *DASFAA*, pages 1001–1006, 2007.

[17] P. Valduriez and T. Ozsu. *Principles of Distributed Database Systems*. Prentice Hall, 1999.

[18] W3C. WSDL: Web Services Definition Language 1.1.

[19] W3C. SOAP version 1.2 part 1: Messaging framework (second edition), 2007.

[20] P. Wu, Y. Sismanis, and B. Reinwald. Towards keyword-driven analytical processing. In *SIGMOD*, 2007.

[21] Exist: Open source native XML database. Available at http://exist.sourceforge.net, 2004.

[22] MonetDB database system with XQuery front-end. Available at http://monetdb.cwi.nl/XQuery, 2007.

[23] WebContent, the Semantic Web platform (rntl project). www.webcontent.fr.