Evaluation of Two XML Storage Approaches for Scientific Metadata

Scott Jensen, Devarshi Ghoshal, Beth Plale
Indiana University
scjensen@indiana.edu, dghoshal@indiana.edu, plale@indiana.edu

Abstract

Scientific data are increasingly described by metadata based on detailed XML schemata that capture both general and domain-specific concepts about the underlying data. Metadata captured using detailed XML schemata tailored to specific scientific domains increases the potential for data reuse by providing the ability to discover data products described by detailed concepts. Since such metadata is captured as XML, one alternative for managing scientific metadata is to store and query the metadata using a native XML database. Our research shows that a hybrid XML-Relational structure such as is used in the XMC Cat metadata catalog outperforms a native XML database for storing and querying scientific metadata; and significantly outperforms the native XML database under a scaled workload of concurrent inserts and queries.

1. Introduction

With the increasing deluge of scientific data being generated, particularly *in silico* through simulations and scientific workflows, increasingly scientists need detailed metadata to assist them in finding the scientific data that is relevant to their research and needed for their experiments. As scientific communities have started to describe their data using profiles and extensions of detailed metadata standards, a richer set of metadata is available to enable data discovery if the metadata can be efficiently queried.

Our research has shown that a hybrid XML / relational approach to storing and querying scientific metadata as XML using a relational database performs well and can be generalized to accommodate a range of scientific metadata schemata without requiring the relational database schema to be modified. This approach has been implemented as the XMC Cat metadata catalog and previously we have shown [9] that it performs well in comparison to other approaches to storing and querying XML using a relational database. However, since scientific metadata conforming to established metadata standards is increasingly expressed as XML, this paper compares the performance of the hybrid XML-relational storage solution used in the XMC Cat metadata catalog to a native XML database; Oracle's Berkeley DB XML [14]. We measure baseline performance of each system in executing the most common insert and query operations for a metadata catalog. We then compare their scalability in handling an increasing workload of concurrent inserts and queries based on metadata describing meteorological workflows generated by actual users of the Linked Environments for Atmospheric Discovery (LEAD) Science Gateway [3]. Baseline performance for the two approaches on inserts and simple queries is comparable; with the XML database performing slightly better on inserts and XMC Cat performing slightly better for simple queries. However, on complex queries, XMC Cat's performance is an order of magnitude better than the native XML database. The most significant performance difference between the two approaches arises when handling a workload of concurrent inserts and queries; with XMC Cat able to scale to a workload at least 8,000% of the concurrent workload that the Berkeley DB XML database can manage.

The remainder of the paper is organized as follows. In section 2 we discuss the primary operations of a metadata catalog that are used for this comparison. Section 3 discusses the data structures used in XMC Cat and Berkeley DB XML, and Section 4 provides details of the datasets and the experimental setup. Sections 5 and 6 present the baseline performance of query and insert operations respectively, and Section 7 compares the scalability of the two systems in handling a workload of concurrent inserts and queries. Sections 8 and 9 discuss related work and conclude.

2. Key Activities of a Metadata Catalog

In using a scientific metadata catalog, the most frequent operations are metadata insertion and responding to queries as scientists either browse the metadata describing the data holdings or use the metadata to search for data objects relevant to their research. In a scientific gateway, the new observations being cataloged are often generated through scientific workflows, and metadata regarding new digital objects are cataloged as a workflow generates files or uses existing files as inputs for a stage in the workflow. Metadata is also added incrementally to the metadata describing existing data objects as on-going

experiments generate metadata regarding events in the workflow.

In the LEAD Science Gateway, the most common metadata catalog operations are (1) inserting metadata describing a new data object, (2) adding metadata for an existing object, (3) querying for the full metadata of a specific object, and (4) searching for data based on their detailed metadata. The queries against the metadata catalog serve two different purposes: browsing and searching. When a scientist is browsing a metadata catalog, they are reviewing summary metadata and then clicking on a specific item to see the detailed metadata that describes that object. It is that action of clicking on a specific object that generates a query against the catalog based on that specific object's ID. In contrast, search queries are generated against the catalog when a scientist knows the characteristics that describe the data objects they are looking for, and those characteristics (which are captured as discovery metadata) are then used to find the data objects matching the scientist's needs.

Although both insert and query operations against the metadata catalog are frequent, the response time for queries is more important than the response time for inserts. The reason for this is that queries are executed interactively by a scientist as they are browsing or searching the metadata catalog to find the data products relevant to them or to review the results of an experiment. Insert operations in contrast are most often triggered by long-running workflows and the metadata catalog's execution time for the insert is trivial in comparison to the execution time of the workflow the metadata describes.

3. Data Structures in XMC Cat and Berkeley

For scientific metadata schemata such as FGDC [6], metadata can be captured using a domain schema for data objects at multiple levels of aggregation using a domain hierarchy. In LEAD, this hierarchy consists of projects, experiments, collections, and files, with projects being the broadest aggregation, but not every level is required in each path of the hierarchy within a user's workspace. For example, a project could be the direct parent of a collection or file with no intervening experiment. Meteorological workflows in LEAD are captured as experiments with some files used or generated within an experiment being grouped into collections (such as the input files), but other files are contained directly within an experiment.

3.1 Data Structures in XMC Cat

The XMC Cat metadata catalog can be implemented for different metadata schemata - tailored to the domain implementing the catalog, and the hierarchy used to manage the data objects can also be tailored to the domain. To provide this flexibility, the schema for the relational database is not coupled to the XML metadata schemata, but is instead generalized to manage shredded scientific metadata concepts. When the catalog is implemented, the schema is partitioned into independent concepts represented in the domain metadata schema, and as metadata is ingested into the database, it is parsed into two representations based on the identified concepts.

When metadata is added to the catalog, the first representation of the metadata is as character large objects (CLOBs) parsed out of the XML metadata for each concept. All of the necessary XML namespace declarations for a concept are pushed down into each concept - allowing the concept to essentially stand on its own, independent of the rest of the XML document. These CLOBs are used to quickly rebuild the XML metadata describing a digital object in response to queries - after XMC Cat has first identified the objects that match a user's search criteria. In addition to the CLOBs, a second representation of the metadata is stored which allows for querying the metadata to identify the data objects of interest to a user. In this representation of the metadata, each concept is "shredded" into its component leaf elements as described in [10] and stored in relational tables using a generic entity-attribute-value (EAV) approach. Each value is associated with its structural schema element based on separate tables containing the schema definitions for the concepts, any sub-concepts, and elements they contain. The shredded metadata is used only as an index to find the internal IDs of the objects that match a user's search criteria. Once the relevant data objects have been identified, the CLOBs are used to rebuild the XML. This approach allows XMC Cat to be configured for the XML metadata schemata used in different scientific domains without modification to the underlying relational database schema.

