A Hybrid XML-Relational Grid Metadata Catalog

Scott Jensen, Beth Plale, Sangmi Lee Pallickara, and Yiming Sun
Department of Computer Science, Indiana University
{scjensen, plale, leesangm, yimsun} @cs.indiana.edu

Abstract

The ability to manage metadata is a critical requirement of the grid, but scientists have not been given the tools needed to catalog experimental data based complex metadata attributes. Our research has shown that the specific characteristics of metadata catalogs require a different approach than that used for general queries over XML data. This paper presents a hybrid approach to storing XML in a relational database that exploits the specific characteristics of a metadata catalog.

Index Terms — Metadata, Cyberinfrastructure, Grid Computing, XML, Databases, LEAD.

1. Introduction

The scientific community has identified the need to develop cross-domain data catalogs that can be queried based on rich sets of metadata so research can be leveraged to the greatest extent possible. The NSF's Blue-Ribbon Advisory Panel on Cyberinfrastructure noted that "multidisciplinary, well-curated federated collections of data" should be part of the infrastructure, and that "A significant need exists in many disciplines for long-term, distributed, and stable data and metadata repositories that institutionalize community data holdings"[1]. In the UK, the Central Laboratory of Research Councils (CLRC) is bringing together data from multiple science disciplines with an aim of providing a single cross-discipline method for browsing and searching metadata [2].

Back in 1992, the Federal Geographic Data Committee (FGDC) started working on a cross-discipline metadata standard for describing spatial data products [3], with the eventual development of a geospatial data clearinghouse as part of the National Spatial Data Infrastructure (NSDI). While clearinghouse networks such as the NSDI Clearinghouse provide a means to catalog metadata for sharing publicly available datasets, the opportunity to gather the most detailed and accurate metadata to describe the source data and experimental results is when the data or experiment results are first generated or augmented with additional insights generated during the scientific process. For this metadata to be captured, the cyberinfrastructure must include a personalized metadata catalog that can be used to both capture the metadata as it is generated and provide the ability to catalog and query on-going experiments; while also being able to ensure the privacy of unpublished data and results. As with metadata catalogs such as the NSDI or CLRC, grid environments capture and exchange metadata using one or more XML schemas common to their community [4][5].

Since metadata in a grid environment is exchanged in an XML format, a metadata catalog must be able to ingest metadata in an XML format and respond to queries using the schema of its grid community. Although a native XML database such as Xindice [6] may seem to be an obvious choice for the backend store, prior work by our group showed Xindice to be far inferior to a relational database in terms of throughput. The alternative is storing and querying XML using a relational database [7]. In a schema aware environment such as a grid, current research on publishing XML from relational databases, (and storing XML data in relational databases) has focused on a lossless shredding of XML into relational tables based on an inlining approach. Our research on personal metadata catalogs in the Linked Environments for Atmospheric Discovery (LEAD) [9] project has identified characteristics of a metadata catalog that suggest a modified architecture for storing and querying metadata as XML in a relational database.

Although scientists have a number of requirements related to the ability to manage data in a grid environment [8][9], not all of these directly impact the approach used to store and query metadata. In this paper, we show that the following three characteristics are key to the architecture of a metadata catalog and taken together suggest an alternate storage approach:

- *Queries Over Metadata Attributes:* Since metadata is "data about data", a metadata catalog is used to store properties of the data scientists have used in experiments and the products generated. In myLEAD these properties are referred to as metadata attributes. In querying a metadata catalog, scientists are looking for objects (files or aggregations) that have a certain range of values for specific metadata attributes. This differs from the prior research that has focused on being able to respond to arbitrary queries over XML stored in a RDBMS.
- *Unordered Queries Generating Ordered XML Results*: In the metadata catalog, scientists are not particularly concerned with the order or structure of the XML documents, but query responses require reconstructing the original schema-based documents.
- *Validated Dynamic Metadata Attributes:* When XML is shredded for storage in relational tables, the relevant schemas are usually used to define the structure of the tables and the relations between tables. However, one of the key requirements of a metadata catalog is that scientists need the ability to define new properties of the data products that will be described by the metadata. For metadata queries to be meaningful, scientists must be able to define the structure of these properties and validate the data – but without changing the XML schema used to communicate in the grid environment. The schema alone does not validate the metadata being communicated in the XML documents being stored.

