Grid-Based Metadata Services

Ewa Deelman[1], Gurmeet Singh[1], Malcolm P. Atkinson[2], Ann Chervenak[1], Neil P Chue Hong[3], Carl Kesselman[1], Sonal Patil[1], Laura Pearlman[1], Mei-Hui Su[1]

[1]Information Sciences Institute, University of Southern California
[2]Department of Computing Science, University of Glasgow & Division of Informatics, University of Edinburgh
[3]Edinburgh Parallel Computing Centre, UK
deelman@isi.edu, gurmeet@isi.edu, mpa@nesc.ac.uk, annc@isi.edu, n.chuehong@epcc.ed.ac.uk,
{carl, sonal, laura, mei}@isi.edu

Abstract

Data sets being managed in Grid environments today are growing at a rapid rate, expected to reach 100s of Petabytes in the near future. Managing such large data sets poses challenges for efficient data access, data publication and data discovery. In this paper we focus on the data publication and discovery process through the use of descriptive metadata. We describe the requirements for metadata services in the context of Grid and the available Grid services. We present a data model that can capture the complexity of the data publication and discovery process. Based on that model we identify a set of interfaces and operations that need to be provided to support metadata management. We present a particular implementation of a Grid metadata service, basing it on existing Grid services technologies. Finally we examine alternative implementations of that service.

1. Introduction

Today, advances in science are made possible largely through the collaborative efforts of many researchers in a particular domain. We see collaborations of hundreds of scientists in areas such as gravitational-wave physics [1], high-energy physics [2], astronomy [3] and many others coming together and sharing a variety of resources within a collaboration in pursuit of common goals. These resources are distributed and can encompass people, scientific instruments, compute and network resources, applications, and data. Although the scale of all resources pooled within collaborations is growing, there is a particular explosion in the amount of data that is made available. It is common to see datasets on the order of terabytes today, with petabyte-scale sets are coming online. Grid technologies [4] enable efficient resource sharing in collaborative distributed environments. In this paper we focus on the area of data management on the Grid, with a particular emphasis on metadata management issues and data discovery.

One of the challenges of these shared environments is to identify and locate the subset of data objects that is of interest to a particular data analysis activity. The standard solution to this problem is to describe the characteristics of each data object with one or more attributes, or "metadata," and to use this metadata as the means of identifying relevant data objects. Metadata allows collaborations to publish data with enough information for scientists to be able to identify the desired data products. Metadata can encompass a variety of information. Some metadata is application independent, such as the creation time, author, etc. described in Dublin Core [5], while other metadata is application dependent and may include attributes such as duration of an experiment, temperature, etc. Metadata adds value to scientific data. Without metadata, the researcher is unable to evaluate the quality of the data. For example, it is impossible to conduct a correct analysis of a data set without knowing how the data was cleaned, calibrated, what parameters were used in the process, etc.

The boundary between data and metadata is to some extent arbitrary and may vary during a data object's lifetime. For example, for some users a table of astronomic objects derived from images is primary data, while for others it is metadata that indexes the primary or calibrated data. A second example is the extensive use of human annotation in many curated biomedical databases. These annotations may be considered metadata or they may be considered primary data, with additional metadata required to describe the annotations and to gauge their quality. The ambivalence between data and metadata is generally resolved within a particular context. For the remainder of the paper, we refer to the data currently being used to enable the interpretation of other data as "metadata".

Traditionally, metadata was recorded in laboratory notebooks. More recently scientists have devised data formats to encode the metadata in the data files themselves (as in the case of the FITS-formatted [6] files used to contain astronomy images). However, these mechanisms do not scale to large collaborative environments that include many databases or files. Grid technologies [4] are being increasingly used to provide data management infrastructure to access distributed resources within a collaboration. However, the issue of data discovery still remains. For ease of use and scalability, data discovery needs to be based on metadata attributes or annotations.

This paper addresses the question of what metadata services are needed in Grid environments to facilitate data publishing, discovery and access for large-scale data sets. We argue that although standardized database services can be used, specialized metadata services can greatly simplify

Metadata management. The contributions of this paper are:

- A description of metadata services requirements in Grids.
- A placement of metadata services in the context of service rich environments such as the Grid.
- A description of the data model supported by the service and the interfaces it exposes.
- An evaluation of the alternative implementations of the service: Metadata Catalog Service (MCS).

2. Requirements for Metadata Management on the Grid

The astronomy community provides a good example of how metadata services are used in collaborative environments. Initially when the data, in the form of images, is collected by an instrument, such as a telescope, it is pre-processed, calibrated and stored in an archive. Metadata about the images describing the location in the sky, the calibration parameters, etc., are stored as well. Additional processing may occur to extract interesting features of particular regions of the sky or to produce images focusing on particular celestial objects. The information about the processing is captured in metadata attributes and stored as well. Once the metadata and data are prepared in this fashion, they are released to the group of scientists within the collaboration. Researchers can then pose queries on the metadata to discover data relevant for them. Based on the results of the searches, scientists may want to organize the metadata in a way that is most appropriate for their research. During this phase of the data publication process, the data may be further annotated by the collaborators. After a certain period of time, usually on the order of two years, the data and the metadata are released to the general public. At that time, users outside of the initial collaboration can search the metadata based on attributes that are important to them. There are several requirements for managing metadata on the Grid. The requirements can be broadly grouped into four categories:

1. The need to store and share the metadata.
2. The need to organize the metadata in a logical fashion for ease of publication and discovery.
3. The need to customize the view of the data by individuals.
4. The need to support metadata about large-scale data sets.