Since each scientific domain implementing a metadata catalog may have different perspectives on the data hierarchy applicable within their field, the XMC Cat metadata catalog takes a generalized approach to the hierarchical relations between data objects. The table that defines each object contains a column for the internal ID of each object's direct parent, but there is also a separate parent table that tracks the hierarchical relationships between each data object and all of its direct and indirect children. This allows queries to quickly construct a subtree of the user's workspace or perform context queries without requiring recursive self-joins on the tables containing the data object definitions.

3.2 Capturing scientific metadata using Berkeley DB XML containers

A scientific metadata catalog needs the flexibility to accommodate diverse data object hierarchies and perform context queries. In Berkeley DB XML, documents are stored in "containers", and an XML database can consist of multiple containers. Two alternatives for structuring Berkeley's containers are: (1) store all of the data objects within their respective top-level object (e.g., project in LEAD) as a single XML document, or (2) use a separate container to maintain the relationships between data objects. Since there is no limit in LEAD as to how many experiments or other child data objects are contained in the hierarchy for a single project (in LEAD, some users create all of their workflows as experiments within a single project), having a single XML object for each project becomes unwieldy and significantly impacts both update performance and scalability. For this reason, the implementation of metadata storage in Berkeley XML DB for purposes of this comparison uses separate containers for the metadata describing data objects and the relationships between data objects.

In Berkeley DB XML, two types of indexes can be created – node indexes and path indexes. The index type is specified as the "path type" when defining an index in Berkeley DB XML. A node index creates an index over elements or attributes in the XML document based on the name of the element or attribute. A path index, referred to as an "edge-type" index in Berkeley DB XML, indexes the path between an element and its parent element (the edge in the graph between an element node and its parent node). Path indexes are core to query languages for semi-structured data [19] and since they tend to be more selective, are preferred by the Berkeley DB XML query optimizer over node indexes when both can be used in a query plan [15].

4. Experimental Datasets

The comparison presented in this report utilizes metadata extracted (with permission) from the workspaces of active users of the LEAD science gateway. These metadata describe both the inputs and outputs of actual meteorological experiments. Baseline performance evaluation of both query and insert operations (i.e., without a concurrent workload), requires first populating the database with base data. This evaluation uses base data consisting of metadata describing experiments for 125 users based on the current number of users in LEAD. For this base data, the user accounts are populated with metadata from the workspaces of 15 active LEAD users and the metadata is then replicated to populate the remaining users.

4.1 Experimental environment

The focus of this comparison is on the storage and retrieval of the metadata as XML based on detailed community XML metadata schemata using XMC Cat and the Berkeley DB XML database. A web service front-end could be used with either backend approach, so we focus on the operations performed within the web service which are specific to cataloging metadata. Since scientific metadata is communicated as XML and the domain specific XML being inserted or returned in response to a query is passed as parameters in the metadata catalog's WSDL, the time required to connect to the web service or any network latencies in the web service receiving requests or transmitting query responses is the same under either approach.

To prevent minor fluctuations in network performance from influencing the comparison, both the baseline and scalability tests measure the insert performance on the server side from the point at which the XML being inserted is retrieved from the web service call. For queries, the tests measure the performance up to the point where the XML is reconstructed and ready to be returned in the web service response.

Both the XMC Cat web service and the Berkeley DB XML database are hosted on a Dell PowerEdge 6950 configured with quad dual-core AMD 2.4GHz Opteron processors, 16GB of memory, and running RHEL 4. The XMC Cat web service stack uses version 1.3 of Apache Axis2 on Tomcat version 5.5.28, and MySQL version 5.0.41. The native XML database used is Oracle's Berkeley DB XML (BDB XML) version 2.5 and the Berkeley DB XML Java API was used to perform insert operations and submit queries.

5. Metadata as a Search Tool

As discussed in Section 2, two of the key queries in a metadata catalog are 1) searching based on the unique ID that identifies a data object, and 2) more complex searches based on the detailed discovery metadata that describes each data object and the relationships between related objects.

To query based on the rich domain metadata captured in LEAD, we query based on both workflow notifications and keywords. In LEAD, each stage of the workflow generates a "computation duration" notification that is captured as metadata and contains the workflow's ID, a timestamp, the name of the calculation stage, and the

duration of the calculation in milliseconds. We query for experiments where the computation is for the module named "Terrain Preprocessor" and the computation took 130,550 milliseconds or longer to run. Since scientific communities describe their data using hierarchical classifications, this search also includes the relationship between the experiment and a configuration file used in that experiment. The configuration file is identified based on the keyword (Terrain-V5.2.7) and thesaurus (NAMELIST) which identifies it as a version 5.2.7 terrain configuration file. The full query is illustrated in Figure 1. We refer to this type of query as a "context" query since we are searching for a data object (an experiment in this case) that exists within a particular context (contains a specific type of configuration file).

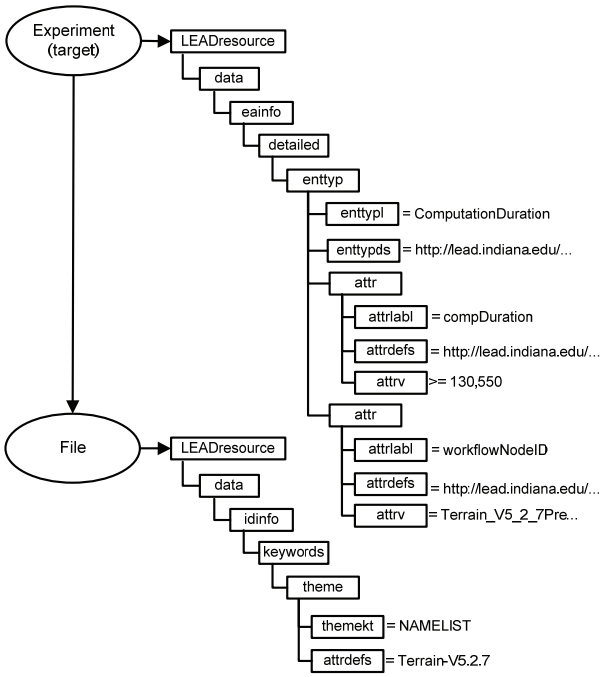

Figure 1: Context query for an experiment. The baseline query for an experiment based on the duration of the terrain preprocessor calculation and using a specific configuration file.