The contribution of this research is a new hybrid XML/Relational approach to shredding, storing, and querying scientific metadata in a grid environment that leverages the distinct characteristics of metadata storage. Due to the focus of a metadata catalog on locating objects meeting specified criteria, the shredding of XML data for querying can be separated from the storage of metadata attributes as CLOBs for use in reconstructing XML documents for query responses. This eliminates the need for achieving lossless shredding from XML since the shredded data is no longer needed to construct the XML documents returned in query responses. Although our research has been in the context of the LEAD grid, this approach generalizes to metadata in other scientific grid environments as well.

This paper is organized as follows. Section 2 provides an overview of the hybrid approach we propose. Sections 3, 4, and 5 discuss shredding XML, querying, and building the query response using a hybrid approach. Section 6 discusses related work, and we conclude with future research in Section 7.

2. A Hybrid Approach

When storing XML data in a relational database there are two main approaches – storing as an XML string in a Character Large Object (CLOB), or extracting individual data items by shredding the XML into relational tables [10][14][16][17][25]. For metadata catalogs, we advocate a hybrid approach. In a hybrid approach, the metadata is shredded into both CLOBs and relational tables as illustrated in Figure 1.

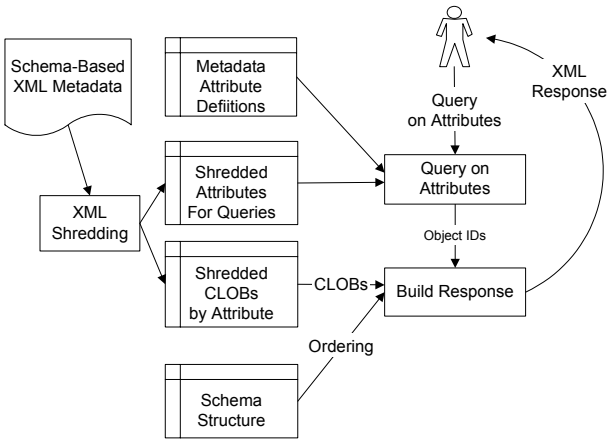

Fig. 1 Hybrid Approach

Under the hybrid approach, the XML schema used to communicate the metadata is first partitioned into metadata attributes based on the set of rules listed below; with each metadata attribute representing a single concept contained in the schema. To allow for complex concepts, each metadata attribute can contain multiple sub-attributes - without limit as to the nesting of sub-attributes. Within both metadata attributes and sub-attributes, metadata elements define the actual data values contained within the attributes. The metadata elements are always leaf nodes in the schema and the metadata attributes and sub-attributes are always interior nodes in the schema (except for those nodes that are both a metadata attribute and a metadata element). As an illustration of the hybrid approach, Figure 2 shows a portion of the LEAD schema with each metadata attribute or sub-attribute **bolded** and each metadata element *italicized*.

The following rules apply in determining which elements in the schema are defined as metadata attributes:

- Metadata attributes should define a concept. As an example, in Figure 2 the "status" metadata attribute contains two metadata elements: *progress* and *update*. Together these two elements define the concept of document status.
- If an element allows for multiple instances, then it must be contained within a metadata attribute - it