We can refine each of these requirements further. Sharing metadata necessitates having well defined interfaces for storing and querying metadata attributes and for adding new attributes. In general, metadata is domain-specific. However, some metadata crosses many domains, such as creator and creation time. Metadata services thus need to be able to support generic as well as domain-dependent attributes. In the scenario above, the metadata evolved over time; thus metadata services need to be able to evolve as well, providing a flexible way to add and delete metadata attributes. Queries need to be able to support the discovery of the metadata attributes of the objects stored in the catalog as well as the discovery of attributes of a particular object and the discovery of objects with particular attributes.

It is also useful to organize metadata in some logical fashion. For example, all data relating a particular region of the sky may be placed within one group or *collection*. Attributes may also be associated with such groupings for ease of discovery. Aggregation allows for the grouping of objects that are related to each other. This may facilitate data discovery, for example, by allowing a user first to find all collections with particular characteristics and then to refine the search within these collections. If the search is performed on a flat object space without collections, then queries may return objects that have no particular relation to one another. Search performance may also improve with collections, because in general, there will be fewer collections than individual data items, making queries for collections with particular attributes more efficient than queries across all data items. The grouping may be hierarchical, representing the aggregation of collections. In the case of data publication by the collaboration, collections are usually organized by the publishing body. These collections are typically dynamic, intersecting and associated with particular scientific interpretations.

As part of its data organization, a collaboration may impose policies for publication and access for collections, collection groups, or for individual items. A Metadata Service must implement the required policies for authentication, authorization and auditing. Authentication and authorization allow control over who is allowed to add, modify, query and delete mappings in the Metadata Service. Auditing information maintained by the Metadata Service may include information about creators and creation times of metadata mappings as well as a log of all accesses to a particular metadata mapping, including the identity of the user and the action performed. Different types of users may be given different access privileges for the metadata.

Although collections allow publishers to organize the metadata, they do not allow customization by individuals who are part of the collaboration. In general, scientists within a collaboration may have different research goals and may want to organize the data in a way that is most appropriate for them. These individual-based views of the metadata should not affect the structure or authorization policies imposed by the publisher. Rather, they should be layered on top of the existing collections.

Data sets and their metadata are quite large today and ever growing in size. Often, metadata is stored independently of the data itself. Metadata services need to provide data handles that can be resolved by other services that perform the data access. Metadata services must provide the ability to store information about millions of data objects and provide good performance. They should

provide short latencies on query and update operations and relatively high query and update rates. To support reliable access to metadata, the services may need to be replicated.

3. Role of Grid Services

Grid services are used in many application domains today to deliver computing power and data management capabilities needed by large-scale science. Grid services extend standard web services by providing support for associating state with services, managing the lifetime of service instances, and standard mechanisms for subscription and notification of state changes. Grid service interfaces are being standardized as part of an overall Open Grid Services Architecture (OGSA)[7] through the Global Grid Forum[8]. Large scale testbeds such as the Teragrid [9] and iVDGL's Grid3 [10] are deploying Grid services that allow authentication [11], remote job scheduling [12], data access [13], data replication [14] and others. These are basic services available as part of the Globus Toolkit 3 [15], the de-facto standard for Grid services.

One particularly relevant Grid service is the OGSA Database Access and Integration (DAI) Service being developed by a UK consortium of Edinburgh, Manchester and Newcastle Universities, IBM Hursley and Oracle UK. Service interfaces are being standardized through the DAIS Working Group of the Global Grid Forum [16]. The DAI service provides a common Grid service access interface to a variety of data resources ranging from relational to XML databases and eventually to structured files. The DAI service is intended to provide a basis for higher-level services, for example, to provide federation across heterogeneous databases. The service has three main components: a service registry for discovery of service instances, a data factory service for representing a data resource and a data service for accessing a data resource, such as a relational database. The DAI service uses an extensible activity framework. Activities can be extended by other developers to provide additional functionality.

There are several reasons for providing grid service-based metadata services:

1. Integration with other Grid services: Resource access in Grids uses Security Infrastructure (GSI) [11] authentication based on PKI. To integrate databases with other services on the Grid, GSI authentication needs to be performed by the database service. DAI supports this type of authentication and maps authenticated users to database roles. Integration with other services is also enabled by DAI's XML-based communications.
2. Service discovery: In a distributed environment, there may be several metadata services. Discovering the appropriate service is necessary. Grid services and DAI support the use of service registries and the discovery of services based on service-specific attributes.
3. Federation of multiple databases: As mentioned above, there may be several relevant metadata services. It is important to be able to query across them in search of desired data. Grid services and DAI provide support for service data publication and notification of changes in the values of the service data, thus providing the basic mechanism for federating multiple services.

In this work we developed a Metadata Catalog Service (MCS) that builds on the DAI service. Below, we discuss the data management model supported by MCS.

4. Data Management Model

In the Section 2, we outlined a scenario for the use of metadata services for data publication and discovery. It is clear that metadata attributes could be represented in some database technology, such as relational or XML, and that the discovery of data objects be mapped into queries. We believe, however, that in many instances, it is desirable to have a more specialized metadata management service. We argue that in Grids, we need to have dedicated Metadata Services because of several concerns, such as usability and ease of schema discovery. Databases provide a very general and flexible infrastructure for data management. However, one has to be familiar with query languages such as SQL or XQuery to be able to efficiently interact with a DBMS. Many domain scientists in physics, biology, etc. simply are not comfortable with query languages. Users and applications also need to know the internal database structure to be able to pose appropriate queries. Our schema and API (described below) provide an easy way to discover attributes and to interact with the system.

