Metacat: a Schema-Independent XML Database System

Chad Berkley
Matthew Jones
Jivka Bojilova
Daniel Higgins
{berkley, jones, bojilova, higgins}@nceas.ucsb.edu

National Center for Ecological Analysis and Synthesis (NCEAS)
University of California, Santa Barbara

Abstract

The ecological sciences represent a challenging community from the perspective of scientific data management. Ecological data are collected by investigators who are spread out over a large geographic area and who are using a wide variety of research protocols and data handling techniques. The resulting heterogeneous data are stored in autonomous database systems that are dispersed throughout the ecological community. The Knowledge Network for Biocomplexity seeks to address these issues through the use of structured metadata encoded in Extensible Markup Language (XML). The main goal of this project has been to design and implement a schema-independent data storage system for XML which is called Metacat. Metacat uses a hybrid XML storage approach using a commercial RDBMS backend while still allowing any arbitrary XML document to be stored. This paper describes the Metacat XML data storage system and its relevance to scientific data management in the ecological sciences.

1. Introduction

The ecological sciences represent a challenging community from the perspective of scientific data management. Ecological data are collected by investigators using a wide variety of protocols to address complex topics ranging from marine bacterial community function to global carbon flux. The resulting heterogeneous data are stored in autonomous database systems that are dispersed throughout the ecological research community. Thus, data management for ecological analysis and synthesis requires that one address these three primary characteristics of ecological data: heterogeneity, dispersion, and autonomy.

The Knowledge Network for Biocomplexity project [7] seeks to address these issues through the use of flexible and extensible metadata standards that accommodate the tremendous variety of data representation formats and schemas found in ecology. We have taken the approach of standardizing metadata exchange syntax using Extensible Markup Language (XML) [1] to allow for a higher degree of interoperability among the distributed research groups in the discipline. Thus, one important challenge for the project has been to design and implement a schema-independent, scalable data storage system for XML data that can be used at distributed ecological research sites and permit autonomous control of data by site researchers.

Relational database systems have been used in a variety of systems for storing XML data [3][10], generally by either storing the data as a large string object, or by mapping the XML schema onto the relational schema [2][9] using template driven systems. In the former case, the XML data is not searchable except through a crude free-text mechanism. In the latter case, the schemas of the XML documents must be predetermined to allow for the mapping to a fixed relational schema, thus limiting the flexibility of the system. Consequently, there is a trend towards dedicated XML databases [6][8] that allow storage of XML documents with arbitrary schemas, but these databases have yet to mature to support the enterprise features commonly found in relational database management systems. Thus, we have developed a hybrid approach by storing XML data with arbitrary schemas in a relational database while still permitting structured, path-based queries. We do this by modeling the structure of XML itself rather than the schema represented in the document.

Any XML document, regardless of the schema it represents, can be stored in the Metacat system. Once stored, the information in the XML document can be queried using a path-oriented query language. Each feature of Metacat has been designed with maximum flexibility in mind. Even though its original purpose was to fulfill the needs of the ecological community and their metadata storage needs, Metacat is by no means limited to this. Because of its RDBMS backend, it is as scalable as the particular RDBMS used to store Metacat data. With its advanced query capabilities, almost any SQL statement can be represented (through the HTTP front end) and executed against the internal schema of a particular XML document type.
In this paper we describe the Metacat XML data storage system and its relevance to scientific data management in the ecological sciences.

2. Schema-independent XML Storage

Metadata used by the KNB project is encoded in an XML format. An XML document is structured as a tree of nodes where the root node is the document entity and children of the root node are elements and attributes. The tree’s leaves are typically character data nodes. The Document Object Model (DOM) [11] is an effective way of modeling the structure of XML data. Metacat uses the DOM object model to store XML documents in a relational database. Metacat does not, as of yet, implement the complete DOM API although that is a future goal of the project.

![Figure 1: Basic entity relationship diagram for the Metacat database.](image)

The relational model differs substantially from this model of an XML document, and therefore the XML tree must be decomposed into its constituent nodes so that it can be represented in a table. Metacat does this by parsing the XML document (Listing 1) into a series of DOM nodes and then inserting each node as a record into a database table (Figure 1, Table 1). The table has a defined recursive foreign key, which allows each record (representing a node in the XML tree) to point to its parent. Since XML is a tree structure, each node may have only one parent.

```xml
<dataset>
  <ds_id>12345</ds_id>
  <creator>
    Jim Bob
  </creator>
  <desc>
    <title>
      Red Abalone along the Santa Barbara Coast
    </title>
    <dept>
      Marine Biology
    </dept>
  </desc>
</dataset>
```

Listing 1: Example XML metadata document.