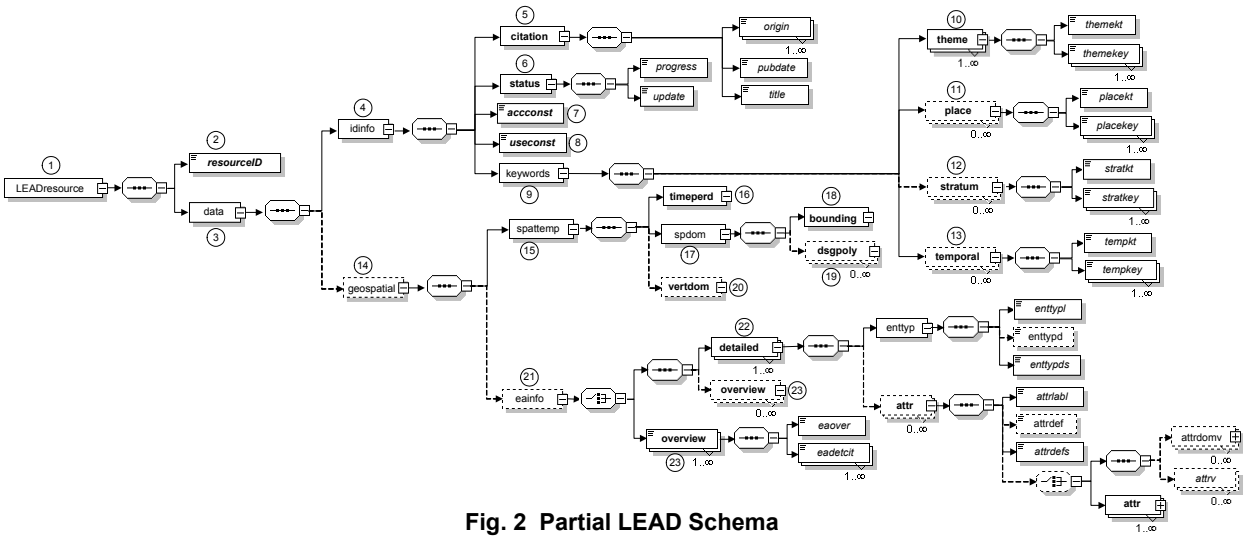

Fig. 2 Partial LEAD Schema

cannot have a metadata attribute start below it (except for sub-attributes). If multiple instances are allowed for an element in the schema, then that element most likely defines a concept. In the example schema, the "theme" element is a metadata attribute that defines the concept of a theme keyword (or set of related keywords).

- Any element in the schema that has attribute nodes (XML attributes) must either be a metadata attribute or be contained within a metadata attribute. The LEAD schema does not contain any XML attribute nodes, but attribute nodes can be defined as metadata attribute/elements.
- Any recurrence in the schema must be contained within a metadata attribute. In Figure 2, the "attr" element allows for recursion in that it can contain child attr elements. This recursion is contained within the detailed element which is defined as a metadata attribute.
- Every leaf element in the schema must be contained within a metadata attribute, but each metadata attribute does not need to be queryable. Those metadata attributes that a scientist may want to use as criteria in a query are known as *queryable metadata attributes*.

After the elements in the schema that are metadata attributes have been identified, an ordering of the schema nodes is created. As discussed in [19], a total ordering of the nodes in each document is required for reconstructing an ordered document in response to a query. Three possibilities proposed in [19] are global ordering (using a pre-order depth-first traversal), local ordering (children of an element are numbered independent of the children of other elements), and Dewey ordering based on the Dewey decimal classification approach. However, since the hybrid approach stores a CLOB for each metadata attribute during the shredding process, only those elements that are metadata attributes or higher in the schema need to be ordered – the elements within the CLOB are inherently in their original order. Since all of the elements in the schema that have recursion or a potential cardinality greater than one are contained within a metadata attribute, the ordering can be done once for the schema instead of having to be created for each document. In Figure 2, this node ordering is displayed as circled numbers next to each node.

The most efficient ordering of nodes would be same-sibling order as defined in [19], but as the authors note, it does not provide a total ordering of the nodes in the document. However, since the hybrid approach uses a global ordering based on the schema, this can be combined with a same-sibling ordering to create a total ordering for those metadata attributes which allow for multiple instances (such as the theme element in Figure 2). A table in the catalog contains this global ordering based on the schema – tracking for each node its order, tag, and the order of its last child (which for metadata attribute nodes is the same as the node order). The catalog also contains tables with the definition of each metadata attribute and metadata element. For each metadata attribute the definition includes a unique internal ID, the schema order, and parent metadata attribute ID (in the case of sub-attributes). Each metadata element is also assigned a unique ID and the catalog tracks the associated metadata attribute and data type. Each metadata element definition is associated with a single metadata attribute definition.

Metadata attributes such as the "status" element in Figure 2 are defined based on the structure of the schema and we refer to these as structural metadata

attributes. However, one of the three distinct characteristics of a metadata catalog is the need to provide for validated dynamic metadata attributes. Such attributes are not directly based on the structure of the schema, and are the cause of recursion in the schema used to communicate metadata in a grid environment. In Figure 2, the subtree rooted at the detailed element addresses the need for dynamic metadata attributes.