Given the benefits of providing an extensible metadata model, we present a model that supports a variety of interactions, allowing users to describe attributes of data items and organize them in ways that are needed by a collaboration and by individual users. We also describe the API that supports the data model and is used to manage objects within the Metadata Service.

The model also supports a flexible set of attributes. We define the basic object within a metadata service as a data item. This data item may for example represent an individual image. Figure 1 shows a UML diagram of the objects supported within MCS. The following sections give more details about the role of each of the entities.

Users

In general we can distinguish between three types of users: the collaboration, the members of the collaboration and the general public. The collaboration is in charge of publishing the data sets. Individuals or groups within a collaboration may provide additional attributes and annotation and may structure the metadata in a personalized way. Finally, members of the general public (or community at large) may query the metadata services.

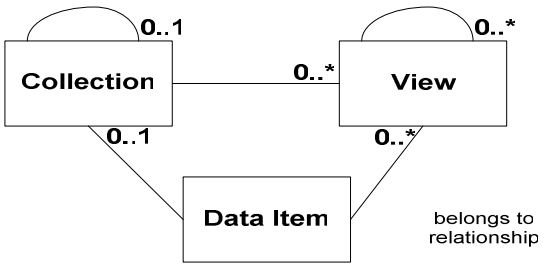

Figure 1: A data model for Metadata Services. The depicted relationship refers to the "belongs to" relationship.

Structuring Data Items and Imposing Authorization Policies

It is often convenient to be able to refer to a set of data items with a single name. This can play an important role in improving the scalability of a metadata service. For example, authorization can be attached to that single object, without having to impose it on every individual item. Name sets can also be useful in the data publication process. They allow the data publisher to impose a logical structure on the data items being published. For example, a collaboration may want to assign a single name to all the data collected during a particular run of an instrument. For these reasons, our metadata model includes a concept of *collections*. Collections allow a name and attributes to be associated with an arbitrary set of data objects.

Collections can aggregate a set of data items or other collections. The collaboration may also impose authorization on data items and on data collections. This allows for defining authorization in a scalable way. To assure consistent authorization, a particular data item or data collection may belong to only one parent collection. If a data object could belong to multiple collections, then determining the authorization for the object would involve examining a set of possibly contradictory policies.

In our model, we view collections as being defined by the collaboration. However, individual members of the collaboration may want to organize and name the published metadata in a customized way. Individuals may also want to associate additional attributes with the named entities. The organization designed by individuals should not affect the organization and authorization imposed by the collaboration as a whole. To support this functionality, we introduce the notion of a *view*. A view allows data items and collections to be organized by members of the collaboration into groups that are relevant to them. Views do not have any effect on the way that the metadata is published to the community or how access to the data is authorized. Views may be described by attributes, be annotated and made part of other views. Because a view imposes no authorization, data items and collections may belong to several views. Members of the general public would typically be able to query metadata services but not to create views. Composition of both collections and views is acyclic.

Flexible Schema

There is no common set of attributes that can describe data in a variety of domains. Usually, a collaboration agrees on a set of terms that describes their particular data set. However, metadata services need to span domains and collaborations and thus need to support a dynamic attribute set. Even within a collaboration, some flexibility may need to be supported. For example, members of the collaboration may come up with additional ways of describing the data, providing annotations and other attributes that are necessary for data interpretation. Flexibility in the attribute set is needed for all data objects managed by the metadata catalog: data items, collections and views.

Metadata attributes can be divided among a set of core attributes, such as those described in Dublin Core [5] and additional domain-specific attributes that can vary depending on the underlying application domain. There are also attributes that are specific to the Grid environment, where data may be replicated. As we already mentioned metadata discovery is the process of mapping a set of attributes to one or more identifiers that locate the data objects that possess specified attributes. This location specification, which we call a logical name, can then be further resolved, using mechanisms such as the Replica Location Service [14] to specific data object instances.

Metadata Interfaces

Figure 2 illustrates and categorizes the range of schema models that can support metadata publication and discovery. On the left side of the graph, we show a rigid, fixed schema and the associated API. In the fixed schema all the metadata are known and encoded ahead of time. This schema does not provide a very flexible data model. If the collaboration wants to modify the attribute set, the entire schema needs to be re-worked and a new set of access and discovery mechanisms may need to be provided. This restricted model may, however, facilitate data discovery, because the data model can be easily exposed.

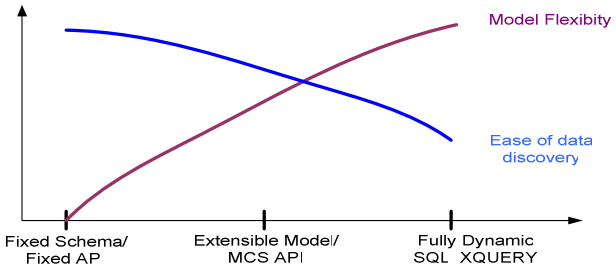

Figure 2: Classification of Metadata Model Flexibility and Ease of Data Discovery.