The relational table contains all of the information needed to completely reconstruct the XML document. The parentnodeid is the recursive foreign key (see figure 1), which allows each child node to link back to its parent node. The rootnodeid field allows a query hit on a particular node to be quickly linked back to its document root.

<table>
<thead>
<tr>
<th>Nodeid</th>
<th>Nodetype</th>
<th>Nodename</th>
<th>Nodedata</th>
<th>parent nodeid</th>
<th>rootnodeid</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>DOCUMENT</td>
<td>dataset</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>ELEMENT</td>
<td>ds_id</td>
<td>12345</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>TEXT</td>
<td>desc</td>
<td>Jim Bob</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>TEXT</td>
<td>title</td>
<td>Red Abalone along the Santa Barbara Coast</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>ELEMENT</td>
<td>creator</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>TEXT</td>
<td>dept</td>
<td>Marine Biology</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>ELEMENT</td>
<td>title</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>ELEMENT</td>
<td>dept</td>
<td>6</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>TEXT</td>
<td>desc</td>
<td>Jim Bob</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>ELEMENT</td>
<td>dept</td>
<td>6</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>TEXT</td>
<td>dept</td>
<td>Marine Biology</td>
<td>9</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1: Nodes of an XML document are recorded as records in a relational table.

3. Path-based Queries

Query Efficiency

Given this ability to store metadata documents encoded in XML using arbitrary schemas, one must then
address the issue of querying the Metacat document store in an efficient manner. A major problem with storing XML documents in this decomposed DOM model is that searching by a specific path in the document becomes a nested query. Listing 2 shows an SQL statement for finding a dept name of Marine Biology.

```
SELECT rootnodeid FROM xml_nodes
WHERE nodedata LIKE 'Marine Biology'
AND parentnodeid IN
(SELECT nodeid FROM xml_nodes
WHERE nodename LIKE 'dept'
AND parentnodeid IN
(SELECT nodeid from xml_nodes
WHERE nodename like 'desc'
AND parentnodeid IN
(SELECT nodeid from xml_nodes
WHERE nodename like 'dataset')));
```

**Listing 2: Nested SQL query for searching the DOM structure.**

Our initial reaction to this type of query was that it was much too expensive to perform on a regular basis, especially as the database scales to large numbers of records. Because of this, we developed an alternative method to execute queries based on a pre-computed table that contains all of the paths through the document. This table is an index, which allows Metacat to quickly locate paths in the XML data. The records in this table are created in an indexing phase that occurs when a document is inserted or updated.

```
SELECT rootnodeid FROM xml_nodes
WHERE nodedata LIKE 'Marine Biology'
AND parentnodeid IN
(SELECT nodeid FROM xml_index
WHERE path LIKE '/dataset/desc/dept');
```

**Listing 3: Streamlined SQL query possible because of path indexing.**

Our expectation was that the performance difference between the SQL statements in Listing 2 and 3 would be great, and that the nested query would become slower as the depth of the paths being searched increased. This is true when the structure of the documents within Metacat are very constrained. Figure 2 shows a test done with a highly structured document which contained 16 mixed content nodes all nested inside one another. The graph shows that while using the xml_index query, the depth 1 and the depth 16 query times increase together linearly with respect to the number of nodes in the database. When the index is turned off and nested queries are used, query time increases linearly in both the depth 1 and depth 16 queries, however, the depth 16 query diverges from the path of the depth 1 query as the number of nodes in the database becomes greater showing a slowing query time. The graph also shows that when using the indexed search, the query time is a much steeper curve, showing that the index significantly reduces query time overall.

However, other tests have shown that with less structured documents than the one used in the test depicted by figure 2, the index becomes less of a factor in the time that it takes to perform a query. When testing with a much less structured document set (we used the Old Testament marked up in XML which has a maximum depth of 6 and approximately 80% of the 3.32 MB XML file resides in the “verse” tag) the nested query actually out performed the index based query. Figure 3 shows the results of the Old Testament test. In this case, the number of nodes which Metacat must do string matching against (in other words, the content of all of the nodes that match the path expression) is the indicator of query time.