3. Shredding Using the Hybrid Approach

Figure 3 contains an example XML fragment based on the schema shown in Figure 2.

```
<Leadresource>
    <resourceID></resourceID>
    <data>
        <idinfo>
            .
            .
            .
            <keywords>
                <theme>
                    <themekt>CF NetCDF</themekt>
                    <themekey>convective_precipitation_amount</themekey>
                    <themekey>convective_precipitation_flux</themekey>
                </theme>
                <theme>
                    <themekt>CF NetCDF</themekt>
                    <themekey>air_pressure_at_cloud_base</themekey>
                    <themekey>air_pressure_at_cloud_top</themekey>
                </theme>
            </keywords>
        </idinfo>
        <geospatial>
            .
            .
            .
            <eainfo>
                <detailed>
                    <enttyp>
                        <enttypl>grid</enttypl>
                        <enttypds>ARPS</enttypds>
                    </enttyp>
                    <attr>
                        <attrlabl>grid-stretching</attrlabl>
                        <attrdefs>ARPS</attrdefs>
                        <attr>
                            <attrlabl>dzmin</attrlabl>
                            <attrdefs>ARPS</attrdefs>
                            <attrv>100.000</attrv>
                        </attr>
                        <attr>
                            <attrlabl>reference-height</attrlabl>
                            <attrdefs>ARPS</attrdefs>
                            <attrv>0</attrv>
                        </attr>
                    </attr>
                    <attr>
                        <attrlabl>dx</attrlabl>
                        <attrdefs>ARPS</attrdefs>
                        <attrv>1000.000</attrv>
                    </attr>
                    <attr>
                        <attrlabl>dz</attrlabl>
                        <attrdefs>ARPS</attrdefs>
                        <attrv>500.000</attrv>
                    </attr>
                </detailed>
            </eainfo>
        </geospatial>
    <data>
<Leadresource>
```

Fig. 3 Metadata Document

As illustrated in Figure 1, each element in the document that represents a metadata attribute is stored as a CLOB in the catalog and also shredded into queryable metadata attributes and elements. For example, the two theme elements in Figure 3 are structural metadata attributes and each of the theme metadata attributes (in the bolded section) would be stored as a CLOB along with their global node ordering (10) and their sequence IDs based on same-sibling ordering (1 and 2). Each of the theme metadata attributes would also be shredded, and the metadata attribute definition is determined based on the tag for that element. Likewise, the metadata element definitions are determined based on their tag and their parent metadata attribute. For each metadata attribute and element the following data is stored:

Metadata Attribute:

- *Object ID* – the internal ID assigned to the object.
- *Attribute ID* – the internal ID assigned to the definition of the metadata attribute.
- *Sequence ID* – same sibling ordering.
- *CLOB Sequence* – same sibling ordering for CLOB.

Metadata Element:

- *Object ID, Attribute ID, and Sequence ID* – Primary key for parent metadata attribute.
- *Element ID* – the internal ID assigned to the definition of the metadata element.
- *Element Sequence* – local order within the attribute.
- *Element Value* – string or other data type.

The theme metadata attributes are structural attributes defined by the schema, whereas the detail element in Figure 3 illustrates a dynamic metadata attribute. Discussions early on in the LEAD project identified the need for a metadata catalog to also be able to capture complex attributes that may evolve with the continued refinement of the weather forecasting models used in LEAD. For example, both the Advanced Regional Prediction System (ARPS) [12] and Weather Research & Forecasting (WRF) [13] models use Fortran namelist files containing detailed model parameters which cannot be built into the structure of the schema because scientists must be able to define new parameters as they continue to enhance the models or create new models. In addition, the schema would grow to an unmanageable size if it had to accommodate all possible parameters. The need to address model parameters, and the general need to provide a means to define new complex metadata attributes, requires metadata catalogs to accommodate dynamic metadata attributes. The schema in Figure 2 addresses dynamic metadata attributes through the detailed element.

When the detail element is shredded, the metadata attribute definition is determined based on the name and source of the metadata attribute, but in the case of dynamic metadata attributes the name and source are based not on the element tag but instead on the values contained in the *enttypl* and *enttypds* elements, (which contain “grid” and “ARPS” respectively). Within the detailed element there are three attr child elements which based on the schema in Figure 2 can contain either attrv elements, (which indicates a metadata element) or attr elements (which indicates a sub-

attribute). In Figure 3, the first attr element is a sub-attribute and the last two are metadata elements. In the case of both sub-attributes and metadata elements, the name and source are determined based on the *attrlabl* and *attrdefs* elements. In shredding the structural metadata attributes, the element tag was used for the name, but the source was not necessary. Having both a name and source allows different scientific models such as the ARPS and WRF models to have metadata attributes with the same name based on their respective namelist files, but which may have different meaning or content in their respective models. The shredding validates the name and source of each dynamic metadata attribute with the definitions stored in the catalog. Any element in a document that does not match a defined metadata attribute is still stored as a CLOB, but the data is not shredded into the tables used to support queries. By validating dynamic metadata attributes on insert, the catalog provides a consistent, but dynamic set of definitions for query purposes that could also be connected to an ontology for enhanced search capabilities. Additional metadata attributes can be defined at both an administrator or user level, with those defined at the user level being private to a specific user.