As a measure of baseline performance, we analyze the response time for both queries based on an object's unique ID and context queries based on the discovery metadata. Both types of queries are executed 25 times with the two types of queries interleaved in alternating sets of five, so there are five queries by ID followed by five context queries. The first query for each type is discarded and the results are calculated based on the remaining queries. Each query is executed for a different data object (query by ID) or a different user (context query).

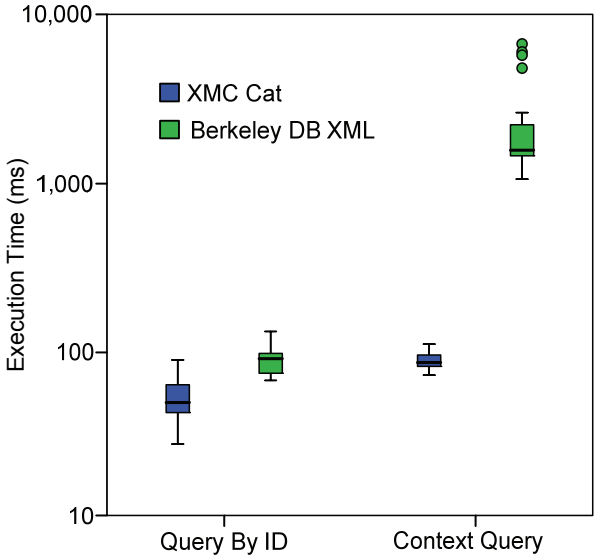

Figure 2: Baseline queries comparing XMC Cat and Berkeley DB XML.

The box-and-whisker diagram in Figure 2 compares the server-side execution time in milliseconds using a logarithmic scale for both XMC Cat and Berkeley DB XML for object ID queries and context queries. For the query based on a data object's ID, XMC Cat outperforms Berkeley DB XML (a median execution time of 55.2 ms versus 89.54 ms as indicated by the solid horizontal line within the shaded box for each query type). Although the Berkeley DB XML query took 79% longer (based on median performance), both queries executed quickly when retrieving documents based on an indexed ID. However, the more significant difference in performance is for the context query where the performance of Berkeley DB XML is more than an order of magnitude slower than that of XMC Cat. In addition, although XMC Cat's performance is very consistent across users (as exhibited by the compressed box plot with short whiskers), Berkeley DB XML exhibits significant outliers (the shaded circles above the box blot) where performance degrades to nearly two orders of magnitude in comparison to the slowest results from XMC Cat. An analysis of the queries that resulted in the outliers for Berkeley DB XML also indicates that switching from queries based on object IDs to context queries has an associated cost. Although all of the queries target different users and different objects (in the case of queries based on the object ID), the outliers all occur when switching to context queries from queries based on object IDs.

The benefit of node indexes in Berkeley DB XML is greatest when the node being queried is unique within the XML document. In this comparison, the node indexes are very beneficial for the queries based on an object's ID since the ID element is a globally unique identifier for that object. The query based on the object ID must also verify that the user has rights to the object being queried. For our purposes the XML documents being stored in Berkeley DB XML are metadata describing a digital object, but from the perspective of the database, these are just XML documents. For each XML document stored in Berkeley DB XML, additional metadata can be stored that describes the XML document. This metadata describes the XML document; not the digital object described by the XML. Since determining the ownership of an object is a frequent function in a metadata catalog, and accessing this metadata about the XML is fast in Berkeley DB XML if it is indexed, we store the distinguished name (DN) of the owner as indexed metadata of the document. Indexing the DN significantly improves query performance. The performance of the query based on an object's ID performs well in Berkeley DB XML since it is searching for an indexed unique field (the object ID) and also retrieving the indexed DN of the object's owner to verify the user has rights to the object.

For purposes of the context queries, we create path indexes for the elements used to capture the computation notifications (in addition to having node indices on the values). However, for scientific metadata schemata which are composed of independent concepts, the element names tend to be unique and specific to a concept. In testing with and without path indices, we found that creating a path index for the detailed element did not improve results over the node indices alone, but instead only increased the indexing workload when cataloging metadata.

In LEAD, each scientist has their own workspace containing their observations or experiment results. In this situation, as a scientist is browsing their workspace, query operations of the XMC Cat web service are executed based on object IDs as a user expands different branches of the tree hierarchy in their workspace and views the detailed metadata describing experiments and files. Although this use pattern would argue for each query targeting a unique data object as measured by the results previously depicted in Figure 2, we also analyze queries based on iterations over the same set of objects. This use pattern may occur more frequently in a public metadata catalog where different users would browse the metadata of popular datasets. Berkeley DB XML performs better at retrieving these repeated requests for cached objects than XMC Cat does.

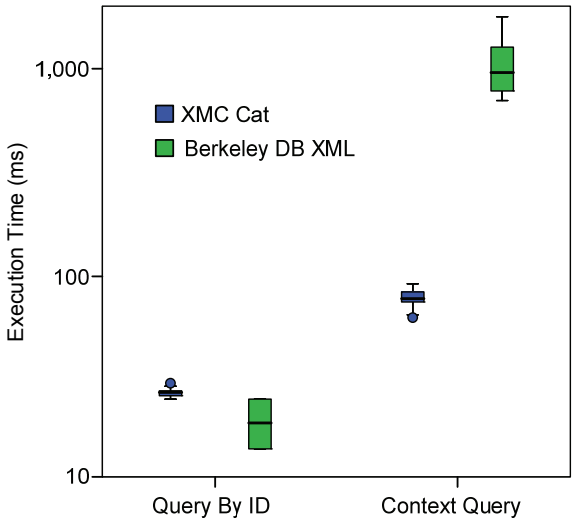

Figure 3: Queries based on cached results. Berkeley DB XML exhibits better performance than XMC Cat for simple queries when based on cached results, but is still an order of magnitude slower for the more complex context queries.