The conclusion that we have drawn from these sets of tests is that the performance of the database is highly dependent on the structure of the documents that are stored in the system. Metadata documents in the ecological community tend to be moderately structured (not as structured as the document used in the figure 2 test but much more structured than the Old Testament document) and on average they have a maximum depth of...
less than 10. Therefore, it is our conclusion that Metacat should perform well on the document base for which it is intended.

Figure 2: Plot of depth 1 (light lines) and depth 16 (dark lines) queries using the indexed query and the nested query. The structure on the top represents indexed queries and the structure on the bottom represents nested queries. The vertical variance in the curve is representative of slight variations in query time. This is thought to be due to other processes and users on the same system at the time of the test.

Figure 3: Query performance of the Metacat system as a function of the node depth of the query when using the pre-computed path index (solid line) and when using nested query (dashed line). In this graph, the repository consisted of $2.03 \times 10^6$ nodes. Shorter times are better. The peaks at depth 0 (free text search) and depth 5 are explained by the much larger numbers of nodes (23,000/25,000 text nodes are at depth 5) that satisfy the path expression match in the test document set and are therefore used in the text matching.

In addition to increasing query performance in some cases, the index also enables other functionality. Because we index both absolute and relative paths through the document, Metacat is able to perform a subset of XPath\footnote{XPath is a W3C recommendation for addressing parts of an XML document. It defines a standard addressing scheme for showing the precise location of a node in an XML document. It is a goal of the Metacat project to fully implement XPath. At this point, only partial XPath support has been implemented.} type queries. This is useful if there are multiple document types that have similar but not identical structures. For instance document type A may have a `/marine/dataset/creator` element. Document type B may have an `/datalist/dataset/creator` element. By storing relative paths, a query for a `dataset/creator` of "Jim" could return hits in both A and B. If the index were limited to absolute paths only, then a single query might return hits in either A or B, but not both.

Query Specification Format

Clients access the Metacat query engine by passing an XML-encoded version of the path query to the query engine. The XML query is marked up according to a DTD that allows for Boolean logic and partial string matching (see pathquery.dtd in the distribution \cite{7} for details). For example, a simple case-insensitive, substring match on
the string "12345" in the path "/dataset/ds_id" is shown in Listing 4. To specify free-text searches (i.e., to not constrain the search to particular paths), one simply omits the <pathexpr> element from the query term specification.

```
<pathquery>
  <querygroup operator="UNION">
    <queryterm casesensitive="false" searchmode="contains">
      <value>12345</value>
      <pathexpr>/dataset/ds_id</pathexpr>
    </queryterm>
  </querygroup>
</pathquery>
```

**Listing 4: Example query specification using the pathquery.dtd schema.**

Boolean logic of arbitrary complexity is specified using a combination of <querygroup> and <queryterm> elements. Within a query group, all query terms are combined using either logical AND or logical OR. Within a query term, attributes can be provided to specify whether the substring matching is case-sensitive and whether to perform an exact match or one of several types of substring match (contains, starts-with, ends-with). Each query term is equivalent to an XPath expression.

**Query Result Set**

Results of queries are returned as an XML document that includes both the query used to generate the results and a list of the documents that satisfies the query. By default, the result set returned contains the document identifier (docid), name (docname), type (doctype), creation date (createdate), and modification date (updatedate) of each document that satisfies the query. Because the XML documents stored in Metacat have arbitrary schemas, constructing a result set that contains additional data from the documents is not as simple as it is in the fixed-schema case. We have enabled the clients to determine which parameters from an XML document are returned to the user as part of the query encoding using the returnfield element.

```
<resultset>
  <query>
    <querygroup operator="UNION">
      <returnfield>/dataset/creator</returnfield>
      <returnfield>/dataset/desc/title</returnfield>
      <returnfield>/dataset/desc/dept</returnfield>
      <returnfield>
        <queryterm casesensitive="false" searchmode="contains">
          <value>12345</value>
          <pathexpr>/dataset/ds_id</pathexpr>
        </queryterm>
      </returnfield>
    </querygroup>
  </query>
  <document>
    <docid>metacat.1</docid>
    <docname>dataset</docname>
    <doctype>dataset</doctype>
    <createdate>2001-02-09 13:58:21</createdate>
    <updatedate>2001-02-09 13:58:21</updatedate>
    <param name="dataset/creator">Jim Bob</param>
    <param name="dataset/desc/title">Red Abalone along the Santa Barbara Coast</param>
    <param name="dataset/desc/dept">Marine Biology</param>
  </document>
</resultset>
```

**Listing 5: Example XML result set showing a pathquery that specifies additional data fields to be returned from the query and those parameters for each matching document.**

The returnfields are returned in param elements with an attribute of name set to whatever the requested returnfield was. These param elements can be parsed out using an XSL stylesheet to be displayed in whatever form is required.