In addition to shredding the metadata attributes and elements into their respective tables, for any metadata attribute that contains sub-attributes, (such as the "grid stretching" sub-attribute within the dynamic "grid" metadata attribute in Figure 3) the relationship between the sub-attribute and attribute is stored in a table which maintains an inverted list of the relationship between a sub-attribute and any parent metadata attribute as well as intervening sub-attributes.

In contrast to handling recursion in general XML documents, in a metadata catalog recursion is used to define dynamic metadata attributes. Although the structure would vary between schemas, the general idea is to allow for the definition of metadata properties not envisioned or captured in the schema structure. The hybrid approach to cataloging metadata benefits from this distinction because the recurrence "disappears" by handling dynamic metadata attributes based the name and source instead of the recursive structure of the document. A more general XML-Relational approach cannot take advantage of this distinction.

4. Querying Using A Hybrid Approach

Research on storing XML in an RDBMS has focused on converting queries written in XPath or XQuery into SQL and running them against a relational database. However, queries over a metadata catalog are looking for objects in what we refer to as unordered queries over metadata attributes. The purpose of a metadata query is to return those documents that contain metadata attributes meeting the criteria specified. We use the term "unordered queries" because only the values within the metadata attributes are important to the query. As an example, a scientist looking for all objects with horizontal grid spacing = 1000 meters that also have grid stretching with a minimum vertical spacing = 100 meters could issue the following XQuery FLWOR expression against the XML schema:

```
for $r in fn:doc("catalog.xml")/LEADresource
let $g :=
$r/data/geospatial/eainfo/detailed/enttyp
  [enttypl eq "grid" and enttypds eq "ARPS"]
let $d := $g/../attr[attrlabl eq "dx"
  and attrdefs eq "ARPS" and attrv eq 1000]
let $z := $g/../attr[attrlabl eq
    "grid-stretching"
  and attrdefs eq "ARPS"]/attr/[attrlabl eq
    "dzmin"
  and attrdefs eq "ARPS" and attrv eq 100]
return
  if (fn:exists($d) and fn:exists($z)) then
    $r
  else ()
```

In a metadata catalog, the path to the dynamic metadata attribute contained in the detail element (grid) is immaterial, the query is over metadata attributes and the question the scientist wants answered is "Which files contain the metadata attributes of interest to me?" The myLEAD metadata catalog has a simple Java API that allows users to construct metadata attribute queries:

```
MyFile fileQry = new MyFile ();
MyAttr gridAttr = new MyAttr("grid", "ARPS");
gridAttr.addElement("dx","ARPS",1000,MYEQUAL);
MyAttr stAttr =
  new MyAttr("grid-stretching", "ARPS");
stAttr.addElement("dzmin",100, MYEQUAL);
gridAttr.addAttribute (stAttr);
fileQry.addAttribute (gridAttr);
```

The MyFile instance created in these few lines is then sent as the criteria for the query method. From a user's perspective, they would not even see this since there is a GUI query tool available that prompts the user with the available attributes and elements and allows them to build a query graphically.

Since queries over the catalog are searching for items that contain certain metadata attributes, queries are first shredded to determine the number of metadata attribute criteria that must be met to satisfy the query. In our simple example, there is only the metadata attribute criteria named "grid", which in turn has one sub-attribute – "grid-stretching". After determining the required metadata attribute and element counts, the metadata criteria are inserted into temporary tables. Figure 4 illustrates the query process that is then used