As shown in Figure 3, the median execution time of Berkeley DB XML on repeated requests for the same object is 14ms versus 33ms for XMC Cat. The variance for Berkeley DB XML when querying based on the object ID appears to be wide as illustrated in Figure 3 (the large box) because the average response time differs for experiments versus files, but the variance within each type is small. There are no whiskers in the box-and-whisker diagram for the Berkeley DB XML cached queries since the only variance in performance is due to retrieving metadata for experiments versus files (where experiments are described by richer metadata). For XMC Cat, the database is caching the relational queries, but not the reconstructed XML. Instead, repeated queries in XMC Cat are handled at the user interface level of the software stack in the Java Server Faces (JSF) backing bean for the workspace browser. As a user requests the details for a specific data object, the backing bean caches the retrieved metadata during that session. As long as repeated queries for the same data object are through the metadata catalog's web interface, XMC Cat avoids a database query by using the cached result (the same approach could also be used to cache results retrieved from a native XML database). The caching of the XML result in Berkeley DB XML is beneficial when multiple users query for the same public data object since these queries would not be in the same session and thus benefit only from caching at the database level of the software stack. For the more complex context queries, rerunning the same queries improves the overall query performance of Berkeley DB XML and eliminates the outliers, but query performance is still more than an order of magnitude slower than XMC Cat.

6. Cataloging Metadata

In addition to retrieving metadata for specific objects and searching for objects using the contextual discovery metadata, the other significant operation for a scientific metadata catalog is ingesting metadata describing data objects. In this analysis, we measure the performance for ingesting metadata describing an object not previously included in the catalog and also the addition of new metadata for an object that had previously been cataloged. The execution time measured in milliseconds for XMC Cat and Berkeley DB XML when performing both of these insert operations is shown using a logarithmic scale in Figure 4. The data objects described by minimal metadata represent experiments where only the core metadata is initially inserted in the catalog and then metadata containing additional concepts that describe the object are added incrementally to the metadata as the experiment executes. The data objects represented by a moderate amount of metadata are input files used in the experiment. The input files are described by core metadata plus keywords from a controlled vocabulary used in meteorology, CF-1.0 [5] and metadata describing the distribution of the data files. In contrast, experiments are described by a much richer set of metadata that includes the configuration settings for multiple stages of the experiment as well as metadata regarding the execution of the workflow. The objects represented by extensive metadata are experiments where all of the metadata describing the experiment is added as a single insert.

For each type of data object, 25 iterations of the insert operation are executed, and for the experiments where additional concepts are added incrementally, 10 additional metadata concepts are added for each data object cataloged (for a total of 250 additional concepts). When adding both the initial metadata describing a data object and the additional concepts, the first of the 25 iterations is discarded in calculating the performance results. Berkeley DB XML's average insert performance is almost twice as fast as XMC Cat for objects described by minimal to moderate metadata (30.04 ms faster for minimal metadata and 90.33 ms for files with moderate metadata). When inserting the more extensive metadata describing an experiment, XMC Cat performs slightly better on average (276.88 ms faster).

For the insert operations that add metadata for an existing data object (the objects described initially by only core metadata) XMC Cat performed better on average than Berkeley DB XML (46.92 ms versus 84.93 ms), but both approaches exhibited a wide distribution in execution time with a number of outliers. Both approaches took longer to add the first additional concept for a data object than to add subsequent concepts. For this baseline analysis, 10 concepts are added to each experiment, but for actual experiments in the LEAD Science Gateway, experiments frequently are described by over 200 concepts, so in domains where experiments are described by rich metadata, the cost of the initial insert would be significantly less of a factor in overall performance.

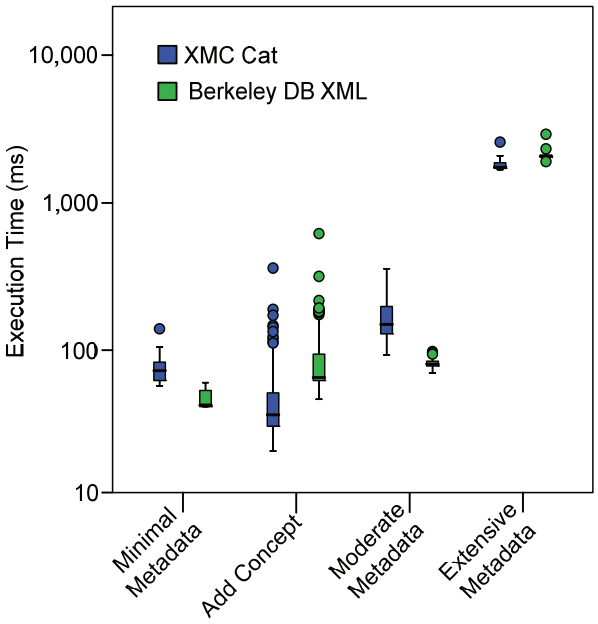

Figure 4: Baseline inserts for differing volumes of metadata.

Updates to existing XML documents in Berkeley DB XML are based on the XQuery Update Facility [2] and must specify a position in the document using an XQuery expression and one of the following keywords that specifies a location relative to that position: before, after, as first into, as last into, or into (which will also make it the last child) [15]. For purposes of this test, every insert is a `detailed` element from the LEAD Metadata Schema (LMS) which extends the FGDC schema to catalog additional domain-specific metadata. Figure 5 presents a snippet of the first few levels of the LMS which shows that the `detailed` element is an optional element within an optional `eainfo` element. The `eainfo` element can contain `detailed` and/or `overview` elements. Based on *a priori* knowledge regarding the metadata being inserted in this test, we know that each object's metadata contains a `geospatial` element and does not contain any `overview` child elements within its `eainfo` element. Since the minimal core metadata being initially inserted does not include an `eainfo` element, the baseline test only needs to check for the existence of an `eainfo`