At the other end of the spectrum is a fully dynamic schema, where users may create data objects and attribute structures on the fly. This is a more general and flexible model that requires a very general interface. Such an interface must expose the internal structure of the

underlying database. This may make the process of discovery complex, especially for users who fall into the general public category.

Our interface hides implementation details from the end user. This provides us with flexibility in how we structure the database tables, in order to optimize performance. In Section 7 we explore alternative table layouts we designed to support a flexible attribute set. We believe that with our approach we are combining the benefits of ease of publication and discovery of the fixed schema model with some of the model flexibility of the fully dynamic schema.

We propose an API that allows for metadata publication, discovery and management of authorization. Publication includes creating and deleting logical objects: data items, collections and views. Attributes can be defined, undefined and set on all the logical objects. As part of publication, the content of a collection or view can be modified.

The API also supports a variety of metadata discovery methods. The API allows clients to discover the set of attributes defined within the Metadata Service and to search for logical objects based on these attributes. The attributes of a particular object can be retrieved. The API also supports discovery of the content of a collection or view. Parent collections and views of a particular data object can be found as well.

The API also provides the granting and revocation of authorization on data objects as well as the service itself. For example, the API supports authorizing users and user groups to define new data objects.

The data model and the API we described here are not a general solution. Rather, they support a set of functionality that can satisfy a class of applications that conform to the metadata publication, annotation and discovery requirements and scenarios we described in Section 2. This model is also limited in the way it handles attributes. Our attribute namespace has a flat structure and does not support more complex attribute structuring schemes.

5. Related Work

The Storage Resource Broker (SRB) from the San Diego Supercomputing Center [17] and its associated MCAT Metadata Catalog [5] provide metadata and data management services. SRB supports a logical name space that is independent of physical name space. The logical objects, logical files in the case of SRB, can also be aggregated into collections. SRB provides various authentication mechanisms to access metadata and data within SRB. Our Metadata Catalog Service model differs from MCAT in several ways. Perhaps most significantly, the architectural models of the two systems are fundamentally different. MCAT is implemented in tight integration with other components of SRB and is used to control data access and consistency as well as to store and query metadata. MCAT cannot be used as a stand-alone component. In addition, MCAT stores both logical metadata and physical metadata that characterizes file properties as well as attributes that describe resources, users and methods. By contrast, we have designed our Metadata Services to be one component in a layered, composable Grid architecture. We have factored this Grid architecture so that the Metadata Services contain only logical metadata attributes and appropriate handles that can be resolved by a data location or data access service.

There are a growing number of standards and formats in existence or under development for specifying metadata. Some of these are generic such as the Dublin Core whereas others are domain specific formats. For example, the Federal Geographic Data Committee has developed a standard [18] for the documentation of digital geospatial data. Encoded Archival Description [19] is a standard for finding inventories, registers, indexes, etc. The Data Documentation Initiative [20] has proposed a standard format for specifying metadata in the social and behavioral sciences. Metadata services and formats complement each other because while services such as MCS provide interfaces for storing and querying the metadata, formats provide a standard syntax for specifying metadata. We have kept our API independent of any particular format. Libraries can be written to convert metadata in standard formats so that the metadata can be stored and queried using MCS. For example, MCS can be used as the underlying engine for development of web portals such as NESSTAR[20] and metadata registries that facilitate data publication and discovery.

6. Implementation Issues

In our previous work [21], we presented an initial design of the Metadata Catalog Service (MCS) and reported performance numbers related to an implementation of the service based on the Apache web service [22]. The web service implementation lacked some desired functionality mentioned in Section 3, including GSI authentication, service element publication and notification.

In our previous study, we showed that layering a web service interface on top of a DBMS (in our case MySQL) results in an order of magnitude in performance degradation over accessing the database directly via ODBC. We measured the service performance in terms of add and query rates performed by multiple clients. In our current study, we continue to use relational technologies and evaluate the use of Grid services to support metadata management on the Grid.

We decided to use the DAI service as the basis for our MCS implementation. By leveraging off the significant development efforts of the EPCC group as well as the OGSA developers of the Globus team, we were able to

build upon a general purpose database access service with well-defined and extensible interfaces. We avoided having to implement our own Grid service to provide GSI authentication, lifetime management, etc.

The success of our MCS implementation is based on the extensible activity scheme of the DAI implementation [23]. Activities provide a way to add functionality that is not provided by DAI by allowing customized functions to be called in response to application-specific queries passed through the DAI interface. To layer MCS on top of DAI, we defined an MCS-specific activity that contains functions corresponding to the MCS API. This activity includes functions that can add, delete and query logical items, collections or views and their associated attributes.

This section describes implementation issues related to layering of MCS on top of the DAI service.

Extending DAI to support the MCS API

In order to add a new activity, an XML schema for the activity has to be specified and the implementation of the activity has to be provided. Currently the activities that can be performed over a data resource need to be specified in a configuration file prior to service initialization. The DAI Grid Service instances can execute the activities. We implemented a new activity called *mcsActivity* that includes the definition and implementation of the MCS API. The results returned by the activity are also in XML format. *mcsActivity* is implemented as a synchronous activity; thus the client blocks until the response is received.

Support for Authorization

DAI supports authorization by mapping users to database roles. The Distinguished Name (DN) from the certificate that the user presents for authentication identifies a user in MCS. If the user does not use any authentication in accessing the Grid Data Service, then the user is mapped to an anonymous DN. The union of the permissions granted to the anonymous DN and the user's DN is considered while evaluating the authorization for the user.