The query system described here allows data managers to determine the types and formats of metadata that are appropriate for their research needs and construct queries against that metadata in a powerful and flexible way.

4. Replication

Metacat was originally designed as a centralized ecological metadata server. It was soon realized, however, that multiple research stations needed to have a local Metacat server that could share data with other Metacat servers from different sites, thus maintaining local autonomy over metadata and datasets while still allowing the broad data sharing that was in the original plan. The resulting Metacat replication scheme was based upon three concepts. The first was that the data must remain consistent on each server. The second was that some
servers may want to share their data with other Metacat servers but not want to receive outside information onto their servers. This is known as one-way-replication. The third concept was that a document would have a home server and the master copy would be kept on the home server. If the home server went down or could not be contacted, documents from it could not be updated. This prevents an agency’s documents from being updated without their knowledge (and permission).

Metacat uses two different mechanisms to replicate documents among servers (see figure 5). It has a time checking system that checks each replicated server on a given time schedule. It also has event-based notification (document insertion, deletion and update) so that each server is notified when a change occurs.

Metacat uses file locking to maintain consistency among multiple versions of documents. If a user on server A tries to update a document whose home server is B, server A must first get a lock on the document before requesting the update. If B rejects the lock, the user on A must re-checkout the document, make changes and then attempt to reinsert it. Once a lock is granted, the document will be updated on A and B and then B will replicate the changes to all of the other servers it has registered. Only a document's home server can initiate the replication of a document that it owns and a document belonging to a certain server will only be copied by another server from its home server. Consequently, both servers involved in replication (source and destination) must agree to the transfer before it occurs. If one of the servers in the transaction wishes to do one-way-replication only, then a flag can be set for it to send documents out only.

The easiest way of implementing replication would have been to allow the backend database (i.e. Oracle, SQLServer, etc.) to handle it, however this would limit sites to a specific RDBMS or not allow sites using different backend databases to replicate amongst one another. Using the RDBMS’s built in replication would violate the autonomy requirement of the Metacat project. Each site must have total control over its metadata documents and not be limited to a particular RDBMS. Thus, through this flexible, document based replication scheme, Metacat allows individual scientists and their organizations to share their data or metadata without relinquishing their autonomous control over it. The Metacat replication mechanism was meant to address the
dispersion problem in that Metacat servers are configured to replicate data from all of the registered research sites, thus providing a cross-organizational metadata-based search facility. This scheme fulfills that goal without sacrificing the autonomy of individual sites.

5. Access Control & Authentication

Because Metacat is a multi-user system, controlling access to the documents in the system (and to the system itself) becomes essential. Metacat has two levels of access control: user and document level. User level authentication is performed through a connection to an existing authentication system such as a Lightweight Directory Access Protocol (LDAP) server. The interface to authentication systems is flexible such that Metacat can use any authentication service specified by the administrator (not just LDAP).

Once a user is authenticated, he can control the document level access restrictions on his documents or any documents to which he has been given control. To set the permissions, the user inserts a separate XML document, called the Access file (see eml-access.dtd in the distribution [7]). The Access file allows the owner of an existing XML document in the system to give individual or group read or write access to the document. It also allows the user to specify other users who can change permissions on the document as well as to allow or disallow public (unauthenticated) users read privileges.

A given Access file can be applied to one or more XML documents in the system. This allows the user to set several different access schemes and apply them arbitrarily to his documents in the system without having a one-to-one relationship of Access files to XML documents. When a document is replicated to another server, its Access file is also replicated assuring global control over documents.

Metacat accepts Access files for setting permissions on documents in the system, however, the actual user does not need to know of the existence of the Access file for his document. For instance, the Morpho graphical user interface Client (discussed in Section 6) hides the existence of the Access file and provides a simple interface for setting document permissions.

6. Programmatic Interface

All of the XML storage capabilities of Metacat would be useless if the database was not accessible from some sort of front-end interface. In order to provide maximum flexibility, Metacat is accessible through an HTTP based servlet. All functionality is accessible through this interface.