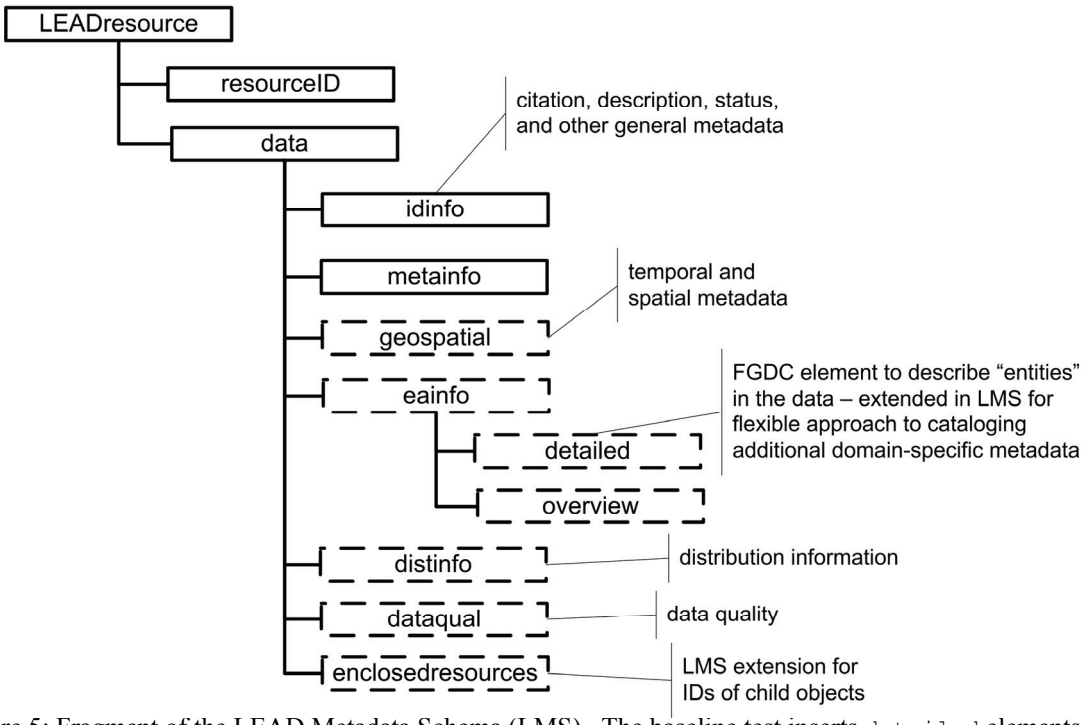

Figure 5: Fragment of the LEAD Metadata Schema (LMS). The baseline test inserts `detailed` elements within `eainfo` when cataloging metadata incrementally.

element before inserting a new `detailed` element. Checking for the existence of an `eainfo` element exhibits very consistent performance and on average takes 15ms. Since we can assume there is a `geospatial` element, the `eainfo` element is always inserted after the `geospatial` element. In a metadata catalog deployed using the LMS, no assumptions could be made regarding the existence of optional elements in a specific data object's metadata, so an additional query would be required to check for the existence of a `geospatial` element if the `eainfo` element did not already exist.

If a metadata catalog implemented on a native XML database uses XMC Cat's concept-based approach which allows incremental metadata capture, depending on the structure of the schema and the use of optional elements (which would be necessary to keep the burden of implementing from being excessive), adding metadata incrementally could require a significant number of queries to determine the position where a concept could be added. In the snippet of the LMS shown in Figure 5, if metadata regarding data quality is being added for a data object, inserting the `dataqual` element requires first determining if the metadata contains a `distinfo`, `eainfo`, or `geospatial` element. An additional check that was not implemented for Berkeley DB XML in the baseline tests is whether the concept being added results in XML that validates against the metadata schema. In XMC Cat these checks are implemented based on the static global ordering down to the concept level. In Berkeley DB XML, it would be necessary to first check for the existence of any mutually exclusive elements.

7. Scalability under a Concurrent Workload

Our comparison of XMC Cat and Berkeley DB XML is based on their performance in storing the metadata that describes scientific workflows and then making that metadata available to search for data products via a web service such as in the LEAD Science Gateway. To determine the scalability of each approach in handling the workload generated by a scientific gateway, we need to measure performance in handling insert and query operations simultaneously. To estimate a realistic initial base workload of concurrent insert and query operations, we use the projected LEAD data workload from a study by Plale [16]. That study projects data generation patterns for the LEAD Gateway, providing us an estimate of the data workload for the production LEAD environment over a 12-hour time frame. To estimate the concurrent query workload, we use the insert-to-query ratio of the Transaction Processing Council's TPC-E online transaction processing (OLTP) database benchmark [18] to determine the number of queries relative to the number of insert operations in the workload. Of the TPC database benchmarks, the TPC-E most closely resembles the workload of a scientific metadata catalog in that it models the workload generated

by customers of a brokerage firm as they simultaneously execute transactions (inserts) and inquire (query) regarding their accounts. The TPC-E workload can be scaled in multiples of 1,000 customers to model the workload of different sized businesses. Similar to the TPC-E benchmark, the scalability that matters for a scientific metadata catalog is its performance as the number of researchers using a scientific gateway increases – generating an increasing workload of concurrent inserts and queries. To test scalability, we start with an initial concurrent workload based on [16] and the TPC-E benchmark. We then measure performance as the initial workload is doubled and then incremented in multiples of 2-times the base workload. Before starting the scalability tests, the database is first populated with the same base data for 125 users that is used in the query and insert baseline tests.

The production data workload for the LEAD science gateway as projected in [16] is based on 125 active users creating a data workload consisting of the following four categories of data generation: educational experiments (50%), canonical experiments (10%), ensemble experiments (1%), and data imports (39%). Each class of experiments generates a different workload, with educational experiments estimated to generate an average of 16 files in total, whereas the larger canonical experiments represent the WRF forecasting model [13] which is the dominant workflow used in the LEAD Gateway and is estimated to generate 9 files in each of its 8 stages (where the terrain computation is an example of one of the stages). Although percentage-wise the ensemble workflow appears to be a minor component, it represents an ensemble of 100 canonical experiments and is a significant factor in the total data workload generated. The data import is not a workflow, but instead represents a user importing 15 files from a public data catalog.

The scalability test builds on the minimal, moderate, and extensive classifications used for the baseline insert analysis. Since metadata is generated incrementally as a workflow executes, the metadata initially cataloged for the canonical and ensemble experiments is based on the minimal core metadata and then concepts based on model configuration parameters and critical workflow notifications are added incrementally based on the running time of the experiments. To test the scalability performance when inserting large metadata documents, the full metadata describing an experiment, which represents the "extensive" insert in the baseline test, is used for the educational experiment since it executes as a single stage. The metadata added for files in each of the four above listed workflow classifications is based on the "moderate" level of metadata from the baseline which represents the metadata describing one of the data input files for a WRF experiment in LEAD. The query workload is based on the insert-to-query ratio of the TPC-E benchmark, with 25% of the queries classified as context queries and the remaining 75% of the queries retrieving files and experiments based on their global IDs. The context query uses the baseline criteria that searches for experiments based on the computation duration notification and certain configuration files, but both types of queries search over a broader range of users than was used for the baseline tests.