The DAI authorization model is useful in general but does not provide the granularity of control required by MCS. Thus, in MCS we implemented an authorization model that provides finer grained control at the level of logical objects. The following are various permissions that a user can have in MCS

- MCS create permission: This permission is required to create a logical item, collection or view. A user having this privilege can grant other users a similar privilege. Initially this permission has to be granted out of band to one user, who can then grant permissions to others.
- Write permission: A user can have write permission on a particular logical item, collection or view. Write permission on any object (logical item, collection, or view) allows the user to modify attributes of that object and to grant (or revoke) read or write permissions on that object to (or from) other users. In addition, write permission on a collection or view allows the user to add objects to and delete objects from that collection or view. Write permission on a collection also conveys write permission on all logical objects in the collection. The creator of a logical object is granted write permissions over it.
- Read permission: A user can have read permission on a particular logical item, collection or view. Read permission on an object allows the user to view the attributes of the object.

One feature of the authorization model is that permissions on a logical collection are also valid on the objects in the collection. Thus a user having write permissions over a collection automatically gets write permissions over the objects in the collection.

MCS Client-side Tools

We developed a wrapper around the DAI client code to expose a simple API interface. Each operation that can be invoked using *mcsActivity* is exposed as a method call in the MCS-API. We have also developed command line tools. When using the MCS-API, the user need not be aware of the syntax of the XML perform documents exchanged between the user and the MCS Grid Data Service. Using the DAI framework provides a very convenient mechanism for extending the MCS schema and executing arbitrary SQL operations, such ass creating new tables, over the extended schema. We have used this functionality to explore alternative implementations of the extensible schema.

Support for a Extensible Attribute Set

The key to representing the metadata is to capture common, domain-independent attributes while making it possible to add additional attributes that represent domain-specific metadata. One way to achieve this within our relational technology-based implementation is to create a basic table that contains the common attributes as table columns and then create additional tables for a fixed set of predefined attribute types. In one variation of our implementation, we support 6 different attribute types: String, Integer, Float, Date, Time, and DateTime. Each attribute-type table (static attribute table) contains three columns: object id, attribute name and attribute value. When attributes are added to an object, the name and value of the attribute are placed into the appropriate table along with an object id that identifies the row in the common attribute table to which this new attribute belongs. All attributes for an entry can then be found by searching all tables for entries with matching object identifiers.

This table set up is simple and allows for an easy addition of new attributes by adding rows to the table. However, as the size of the database grows into millions, the attribute

tables can grow large. For example, for a database size of 5 million logical items, if each logical item has 10 attributes, and 5 of them are of type string, then the string table will grow to 25 million rows. Obviously, searching such a large table can become inefficient.

A second variation of our implementation represents domain-specific attributes in individual tables (*dynamic attribute tables*). When a new attribute, (identified by a name and a type) is created, a new table is created. Subsequently, if the attribute is used to describe data objects, the value of the attribute and the corresponding data object are entered in the table. This approach considerably reduces the size of the tables. However, it increases the number of tables that must be searched to one per attribute (name, type) pair rather than one per attribute type. Nevertheless, this organization eliminates the need to do a join across multiple entries within a table as only one entry per object id will be found in each table. The drawback of this approach is that more tables will have to be stored in the database. Additionally, there is no efficient way to find all the attributes of a particular object, because all the dynamic attribute tables may need to be searched. As a result, we created an additional table that contains records consisting of the object id, attribute name and attribute type.

Next, we compare the performance of these two implementation variations.

7. Performance Study

With this study we aimed to address two issues:

1. How to most efficiently support a flexible schema with a variable number of attributes? Which approach, the static or dynamic attribute tables, provides efficient addition and deletion of attributes of a particular object as well as discovery of objects based on their attributes?
2. What overheads are being imposed by grid services vs. web services? Features such as authentication will likely require additional server-side processing by grid services.

To evaluate the alternative table designs, we performed a series of comparisons measuring the performance of MCS APIs with the two schemas. To address the second issue of web service versus Grid service overheads, we compare the two MCS implementations. We measured the performance of representative MCS APIs with both versions of MCS. In this case we used the fixed attribute table schema.

Experimental Setup

We experimented with databases of three sizes. For each size, we created logical collections with 1000 data items per collection. With each item, we associated 10 user-defined attributes of different types (string, float, integer, date and datetime), and in some cases we evaluated the performance of the system as a function of the number of attributes. Likewise, we associated 10 attributes with each collection. We loaded databases with a total of 100,000, 1,000,000, and 5,000,000 data items and their associated collections and attributes. Since we maintained a constant 1000 items per collection, there were 100 collections in 100,000 entry database, 1000 collections in the 1 million entry database, and 5000 collections in the 5 million entry database.

In the results below, we evaluate the performance of 4 critical metadata operations: add, simple query, complex query, and get user attributes. Add operations add a logical item with ten associated user-defined (dynamic) attributes of various types. To maintain the size of the database, we follow each add operation with a delete operation. Simple query operations do a value match for a single static attribute associated with a data item. The complex query operation does value matches for all ten user-defined attributes associated with a data item. Get user attributes returns all user-defined attributes of a data item. The rates of operations per second are measured at the client-side.

The MCS was installed on a dual-processor 2.2 GHz Intel Xeon workstation running RedHat Linux 8.0. The web-services-based MCS is built upon the Apache Jakarta Tomcat 4.1.24 server and the DAI based implementation uses DAI v. 3.1. Each implementation uses MySQL 3.23.49 relational database. We built indexes on item names, collection names and views (not used for these performance tests). We also built indexes on the database-assigned identifiers for these items and on (name,ID) pairs.