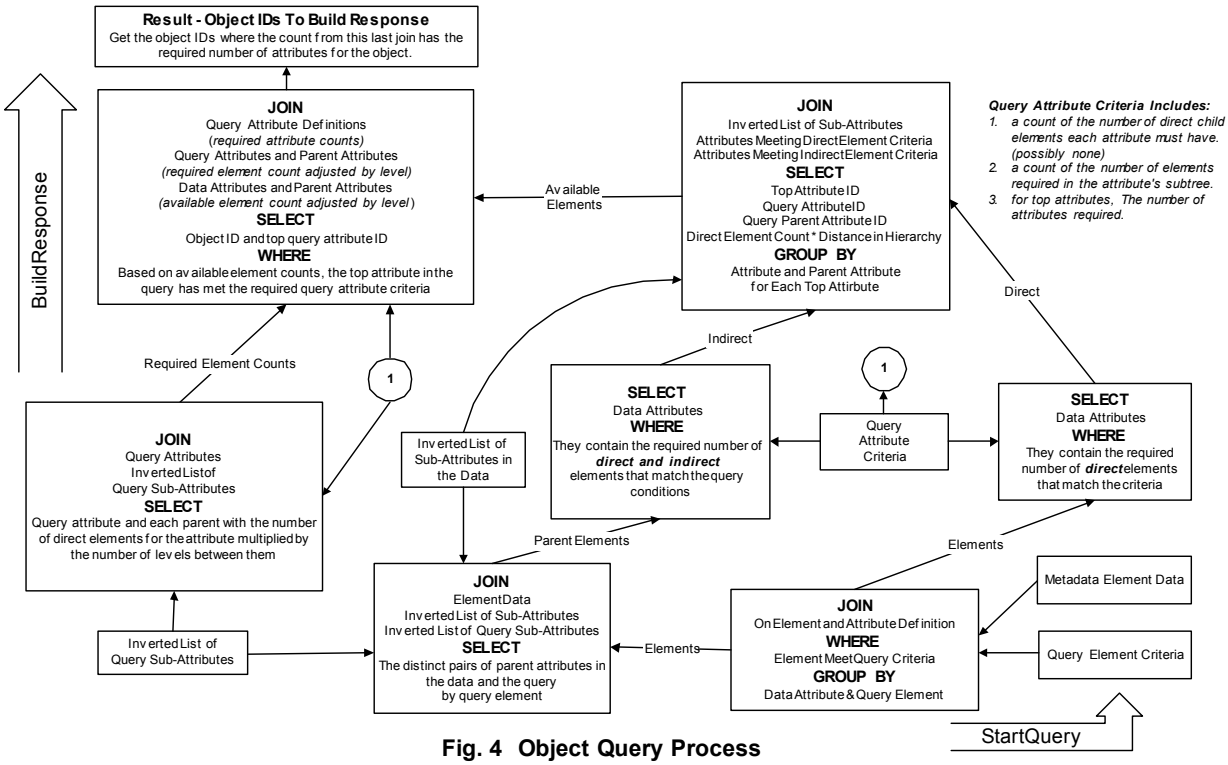

Fig. 4 Object Query Process

to determine which objects in the metadata catalog meet the query criteria.

This approach is based in part on inverted lists that track the relationship of sub-attributes to parent attributes, which allows the query to avoid recursion. If the attributes specified in the query do not have multiple instances within a single object in the data, or if there are not sub-attributes in the query criteria, then the query can be significantly simplified, so the physical implementation may differ – including possible partitioning of the data. The result of this query process is the internal IDs of the set of objects meeting the query criteria – the query response is then built from these IDs as described in the next section.

5. Query Response - the Hybrid Approach

In a metadata catalog, the query result as discussed above is the set of IDs for those objects in the catalog that match the metadata attribute criteria specified in the query. When the document was originally shredded, and as metadata attributes were inserted later, CLOBs were stored for each metadata attribute along with the object ID, the position in the document based on the global ordering in the schema, and a sequence ID for multiple instances of the same metadata attribute. In order to build the query response as shown earlier in Figure 1, the CLOBs are retrieved based on the set of object IDs generated by the query. The contents of the CLOBs for each metadata attribute are inherently ordered, but to build the response, we also need to add the element tags for all of the higher nodes in the document.

An inverted list is maintained for all of the nodes in the global ordering that maps each node in the ordering to those higher level nodes in the schema which are ancestors. This inverted list is joined with the table of CLOBs to determine the distinct set of ancestor nodes that are required for each object in the response (since many of the metadata attributes are optional, not all of the ancestor nodes are required). The set of required ancestor order IDs is joined with the table containing the global ordering to create all of the opening and closing tags for the ancestor nodes. Since the global order is built once and stored in a table in the catalog, the order of the last child element is maintained in that same table – allowing the opening and closing tags to both be added using set-based query operations instead of having to build an external tagger. Although the inverted list must be joined with the table containing the metadata CLOBs, (to determine the required ancestors) the CLOBs themselves are not needed at this point so the join can utilize the index without accessing the CLOBs until needed in the final join.

This approach is possible only because schema elements with cardinality greater than one as well as recursive elements are all contained within metadata

attributes – allowing us to define the global ordering at the schema level. If the global ordering was done at the document level as in [19], then the inverted list mapping CLOBs to their required ancestors would not be possible since the mapping would be different for every document.

The resulting set of required tags for higher level nodes is joined with the CLOB table for the final result returned to the myLEAD server. Since the global ordering allows all of the tags to be determined using set-based queries, no final tagging is needed at the server as in [24] – the results returned by the database are already tagged and can be returned to the client.