The 12-hour estimated LEAD data workload from [16] is scaled to a 30-minute test window, which results in the insert and query operation workload shown in Table 1 being executed as the base scalability workload:

Canonical experiments inserted *(core metadata inserted)* Concepts added incrementally	28 167
Files inserted *(moderate level of metadata)*	1,850
Education experiments added *(extensive metadata)*	17
Queries based on global ID	4,707
Context queries	1,570

Table 1: Base scalability workload of concurrent insert and query operations executed in a 30-minute window. The scalability test is based on executing multiples of this workload.

The first two minutes of each run are excluded for initializing the server and loading data structures that are populated when the web service is first started. The workload is then scaled in multiples of this base workload.

7.1 XMC Cat scalability under an increased workload

To test the scalability of XMC Cat, the workload is initially doubled so that the number of concurrent inserts and queries of each type during the 30-minite test window is twice the volume shown in table 1. The workload is then scaled up in multiples of 2 times the base workload until the server is saturated and cannot handle the workload. The tomcat server hosting XMC Cat uses a time-out window of 30 seconds. XMC Cat scales to over 8 times the projected workload before encountering fatal errors where an insert or query operation does not complete successfully. In XMC Cat, metadata is captured both as XML CLOBs for each concept and also shredded into queryable metadata. As long as the CLOBs are successfully stored in the database, the metadata for an

object can be reconstructed, so any errors in shredding and storing the queryable discovery metadata are tracked but considered non-fatal. As the workload is scaled, a minimal number of non-fatal insert errors occur staring at two times the base workload, with a maximum of 9 objects encountering non-fatal errors in the 30-minute window. The scalability performance of XMC Cat based on multiples of up to 8 times the base workload is shown in Figure 6.

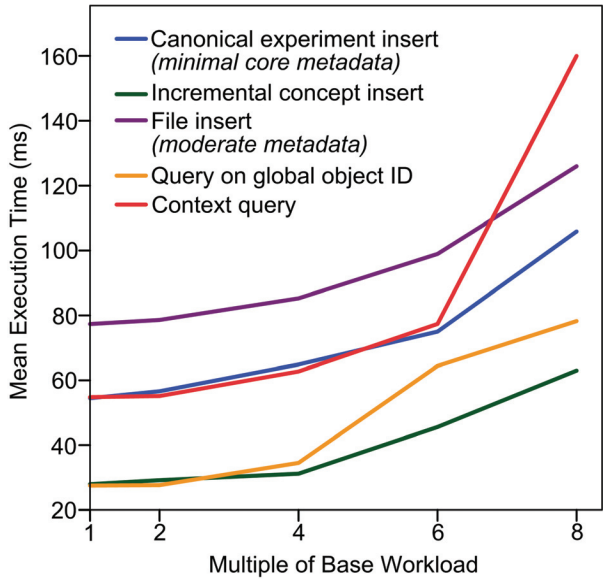

Figure 6: XMC Cat scalability based on multiples of the base workload. With multiple clients executing concurrent inserts and queries, XMC Cat scales up through 8 times the base workload, and performance is particularly consistent for both inserts and queries through 4 times the base

From the initial base workload through 4 times that base workload, XMC Cat scales very well in handling both queries and metadata inserts concurrently, with the heavier workload resulting in only minor increases in the average execution time. The exception to this is when the entire metadata for an experiment is ingested as a single large XML document instead of inserting metadata concepts incrementally. Although the rate of increase in average execution time for both queries and inserts starts to increase more rapidly beyond 4 times the base workload, performance does not degrade significantly until saturation is reached after 8 times the base workload (with the exception of the context queries which degrade more quickly after 6 times the base workload). Our analysis of the performance when inserting large metadata documents (the educational experiments with extensive metadata in the scalability workload) reveals that the degradation in performance as well as the variance in execution time is driven mainly by the cost of validating the XML web service request document (including the full metadata insert). For smaller XML documents such as the metadata inserted for files, the validation of the insert request is not a significant component of the overall execution time. Figure 7 shows the average execution time (with a 95% confidence interval) when calling the insert operation with a large metadata document. When the cost of validating the request is eliminated, the performance is more consistent and scales significantly better.

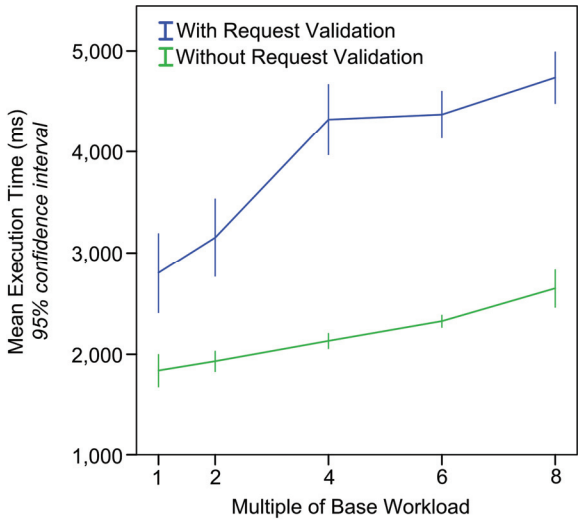

Figure 7: XMC Cat scalability under load of cataloging objects with extensive metadata. When all of the metadata for an experiment is cataloged as a single insert, validation of the insert request is a significant factor in degrading scalability and causes wide variances in execution time.

7.2 Berkeley DB XML scalability under an increased workload

To test the scalability of Berkeley DB XML when cataloging scientific metadata, the same workload based on the projected LEAD data workload is applied as was used to test XMC Cat. However, even at the base workload, the scalability test does not complete successfully when executed on Berkeley and instead generates database errors resulting in the test aborting before the 30-minute test window has elapsed. To determine the scalability of Berkeley DB XML, instead of scaling the workload up using multiples of the base workload, we scale the workload *down* - executing only a fraction of the base workload. At 15% of the base workload, the scalability test completes without aborting, but 3 of the metadata insert operations fail. At 10% of the base workload all of the inserts and queries complete

successfully, but a few of both the inserts and queries show significantly degraded performance near the end of the test. Figure 8 shows the execution time in milliseconds for each of the insert and query operations when the scalability test is executed at 10% of the base workload. The outliers are the inserts and queries with poor performance near the end of the test. At 10% of the base scalability workload, only 2 extensive metadata inserts are executed, so the mean is shown in Figure 8 instead of a box plot.