Optimizing and Supporting a Dynamic Attribute Set

Before presenting our performance results, we address the issue of the number of different attribute names used in our tests. We synthetically populated the attributes for the data objects, drawing names based on a uniform distribution over the set of names. The values were partially related to the names in case of string attributes, and we used the current date and time for the date time attributes. In the case of the static table schema (a table for each attribute type and object type), the number of different attribute names does not affect the size or number of the tables. However, for the dynamic attribute tables schem, the number of tables (one for each attribute name and type) varies based on the number of different attribute names. Obviously, the size of the tables is affected then as well. Based on our experience, we chose for our study sets of 1,000 and 5,000 different attribute names. Although climate modeling applications such as ESG, [24] and gravitational-wave physics [25] usually have fewer distinct attribute names, the larger attribute sets stress the dynamic attribute table approach.

Because a database of 5 Million entries showed the greatest sensitivity for complex queries in our earlier work, we use that size in evaluating schema implementations.

Figure 3 shows the performance of the add operation for the static and dynamic attribute table implementation with the 1,000 and 5,000 attribute namespaces. We varied the number of threads on a single host performing adds from 1

to 24. We measured the number of operations per second that could be sustained on the client-side. We notice that the static schema is on the average 1.7 times faster than the dynamic schema. In the latter case, each time a new attribute is added (up to 1,000 or 5,000 depending on the size of the attribute namespace) a new table is created. This table creation operation represents pure overhead in addition to adding a row into a table.

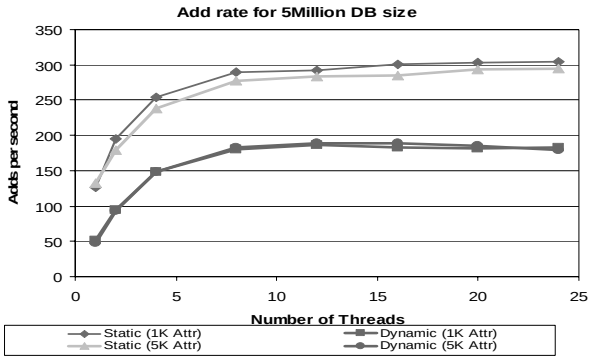

Figure 3: Performance of Add Operation. The Database Size is 5 Million Items. The number of client threads is varied.

Figure 4 shows the performance of simple queries as the number of threads on a single client host is varied from 1 to 12. The performance for both schemas is about the same. The static schema is on average 9% better for the 1,000 attribute namespace, whereas the dynamic schema is 3.5% better on average for the 5,000 attribute namespace.

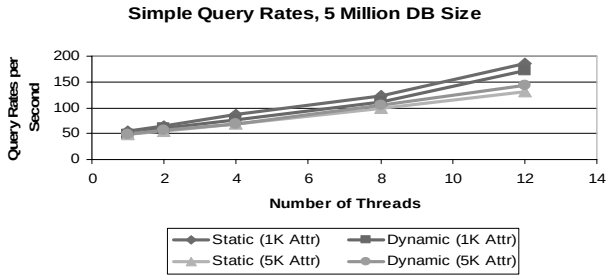

Figure 4: Simple Query Performance for the Static and Dynamic Schemas.

Figure 5 shows the complex query performance for the static and dynamic schemas. We notice that the dynamic solution is 1.5 times better on average than the static schema solution. This is because in the static schema, the operation needs to search for the attributes in very large attribute tables. In the dynamic case, there are more tables to be searched, but the tables themselves are smaller.

Figure 6 shows the performance of a query that returns all user-defined attributes of a given data item. For the static schema, this query accesses each attribute type table and matches the desired object name. For the dynamic schema, the query first searches the table that contains records consisting of the object id, attribute name and attribute type, and then queries individual attribute tables. The static solution performs 1.8 times better on average for the 1,000 attribute namespace and 2.7 times better for the 5,000 attribute namespace.

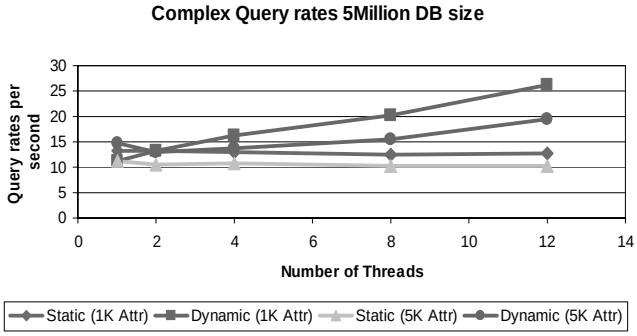

Figure 5: Complex Query Performance, measured in queries per second.

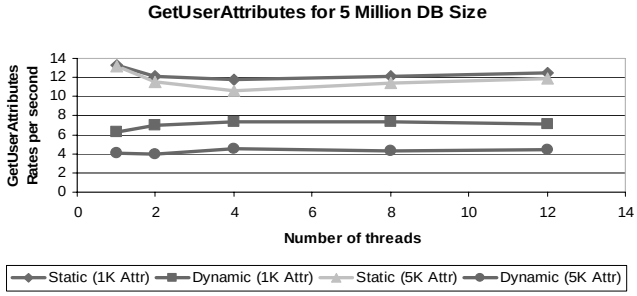

Figure 6: Performance of the Operation that returns all the user-defined attributes of a given item.