6. Related Work

There has been considerable research regarding storing XML data in an RDBMS with the aim of allowing the data to be queried using XPath or XQuery. Since prior work has not focused specifically on cataloging metadata, it could not exploit any characteristics specific to metadata catalogs. When an XML schema is available, research has focused on using an approach of shredding XML into relational tables using variations of a technique know as inlining [14][15][16][10]. Earlier research on shredding using a schema-less approach had often used the edge-table approach [10][17][16][18].

Under the inlining approach, elements are stored in the same relational table to the extent that the schema does not allow for cardinality greater than one. In those cases where multiple instances of an element are allowed, the subtree for that element is split into a separate table. In comparison to the edge-table approach where all edges are stored in the same table, inlining reduces the number of joins required [14]. However, this benefit would be significantly diminished for metadata catalogs since dynamic metadata attributes would be split into numerous tables due to the cardinality issue. The hybrid approach we present here would at first seem to be more similar to the edge-table approach. That approach views the XML document as a directed graph, and each tuple in the edge table represents either a connection between two nodes in the graph, or in the case of a leaf node, either the node value [16] or a pointer to separate value tables based on the data type [17]. However, as shown earlier, since queries over a metadata catalog focus on metadata attributes, we do not need to consider the full path, and through inverted lists can avoid the self-joins that hinder the edge-table approach.

One issue raised in [20] regarding inlining is that it is an unordered data model – so when reconstructing XML documents the system cannot ensure that the elements are in the original order. In a metadata catalog this could be problematic – such as in the LEAD schema where the lineage section tracks the process steps used to create a product. In [19] this was addressed through three approaches to creating a total order. In the hybrid approach we also have a total ordering of the elements, but through a global ordering based on the schema we avoid the update costs of maintaining a total ordering by document [19].

Other research has lead to systems that either allow for a hybrid approach or allow for the option of using either approach. IBM's DB2 XML Extender allows XML data to be saved as a CLOB in an "XML Column" or shredded into a set of relational tables known as an "XML Collection" [21]. As noted in [21], a hybrid approach that uses a combination of both storage methods may be desirable since the CLOB approach allows the document to be retrieved in its original form, while the shredded approach provides faster results. The default storage approach in Oracle's implementation of SQL/XML in version 10g of their product is to use a CLOB, but if a schema is available, the document can be shredded using an object-relational approach [11][22]. In both the IBM and Oracle implementations, the entire document is saved as a single CLOB. In contrast, the hybrid approach used in myLEAD is closer to that used in [15] where CLOBs are stored for the subtrees rooted at every element in the XML document (except the root and leaf elements). In myLEAD, CLOBs are stored for each element identified as a metadata attribute, and since there can only be a single metadata attribute on any path from the root of a document to a leaf node, our hybrid approach would not face the issue in [15] of possibly having high space overhead due to storing multiple BLOBs on the same path.

7. Conclusion and Future Work

Although our research in the LEAD project has focused on the requirements of the meteorological community, we believe many of these same issues apply to other scientific domains that need to manage significant volumes of data. Metadata management continues to be a pressing issue in other scientific domains, with a recent article mentioning metadata as one of the big challenges facing the Large Hadron Collider project [23]. Scientific metadata catalogs are likely to use different schemas tailored to their domain, but many of the issues in querying the catalog as well as a structure based on metadata attributes will be similar. The approach used in myLEAD can be used to create a framework for metadata catalogs that would be based on an annotated schema to indicate which

schema elements are structural or dynamic metadata attributes and elements.

In this paper we have shown how the distinct characteristics of metadata storage can be exploited using the hybrid approach as an alternative to inlining. Future work will focus on quantifying the benefit of the hybrid approach as applied to managing metadata.

Although not discussed in detail due to space limitations, one of the distinguishing features of myLEAD is its ability to perform complex context queries. However, challenges exist in presenting this query capability to users in an interface that is intuitive and easy to use. The current myLEAD GUI interface addresses queries from a containment viewpoint, but it does not address searching for objects based on a broader context. Presenting this capability to users remains a challenge.

References

[1] D. Atkins, K. Droegemeier, S. Feldman, H. Garcia-Molina, M. Klein, D. Messerschmitt, P. Messina, J. Ostriker, and M. Wright. "Revolutionizing Science and Engineering Through Cyberinfrastructure," Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure, January 2003.

[2] B. Mathews and S. Sufi, (ed. K. Kleese van Dam) The CLRC Scientific Metadata Model, Version 1, DL TR 02001, February 2001.