Currently, there are two clients that have been created to use this interface. The easiest to create is the web interface. Because Metacat is accessed via a servlet, form-based web pages can send the appropriate commands and data. The web-based client is extremely flexible allowing different organizations to have their own "look and feel" without compromising functionality. This feature also allows different organizations to customize their search fields to correspond to the paths in the document types that they use.

The other client that is currently in development is a Java application called Morpho. It is aimed at the ecological community as a complete data management tool. It communicates with the servlet just as the web interface does. It contains an XML editor that can create, edit and write XML documents to/from the database as well as other graphical tools for managing metadata in Metacat.

Data is passed to Metacat using servlet parameters that can be encoded in either a URL (if using HTTP GET) or as part of an HTML form (if using HTTP POST). The action parameter controls the type of processing handled by the servlet, and includes 15 possible values (the main actions include delete, insert, update, login, logout, query, squery, validate and read).

To accommodate the cross-site, flexible metadata storage system described here, XML documents stored in Metacat can automatically be transformed into another XML document type or HTML, allowing flexibility in metadata presentation and exchange.

Figure 6: Architectural overview of Metacat and associated client applications.

7. XSL Transformations

Metacat facilitates the automatic transformation of XML documents pulled from the system. This enables the creation of custom resultset views and also allows the automatic translation of one form of metadata to another. This makes the cross-format sharing of metadata between autonomous sites more convenient to the user and thus
improves the willingness of the user community to share data.

This transformation takes place when a user reads a document from the database. It is facilitated by an internal cross-reference which links each document type to an XSLT [4] stylesheet that defines the transformation. For example, the document type “-//NCEAS//resource//EN” could be crosslinked to “http://www.nceas.ucsb.edu/stylesheets/resource.xsl.” Thus, whenever a document of type -//NCEAS//resource//EN was extracted from the database, the resource.xsl stylesheet would be used to transform it. The end user never sees the original form of the document and does not even need to know that the transformation has taken place. The transformation function can also be dynamically turned off so that an advanced user or data manager can view the XML without transformation.

The automatic transformation of documents facilitates data and metadata sharing among autonomous sites because one site's metadata can be automatically transformed into the form of another's. The XML based metadata could also be transformed into an independent metadata standard which promotes the use of national or regional standards while facilitating local autonomy and inter agency sharing.

Since Metacat returns resultsets encoded in XML, resultsets can also be automatically transformed. This adds to the flexible interface described in section 6. Resultsets can be transformed into various forms including tables, lists, etc. as well as being changed to a proprietary format for a data viewer.

8. Conclusions

Metacat is attempting to meet scientific data management needs of the ecological community by focusing on three major issues: heterogeneity, dispersion and autonomy. Data heterogeneity is being dealt with through the storage of structured metadata encoded in XML. Metacat's replication mechanism allows a dispersed community to bring their data together into a central searchable location, yet it allows dataset owners to retain autonomous control.

The hybrid approach used in Metacat to store XML documents gives the flexibility of a dedicated XML database coupled with the enterprise features of a commercial RDBMS. With flexibility in mind, Metacat was designed to be used by more than just the ecological community. In fact, Metacat could be used by anyone who wants to store structured XML data or metadata.

Future plans for Metacat include the implementation of the full DOM API and full XPath compliance. The DOM API would make XML documents on Metacat persistent, in that a document could be edited and accessed without having to be removed from the database, which must be done now. Full XPath compliance will allow standard structured path queries to be executed. Both of these enhancements will make Metacat more standardized and grant even greater flexibility to users.

Metacat is a tool which allows data managers to store heterogeneous data in a centralized searchable manner. This will facilitate data sharing that was previously not possible. Communities of researchers who never thought their data could be useful to another community will find that raw data can be used in differing ways from which it was originally intended. Metacat, with its flexible storage and retrieval system, makes this possible.

9. Acknowledgments

The authors wish to acknowledge the contributions of numerous colleagues without whom, Metacat would not exist in its current form.

Two who deserve special recognition are Mark Schildhauer of NCEAS, who has worked tirelessly as our "end-user advocate" constantly questioning our ideas on what a good interface should be, and Arcot Rajasekar of the San Diego Super Computer Center, who has helped with the product design with his insights in distributed computing.

This work is hosted by the National Center for Ecological Analysis and Synthesis, a Center funded by NSF (Grant #DEB-0072909), the University of California, and UC Santa Barbara.

This material is based upon work supported by the National Science Foundation under Grants No. DBI99-04777 and No. DEB99-80154. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

10. References