In summary, it is not obvious which schema, dynamic or static, is better. The performance of the dynamic approach depends on the number of unique attribute names. Results using 1,000 and 5,000 different attribute names show that the static schema generally performs better for add operations and for querying all user-defined attributes. The dynamic schema performs better for complex queries that match 10 attributes for an increasing number of requesting threads. The choice of schema may depend on the expected operation workload for the MCS. We assume that query operations will be the more frequent than add operations. Additionally, we speculate that value matching of desired attributes will occur more frequently than querying the attributes of a given data item. Under these assumptions, the dynamic schema solution would be beneficial, especially if there are not too many distinct attribute names.

Evaluating Grid Service versus Web Service Performance.

In this section we compare the performance of the MCS implementation layered on top of a standard web service and on top of DAI. In all the following experiments we have used the static schema. For this study we varied the size of the database from 100K to 1 and 5 Million. We also varied the number of hosts issuing the queries and adds. In all experiments each client host was running 4 concurrent threads. In general, because DAI does the parsing and

validation of the request document and the web service does not, we expect to see better performance from the latter.

The experimental setup for the tests consisted of 8 client machines and an MCS server machine connected to the same local area network. We increased the number of client hosts interacting with the MCS server from 1 to 8 in all our tests. Each client host was running 4 threads. For the DAI test, we had 8 grid data service instances running on the server for each database size. Each client host was assigned a particular grid data service instance that was shared by all the 4 threads running on that client.

Table 1 compares simple query rate performance. Since simple queries are very efficiently handled by the native database (as we have seen in out previous work [21]), this experiment clearly exposes the overhead of the services. We can see that the grid service performs on the average 5 times worse than the web service for all the database sizes.

Because complex queries are more costly than simple queries in terms of database performance, the differences between the web service and DAI are not as significant as in the simple query case. Also, as the size of the database increases, the average performance difference decreases (Table 2). For the 100K DB and 1 Million DB sizes, the web-service-based MCS is about 3 times faster on average than the DAI solution. Interestingly, for the 5 Million DB, DAI matches the web-service performance.

Table 1: Simple query rates per second for various DB sizes.

# of Hosts	100K items, web service	100K items, DAI	1Million items, web service	1Million items, DAI	5Million items, web service	5Million items, DAI
1	84.58	24.24	61	21	51.9	16.6
2	152.68	45.65	131.8	40.76	102.3	29.3
4	253.58	46.12	215.97	46.02	186.83	38.22
6	274.26	43	268.26	41.14	206	37.83
8	264	38.87	246	38.6	143	36.17

Table 2: Complex query rates per second for various DB sizes.

# of Hosts	100K items, web service	100K items, DAI	1Million items, web service	1Million items, DAI	5Million items, web service	5Million items, DAI
1	66.15	26.34	56.4	23.56	11.62	10.619
2	104.87	39.8	82.96	30.26	13	10.146
4	123.8	39.926	101.12	30.41	11.07	10.81
6	112.23	37.222	108	30.74	12.28	10.67
8	107.85	34.6	106	27.28	11.05	10.22

The final set of results compares add performance. Table 3 shows that the web-service-based MCS provides on average 3 times higher add rates than the DAI-based MCS.

Table 3: Add rate per second for various DB Sizes.

# of Hosts	100K items, web service	100K items, DAI	1Million items, web service	1Million items, DAI	5Million items, web service	5Million items, DAI
1	62.51	20.96	62.95	18.13	68.31	22.08
2	97.00	39.51	93.43	39.37	92.81	33.98
4	114.45	43.93	116.04	41.65	120.2	42.62
6	138.29	41.41	134.75	40.48	133.7	35.33
8	117.76	36.60	113.60	34.34	117.9	30.79

In summary, we see that DAI performs worse than a web-service, although the implementations are not directly comparable, since DAI does parsing and validation of the request document. Also, we perform authorization in the DAI version of MCS, unlike the web service version. It is interesting to note that DAI performance does not degrade as the number of clients increases, indicating good scalability. When we first started comparing DAI to web-based services, the overheads of the DAI v. 3.0.2 were in some cases an order of magnitude worse than those of the web service (for example for simple queries). The current release of DAI (3.1) used in this work has significant performance improvements relative to 3.0.2, and performance may continue to improve as the DAI service matures.

8. Application Use Cases

We used the web service-based MCS in several applications. We are currently transitioning to the use of the DAI implementation. MCS has been used in Earth System Grid (ESG) application [26], in the Pegasus workflow management system [27, 28], and others.

ESG is a climate modeling application. MCS was used as one component in an ESG demonstration that included replica and storage management services as well as various data storage services. The ESG metadata followed the netCDF convention and was generated in XML format. To store ESG metadata in MCS, we added user-defined attributes to the MCS corresponding to application-specific ESG metadata attributes as well as Dublin Core attributes.

Pegasus [27] is a planning component developed within the GriPhyN project (www.griphyn.org) [25]. Pegasus is used to map complex application workflows onto the available Grid resources. Pegasus uses MCS to discover existing application data products. When the Pegasus planner receives a user request to retrieve data with particular metadata attributes, it queries the MCS to find all

logical files with the corresponding properties. When the workflow generated by the Pegasus planner results in the creation of new application data products, Pegasus uses the MCS to record metadata attributes associated with those newly materialized data products.