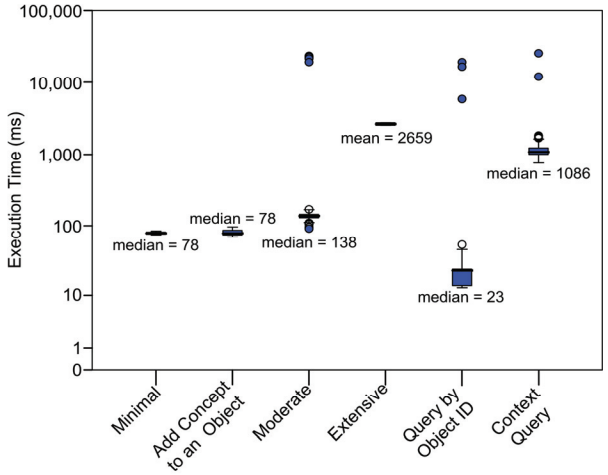

Figure 8: Berkeley DB XML scalability performance at 10% of the base workload. The base workload does not execute successfully on Berkeley DB XML, so the workload is scaled down – the maximum workload that can execute without insert or query failures is 10% of the base scalability workload.

The scalability of Berkeley DB XML is significantly degraded by concurrent inserts and queries. Figure 9 shows the performance of Berkeley at the base workload with insert and query operations run independently as separate tests instead of being executed concurrently. Without concurrent inserts, query performance improves significantly even though the volume of queries increases 10-fold in comparison to the results shown in Figure 8 for 10% of the base workload executed concurrent with inserts. Although the scalability of Berkeley DB XML improves considerably when there are not concurrent insert and query operations, neither the insert nor query workload can be scaled to even 2 times the base workload when run independently.

7.3 Scalability comparison

Berkeley DB XML's scalability does not compare favorably with XMC Cat for managing scientific

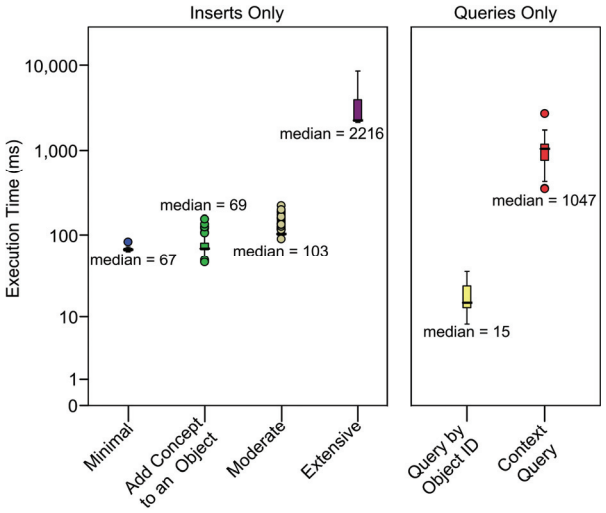

Figure 9: Berkeley DB XML scalability without concurrent inserts and queries. The base scalability workload can be executed on Berkeley DB XML if inserts and queries are not run concurrently but are instead executed as separate tests.

metadata. Even at only 10% of the base scalability workload, the median execution time on Berkeley DB XML for both queries and inserts is higher than XMC Cat at the base workload - even though XMC Cat is handling inserts and queries for an order of magnitude more metadata. The exception is the queries based on the global ID of an object, where the median time for Berkeley DB XML at 10% of the base workload is 3ms faster than XMC Cat at 100% of the base workload. However, the insert and query operations for Berkeley DB XML omit the following steps performed by XMC Cat (which would be required of Berkeley DB XML if implemented in an actual science gateway):

- Validating the operation's request.
- Verifying the user's authority to execute the request.
- For inserts of additional concepts, checking that the result will still be schema-valid.
- Additional checks needed to determine the insert position for metadata concepts added to existing metadata (without *a priori* knowledge of the existing metadata content).
- Making query results available to stream back to the user (in XMC Cat results can be returned directly in the response or made available at a URL and streamed back for better performance). The Berkeley DB XML tests assemble the result but perform no further handling of the XML.

As shown in Table 2, the median performance for Berkeley DB XML at only 10% of the base workload does not compare favorably with XMC Cat even as the

	Inserting Metadata				Queries	
	Minimal *(core)*	Moderate *(file)*	Extensive *(experiment)*	Additional Concept	Query on ID	Context Query
Berkeley 10%	87	138	2,659	78	23	1,086
XMC Cat - Percentage of Base Workload						
100%	52	74	2,316	26	27	63
200%	53	76	2,954	27	27	63
400%	60	80	4,803	29	31	69
600%	67	88	4,628	32	54	72
800%	69	89	4,719	36	34	145

Table 2: Comparison of median execution time (ms) for concurrent insert and query operations in Berkeley at 10% and XMC Cat for 100% to 800% of the base workload. Execution time for XMC Cat at 8 times the base workload compares favorably to Berkeley DB XML even though it is handling 80 times the workload.

workload on XMC Cat is scaled up based on multiples of the base workload.

Comparing the median execution time (in milliseconds) of Berkeley DB XML at 10% of the base scalability workload to XMC Cat as the workload is scaled from 100% to 800% of the base workload, XMC Cat outperforms Berkeley DB XML even at 8,000% of workload that Berkeley DB XML can manage. This holds true for each operation except the query based on an object's global ID (which still only takes 34ms to execute on XMC Cat at 8X the base workload). Although inserting the full metadata for an experiment (the extensive insert in Table 2), takes XMC Cat longer to execute than Berkeley DB XML when the workload exceeds the base workload, the Berkeley DB XML results do not reflect the cost of validating the insert. Eliminating that validation from the results for hybrid as reflected in Figure 7, the median execution time for the extensive metadata insert at 800% of the base workload is 2,477ms, which is lower than the 2,659ms required for Berkeley DB XML at 10% of the base workload.

8. Related Work

The XMC Cat metadata catalog uses an EAV approach to store the leaf elements of the metadata describing each digital object and uses this shredded metadata as an index to find data objects matching a user's search criteria. This approach to storing the metadata elements was inspired in part by the earlier MCS metadata catalog [17] which used an EAV approach to store metadata as user-defined attributes of digital objects. However, As noted in [17], when MCS user-defined attributes were used to represent metadata communicated as XML in the context of the Earth Systems Grid (ESG) project [4], the conversion of XML to and from the name/value pairs used to record user-defined attributes in MCS was a performance bottleneck. In XMC Cat a hybrid XML/relational approach is used which enables quickly reconstructing the XML in response to queries while also allowing for searching over more complex concepts than is possible in MCS.