[3] Federal Geographic Data Committee, Content Standard for Digital Geospatial Metadata Workbook Version 2.0, Federal Geographic Data Committee. (5/1/2000).

[4] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. "Taverna: Lessons in creating a workflow environment for the life sciences," in *Concurrency and Computation: Practice and Experience*, Pub. John Wiley & Sons, Ltd. Published Online 13 Dec 2005.

[5] L. Cinquini. "Metadata development for the Earth System Grid," *NIEeS Workshop*, Cambridge (UK), Sept. 2002.

[6]Apache Xindice. http://xml.apache.org/xindice/.

[7] B. Plale, C. Jacobs, S. Jensen, Y. Liu, C. Moad, R. Parab, and P. Vaidya. "Understanding Grid Resource Information Management through a Synthetic Database Benchmark/Workload," *4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004)*, April 2004.

[8] S. Lee, B. Plale, S. Jensen, and Y. Sun, "Structure, sharing, and preservation of scientific experiment data", *IEEE 3rd International Workshop on Challenges of Large Applications in Distributed Environments*, July 2005.

[9] B. Plale, D. Gannon, J. Alameda, B. Wilhelmson, S. Hampton, A. Rossi, and K. Droegemeier, "Active Management of Scientific Data," in *IEEE Internet Computing special issue on Internet Access to Scientific Data*, Vol. 9, No. 1, Jan/Feb 2005, pp. 27-34.

[10] D. Draper, "Mapping Between XML And Relational Data," in *XQuery from the Experts*, Howard Katz, Ed. Boston: Addison Wesley 2004, pp. 309-352.

[11] Oracle Database 10g Release 2 XML DB Technical Whitepaper, May 2005.

[12] The Advanced Regional Prediction System. http://www.caps.ou.edu/ARPS.

[13] The Weather Research & Forecasting Model. http://www.wrf-model.org.

[14] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton, "Relational Databases for Querying XML Documents: Limitations and Opportunities," in *Proceedings of the 25th VLDB Concference*, Edinburgh, Scotland, 1999.

[15] A. Balmin and Y. Papakonstantinou, "Storing and querying XML data using denormalized relational databases," *The VLDB Journal*, Vol. 14, Issue 1, pp. 30-49, March 2005.

[16] J. Shanmugasundaram, E. J. Shekita, J. Kiernan, R. Krishnamurthy, S. Viglas, J. F. Naughton, and I. Tatarinov, "A General Technique for Querying XML Documents using a Relational Database System.," *SIGMOD Record*, Vol. 30, No. 3, pp. 20-26, 2001.

[17] D. Florescu and D. Kossman, "Storing and Querying XML Data Using an RDBMS," *Bulletin of the IEEE Technical Committee on Data Engineering*, vol. 22, no. 3, pp. 27-34, 1999.

[18] F. Tian, D. J. DeWitt, J. Chen, and C. Zhang, "The Design and Performance Evaluation of Alternative XML Storage Strategies," *ACM SIGMOD Record*, Vol. 31, Issue 1, pp. 5-10, 2002.

[19] I. Tatarinov, E. Viglas, K. Beyer, J. Shanmugasundaram, and E. Shekita, "Storing and Querying Ordered XML Using a Relational Database System", *SIGMOD Conference*, June 2002.

[20] M. Rys, D. D. Chamberlin, D. Florescu, "XML and relational database management systems: the inside story," *SIGMOD Conference*, June 2005.

[21] J. Funderburk, S. Malaika, and B. Reinwald, "XML programming with SQL/XML and XQuery," *IBM Systems Journal*, Vol. 41, No. 4, pp. 642-665, 2002.

[22] M. Krishnaprasad, Z. H. Liu, A. Manikutty, J. W. Warner, and V. Arora, "Towards an industrial strength SQL/XML Infrastructure," in *Proceedings of the 21st International Conference on Data Engineering*, 2005.

[23] P. Thibodeau, "Planet-Scale grid," *ComputerWorld*, October 10, 2005.

[24] J. Shanmugasundaram, E. Shekita, R. Barr, M. Carey, B. Lindsay, H. Pirahesh, and B. Reinwald, "Efficiently Publishing Relational Data as XML Documents," *The VLDB Journal*, vol. 10, nos. 2-3, pp.133-154, 2001.

[25] R. Krishnamurthy, R. Kaushik, and J. F. Naughton, "XML-to-SQL Query Translation Literature: The State of the Art and Open Problems," *XML Symposium (XSym)*, 2003.