MCS is also used by Pegasus to store provenance and performance information about the workflow components that have been executed on the Grid and up-to-date information about a workflow being executed. The information describes the executables used in the workflow execution and the time taken to execute the workflow components, including the time to perform data movement. MCS also provides users various levels of detail regarding a set of workflows or particular workflow instances. As such, MCS is used in conjunction with the Pegasus portal by the Montage application [30, 31], an astronomy application that delivers science grade mosaics of the sky on demand.

9. Conclusions and Future Work

In this paper we described MCS, a Metadata Catalog Service for metadata management on the Grid. We described the requirements that have driven the design of MCS. We presented the data model, the authorization model and the API used to interact with MCS. We discussed alternative schema designs that can support a dynamic user-defined attribute set. Finally, we evaluated the performance of two alternative schemas and the overhead imposed by a grid service-based implementation in comparison to a web service-based version.

In this work we focused on a centralized metadata service design. However, in distributed systems, it is often necessary to distribute services to provide reliability and good performance. We are currently investigating the feasibility of distributing MCS, exploring issues of federation of multiple services in a Grid environment. We are studying the use of query mediation and planning techniques combined with ontology-based attribute models to query across multiple MCS instances [32].

Acknowledgements

This research was supported by NSF under grants ITR-0086044(GriPhyN) and ITR AST0122449 (NVO) and the DOE Cooperative Agreements: DE-FC02-01ER25449 (SciDAC-DATA) and DE-FC02-01ER25453 (SciDAC-ESG.)

References

[1] B. C. Barish and R. Weiss, "LIGO and the Detection of Gravitational Waves," *Physics Today*, vol. 52, pp. 44, 1999.

[2] C.-E. Wulz, "CMS - Concept and Physics Potential," Proceedings II-SILAFAE, San Juan, Puerto Rico, 1998.

[3] "NVO," 2004. http://www.us-vo.org/

[4] I. Foster, et al., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International Journal of High Performance Computing Applications*, vol. 15, 2001.

[5] MCAT, "MCAT - A Meta Information Catalog (Version 1.1)." http://www.npaci.edu/DICE/SRB/mcat.html

[6] "Flexible Image Transport System." http://fits.gsfc.nasa.gov/

[7] I. Foster, et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.," Open Grid Service Infrastructure WG, GGF 2002.

[8] "Global Grid Forum " 2004. www.globalgridforum.org

[9] "TeraGrid." http://www.teragrid.org/

[10] "International Virtual Data Grid Laboratory." www.ivdgl.org

[11] V. Welch, et al., "Security for Grid Services," HPDC-12, 2003.

[12] K. Czajkowski, et al., "A Resource Management Architecture for Metacomputing Systems," in *4th Workshop on Job Scheduling Strategies for Parallel Processing*: 1998.

[13] W. Allcock, et al., "Data Management and Transfer in High-Performance Computational Grid Environments," *Parallel Computing*, 2001.

[14] A. Chervenak, et al., "Giggle: A Framework for Constructing Sclable Replica Location Services.," SC2002.

[15] "Globus Toolkit 3." http://www.globus.org/ogsa/

[16] V. Raman, et al., "Data Access and Management Services on Grid," GGF, DAIS group 2002.

[17] C. Baru, et al., "The SDSC Storage Resource Broker," Proceedings of Proc. CASCON'98 Conference, 1998.

[18] "Content Standard for Digital Geospatial Metadata." http://www.fgdc.gov/metadata/contstan.html

[19] "Encoded Archival Description (EAD)." http://www.loc.gov/ead/eaddev.html

[20] "Norwegian Social Science Data Services: Providing Global Access to Distributed Data Through Metadata Standardisation" *Working Paper No. 10, UN/ECE Work Session on Statistical Metadata*,1999.

[21] G. Singh, et al., "A Metadata Catalog Service for Data Intensive Applications," Proceedings of SC 2003.

[22] "Apache." http://xml.apache.org/

[23] N. Hardman, et al., "OGSA-DAI: A look under the hood: Part 2: Activities and results," 2004. http://www-106.ibm.com/developerworks/grid/library/gr-ogsadai2/

[24] I. Foster, et al., "The Earth System Grid II: Turning Climate Datasets Into Community Resources," Proceedings of Annual Meeting of the American Meteorological Society, 2002.

[25] E. Deelman, et al., "GriPhyN and LIGO, Building a Virtual Data Grid for Gravitational Wave Scientists," HPDC 2002.

[26] ESG, "The Earth Systems Grid." www.earthsystemsgrid.org

[27] E. Deelman, et al., "Pegasus: Mapping Scientific Workflows onto the Grid," Proc.2nd EU Across Grids Conf., Cyprus, 2004.

[28] E. Deelman, et al., "Workflow Management in GriPhyN," in *Grid Resource Management*, Kluwer, 2003.

[29] E. Deelman, et al., "Pegasus: Planning for Execution in Grids," GriPhyN 2002-20, 2002.

[30] R. Williams, et al., "Multi-wavelength image space: another Grid-enabled science," *Concurrency and Computation: Practice and Experience*, vol. 15, pp. 539-549, 2003.

[31] B. Berriman, et al., "Montage: A Grid-Enabled Image Mosaic Service for the NVO," ADASS XIII, 2003.

[32] R. Tuchinda, et al., "Artemis: Integrating Scientific Data on the Grid," IAAI, San Jose, California, 2004 (to appear).