An alternate approach to cataloging XML metadata in a relational database is used in Metacat [11] which is focused on cataloging ecological data. As in XMC Cat, metadata is ingested as XML, but instead of query results returning metadata as XML, query results can contain the name/value pairs parsed out of the leaf elements of the XML ingested. When an XML document is added to Metacat, all of the possible paths and path fragments to each node in the document are stored in a relational table that contains the ID of each node, the ID of the node's parent, and the contents of the node (if it is a leaf node in the document). This approach is similar to the edge table approach presented by Florescu and Kossmann for storing schema-less XML documents [8]. However, to avoid the cost of self-joins of the edge table in traversing paths to respond to queries, Metacat stores not only each edge, but each path fragment in the database [1]. When a document is added, an index is built that contains a record for each possible path or fragment of a path through the document. This index improves the performance of the queries by avoiding recursive joins, but it makes ingesting new documents an expensive process. Also, since updates are performed by replacing the entire document, updates to the metadata require rebuilding the entire index for a document even for minor revisions to a stored document.

The Fedora Commons data archive [7], which has its roots in the library sciences, and originally supported only the METS metadata standard [12], captures a limited set of discovery metadata using FOXML, but can also capture arbitrary XML as metadata. However, Fedora takes a different approach from scientific metadata catalogs in that this additional XML metadata is not

available as discovery metadata that can be used to find data objects, but is instead stored as a file (referred to as a stream) that can only be accessed once the desired data object is identified.

9. Conclusion

Although scientific metadata is communicated using detailed XML schemata, for the operations most commonly performed by a scientific metadata catalog, the XML/relational approach used in XMC Cat outperforms the Berkeley DB XML database. XMC Cat can handle diverse metadata schemata and is designed to manage the scientific metadata used to describe and discover data objects. The purpose of the XML stored in XMC Cat is to serve as a search index, whereas a generalized XML database provides the ability to manipulate the XML and generate new XML documents.

A general XML database provides a broad range of functionality for manipulating XML documents, but although the XML metadata documents cataloged in this comparison are not extremely large, Berkeley DB XML scaled poorly at handling concurrent insert and query operations for more than $1/10^{th}$ of the projected workload for a production science gateway.

10. References

[1] C. Berkley, M. Jones, J. Bojilova, and D. Higgins, "Metacat: A Schema-Independent XML Database System," in *Proceedings of the 13th International Conference on Scientific and Statistical Database Management*, Fairfax, Virginia, July 2001.

[2] D. Chamberlin, M. Dyck, D. Florescu, J. Melton, J. Robie, and J. Simeon, "*XQuery Update Facility 1.0*", June 2009. Last accessed 2010-06-07 at: http://www.w3.org/TR/xquery-update-10/

[3] K. K. Droegemeier, D. Gannon, D. Reed, B. Plale, J. Alameda, T. Baltzer, K. Brewster, R. Clark, B. Domenico, S. Graves, E. Joseph, D. Murray, R. Ramachandran, M. Ramamurthy, L. Ramakrishnan, J. A. Rushing, D. Weber, R. Wilhelmson, A. Wilson, M. Xue, and S. Yalda, "Service-Oriented Environments for Dynamically Interacting with Mesoscale Weather," *Computing in Science and Engineering*, vol. 7, no. 6, pp. 12-29, 2005.

[4] Earth Systems Grid (ESC). Available at: http://www.earthsystemgrid.org

[5] B. Eaton, J. Gregory, B. Drach, K. Taylor, and S. Hankin, "*NetCDF Climate and Forecast (CF) Metadata Conventions*," Version 1.0, 2003.

[6] Federal Geographic Data Committee, Washington, D.C., "Content Standard for Digital Geospatial Metadata Workbook," Version 2.0, 2000.

[7] Fedora Commons. Available at: http://www.fedora-commons.org

[8] D. Florescu and D. Kossmann, "*Storing and Querying XML Data Using an RDBMS*," IEEE Data Engineering Bulletin vol. 22, no. 3, pp. 27-34, 1999.

[9] S. Jensen and B. Plale, "Using Characteristics of Computational Science Schemas for Workflow Metadata Management," in *Proceedings of the 2008 IEEE Congress on Services - Part I*, July 2008, pp. 445-452.

[10] S. Jensen, B. Plale, S. L. Pallickara, and Y. Sun, "A Hybrid XML-Relational Grid Metadata Catalog," in *Workshop on Web Services-based Grid Applications (WGSA'06) in Association with the 2006 International Conference Workshops on Parallel Processing (ICPP-06)*, August 2006.

[11] Metacat data and metadata catalog. Available at: http://knb.ecoinformatics.org/software/metacat/

[12] Metadata Encoding and Transmission Standard: Primer and Reference Manual, Version 1.6, September 2007, Last accessed 2009-05-28 from: http://www.loc.gov/standards/met

[13] J. Michalakes, J. Dudhia, D. Gill, J. Klemp, and W. Skamarock, "Design of a Next-Generation Regional Weather Research and Forecast Model," in *Towards Teracomputing*, W. Zwieflhofer and N. Kreitz, (eds.) River Edge, New Jersey: World Scientific, 1998, pp. 117-124.

[14] Oracle Berkeley DB XML. Available at: http://www.oracle.com/database/berkeley-db/xml/

[15] Oracle. "*Getting Started With Berkeley XML for Java*," Version 2.5, August 2009. Last accessed 2010-06-07 at: http://www.oracle.com/technology/documentation/berkeley-db/xml/intro_xml/BerkeleyDBXML-Intro.pdf

[16] B. Plale, "Workload characterization and analysis of storage and bandwidth needs of LEAD workspace," LEAD TR 001, Linked Environments for Atmospheric Discovery (LEAD), Version 3.0, 2007.

[17] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Manohar, S. Patil, and L. Pearlman, "A Metadata Catalog Service for Data Intensive Applications," in *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, Phoenix, Arizona, November 2003.

[18] Transaction Processing Performance Council: TPC Benchmark E Standard Specification Version 1.2.0 (2007), "Version 1.2.0, 2007.

[19] Y. Wu, Path Query. Encyclopedia of Database Systems. 2008.